Gorka Bonals Sastre

Projecte de Fi de Carrera
Enginyer Industrial

Planificació Probabilística de Trajectòries per a Robots Cooperants

MEMÒRIA

Autor: Gorka Bonals Sastre
Director: Lluís Ros Giralt
Ponent: Joan Rosell Gratacós
Convocatòria: Setembre 2005 (pla 94)

Enginyer Industrial

Setembre 2005 (pla 94)

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

L'objectiu d'aquest projecte és desenvolupar un planificador de trajectòries per a un sistema robotitzat formant una cadena cinemàtica tancada. El sistema està format per dos robots manipuladors Stäubli RX60 que han de transportar un objecte subjectat simultàniament a les seves pinces.

En una primera fase s'ha estudiat quin era l'algorisme més adequat per desenvolupar el planificador, i al final s'ha vist que un mètode probabilístic era el que s'ajustava més a les nostres necessitats.

Les solucions generades són també força curtes quant a la distància recorreguda en l'espai de configuracions, i s'obtenen de forma gairebé instantània una vegada s'ha generat el mapa de rutes de l'entorn de treball, satisfent així els requeriments inicials. S'han aconseguit resoldre dos problemes de notable complexitat que així ho demostren

Finalment, cal dir que hem desenvolupat una interfície gràfica que té un ús prou còmode, i permet definir i resoldre nous problemes de planificació per usuaris que no tinguin massa coneixements de planificació.

Índex

I Memòria **ix**

1 Introducció **1**
1.1 Objectius . . . 1
1.2 Motivació . . . 1
1.3 Abast del projecte . . . 3
1.4 Estructura de la memòria . . . 4

2 L'espai de configuracions **5**
2.1 Problema bàsic de la planificació de moviments . . . 5
2.2 Formulació en l'espai de configuracions . . . 6
2.2.1 Noció d'espai de configuracions . . . 7
2.2.2 Obstacles en l'espai de configuracions . . . 8
2.2.3 Noció de trajectòria . . . 9
2.2.4 Restriccions cinemàtiques . . . 10

3 Antecedents **15**
3.1 Descomposició cel·lular . . . 15
3.1.1 Descomposició exacta . . . 16
3.1.2 Descomposició aproximada . . . 17
3.2 Camp de potencials . . . 18
3.3 Mapes de rutes . . . 21
3.3.1 Grafs de visibilitat . . . 21
3.3.2 Mètodes basats en retracció . . . 23
3.3.3 Mapes de siluetes . . . 23
3.3.4 Mapes probabilístics . . . 24
3.4 Discussió . . . 25

4 Planificació basada en mapes probabilístics **27**
4.1 La fase d'aprenentatge . . . 27
4.1.1 L'etapa de construcció . . . 28
4.1.2 L'etapa d'expansió . . . 32
4.2 La fase d'interrogació . . . 35
4.3 Completesa probabilística . . . 36

5 Mapes probabilístics per cadenes cinemàtiques tancades **41**

5.1 Satisfacció d'equacions de clausura . . . 41
5.2 L'espai de configuracions . . . 44
5.3 Estratègies de planificació . . . 44
5.3.1 Mètode de Yakey, Lavalle i Kavraki . . . 44
5.3.2 Mètode de Han i Amato . . . 47
5.4 Implementació per a dos manipuladors simples . . . 48
5.4.1 Sistema considerat . . . 48
5.4.2 Etapa de construcció . . . 48
5.4.3 Etapa d'expansió . . . 52
5.4.4 Fase d'Interrogació . . . 53
5.4.5 Fase de Suavització . . . 54

6 Resultats experimentals 57
6.1 Modelització dels elements . . . 57
6.1.1 Webots . . . 57
6.1.2 Passos per modelitzar els robots RX60 . . . 59
6.2 Introducció dels experiments . . . 59
6.3 Experiment 1 . . . 60
6.3.1 Introducció . . . 60
6.3.2 Resultats de l'aprenentatge . . . 60
6.3.3 Resultats de les interrogacions . . . 62
6.3.4 Una trajectòria solució . . . 62
6.4 Experiment 2 . . . 66
6.4.1 Introducció . . . 66
6.4.2 Resultats de l'aprenentatge . . . 66
6.4.3 Resultats de les interrogacions . . . 70
6.4.4 Una trajectòria solució . . . 71

7 Conclusions i treball futur 77

8 Agraïments 79

II Annexos 1

A Posició i orientació d'un sòlid rígid 3
A.1 Descripció de la posició . . . 3
A.2 Descripció de l'orientació . . . 3
A.3 Transformacions homogènies . . . 5

B Anàlisi posicional d'un robot manipulador 7
B.1 Paràmetres geomètrics . . . 7
B.2 Cinemàtica directa . . . 7
B.3 Cinemàtica inversa . . . 7
B.4 Paràmetres de Denavit-Hartenberg . . . 8

C Anàlisi del robot manipulador Stäubli RX60 **13**
C.1 Característiques tècniques . . . 13
C.2 Cinemàtica directa d'un manipulador simple . . . 13
C.3 Cinemàtica inversa d'un manipulador simple . . . 16
C.4 Solucions del RX60 . . . 20
C.4.1 Paràmetres de Denavit-Hartenberg . . . 20
C.4.2 Rang de treball dels angles . . . 21
C.4.3 Solucions angulars . . . 22

D Anàlisi posicional de l'objecte a transportar **23**
D.1 Sistemes de referència . . . 23
D.2 Cinemàtica directa . . . 23
D.3 Cinemàtica inversa . . . 24

E Manual d'usuari **27**
E.1 Com utilitzar el manual . . . 27
E.2 Inici . . . 27
E.3 Descripció de les finestres . . . 27
E.3.1 Graus de llibertat . . . 27
E.3.2 Paràmetres . . . 29
E.3.3 Tasques . . . 31
E.3.4 Anàlisi del graf . . . 32
E.4 Guia per l'especificació d'un problema . . . 34
E.4.1 Inserir els obstacles . . . 34
E.4.2 Inserir l'objecte . . . 35
E.4.3 Actualitzar el fitxer de col·lisions . . . 35
E.4.4 Fixar la posició i orientació dels robots . . . 36
E.4.5 Fixar la prensió de l'objecte . . . 37
E.5 Guia per resoldre un problema . . . 38
E.5.1 Definir els paràmetres . . . 38
E.5.2 Executar la fase d'aprenentatge . . . 38
E.5.3 Triar les configuracions inicial i final . . . 38
E.5.4 Executar la interrogació . . . 38
E.5.5 Tipus de fitxers emprats . . . 38

F Anàlisi descendent de l'algorisme **41**
F.1 Introducció . . . 41
F.2 Pseudocodi del programa principal . . . 41

G Documentació dels mòduls **53**
G.1 Introducció . . . 53
G.2 Mòduls principals . . . 53
G.2.1 graf_config.h . . . 53
G.2.2 configuracio.h . . . 57
G.2.3 robots.h . . . 65
G.2.4 objecte.h . . . 71

G.2.5 vector.h . . . 75
G.3 Mòduls secundaris . . . 78

H Codi font **83**
H.1 main.c . . . 83
H.2 globals.h . . . 84
H.3 globals.c . . . 85
H.4 interface.h . . . 85
H.5 interface.c . . . 86
H.6 config.h . . . 110
H.7 support.h . . . 111
H.8 support.c . . . 111
H.9 callbacks.h . . . 113
H.10 callbacks.c . . . 118
H.11 graf_config.h . . . 143
H.12 graf_config.c . . . 145
H.13 configuracio.h . . . 168
H.14 configuracio.c . . . 172
H.15 robots.h . . . 182
H.16 robots.c . . . 185
H.17 objecte.h . . . 188
H.18 objecte.c . . . 190
H.19 lib_colisions.c . . . 192
H.20 tau_colisions.h . . . 194
H.21 tau_colisions.c . . . 194
H.22 colisio.h . . . 194
H.23 colisio.c . . . 195
H.24 component.h . . . 195
H.25 component.c . . . 195
H.26 Gtypes.h . . . 196
H.27 fitxer.h . . . 198
H.28 fitxer.c . . . 198
H.29 glr.h . . . 206
H.30 glr.c . . . 207
H.31 glrtools.h . . . 216
H.32 glrtools.c . . . 216
H.33 Gmotion.c . . . 218
H.34 Gutil.c . . . 223
H.35 graf.h . . . 224
H.36 llr.h . . . 224
H.37 llr.c . . . 224
H.38 llrtools.h . . . 228
H.39 noms_gll.h . . . 228
H.40 traject.h . . . 228
H.41 traject.c . . . 229

H.42 transf_ctnt.h . . . 232
H.43 transf_ctnt.c . . . 233
H.44 transf_homog_ctnt.h . . . 233
H.45 transf_homog_ctnt.c . . . 233
H.46 transf_homogenies.h . . . 235
H.47 transf_homogenies.c . . . 235
H.48 utils.h . . . 238
H.49 vector.h . . . 238
H.50 vector.c . . . 240
H.51 vei.h . . . 242
H.52 vei.c . . . 242
H.53 vertex_exp.h . . . 243
H.54 vertex_exp.c . . . 245
H.55 webots_interface.h . . . 246
H.56 webots_interface.c . . . 248
H.57 utils.c . . . 250
H.58 wgraph.h . . . 251
H.59 wgraph.c . . . 251
H.60 tau_list.h . . . 254
H.61 tau_list.c . . . 255
H.62 Gdata.h . . . 255
H.63 generals.h . . . 255
H.64 ctnt_generals.h . . . 256
H.65 funcions_generals.h . . . 257
H.66 funcions_generals.c . . . 257
H.67 macros_generals.h . . . 258

Part I

Memòria

Capítol 1

Introducció

1.1 Objectius

L'objectiu principal d'aquest projecte és desenvolupar un planificador de trajectòries per a dos robots RX60 que han de transportar un objecte subjectat simultàniament per les seves dues pinces.

El planificador ha de ser capaç de calcular la trajectòria que han de seguir els dos robots per tal de desplaçar l'objecte entre dues configuracions donades, mantenint la subjecció simultània en tot moment, i evitant col·lisionar amb obstacles de l'entorn.

1.2 Motivació

Una de les necessitats fonamentals de la Robòtica és disposar d'algorismes que permetin convertir la descripció a alt nivell d'una tasca en una seqüència detallada de comandes que permetin executar-la. La planificació de trajectòries intenta cobrir una d'aquestes necessitats: la generació automàtica de trajectòries per desplaçar els robots d'un punt a un altre de l'espai de treball.

Una versió clàssica d'aquest problema és l'anomenat *problema dels transportistes de pianos*: suposant que es coneix amb precisió el model geomètric d'una casa, determinar la seqüència de moviments necessaris per dur-lo des d'una habitació a una altra, sense col.lisionar amb els obstacles de l'entorn. Tot i que aquest cas pot semblar poc habitual, problemes de naturalesa semblant sorgeixen amb molta freqüència en l'entorn industrial, on la necessitat de disposar d'un sistema que automàticament planifiqui els recorreguts és més palesa, donada la quantitat d'obstacles amb que hom es troba, i la complexitat de l'espai de treball que d'això se'n deriva. Un exemple d'aquesta complexitat el trobem a la industria de l'automoció, en l'automatització de tasques de soldadura i muntage en cel.les de fabricació robotitzades, on cal planificar acuradament les trajectòries de tots els robots de manera que mai arribin a col.lisionar entre ells o amb l'entorn (Figura 1.1). Problemes semblants apareixen sovint en la industria naval, siderúrgica o aeronàutica.

En les darreres dècades s'ha avançat molt en la resolució d'aquests problemes, primer fent un èmfasi especial en la generació de trajectòries cinemàticament vàlides, i després incorporant als planificadors aspectes com la dinàmica dels cossos en moviment, la in-

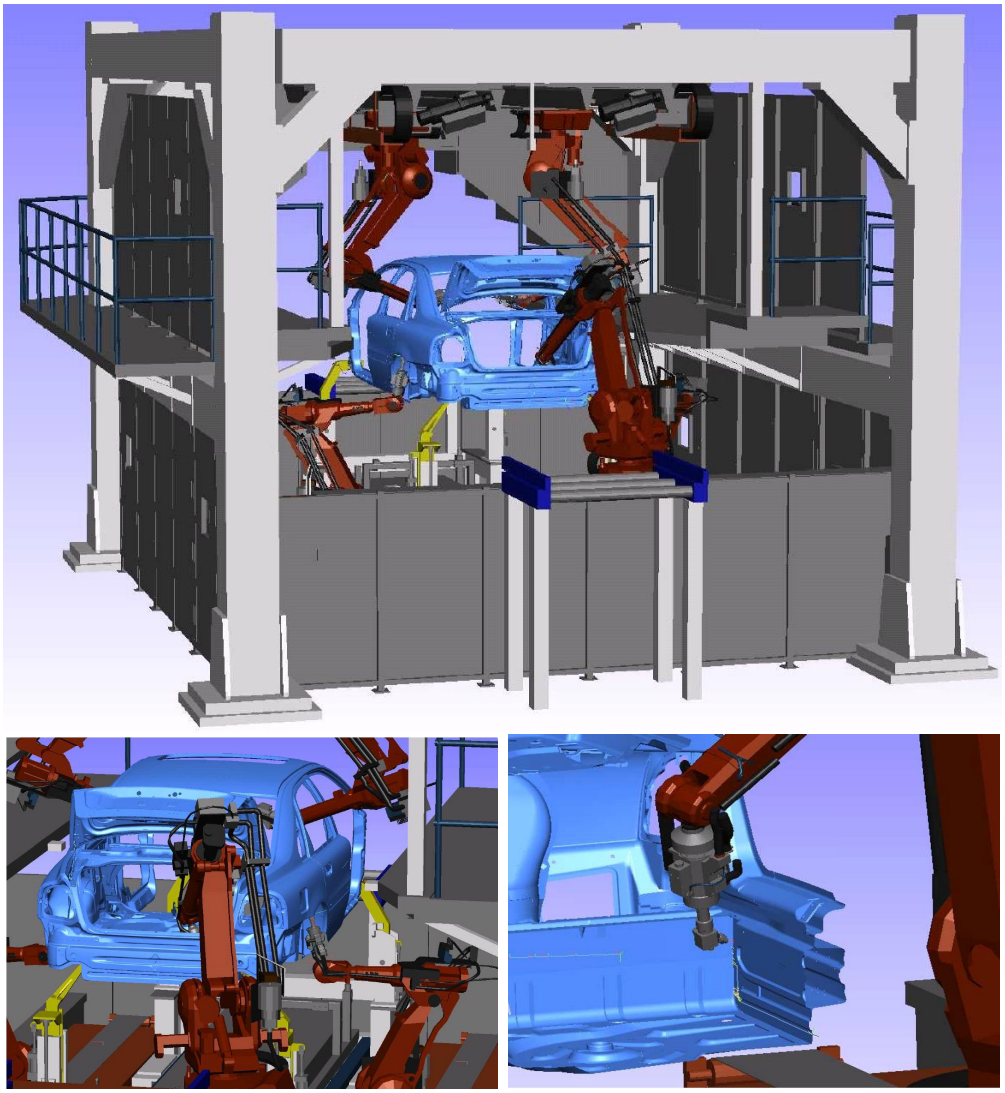


Figura 1.1: Planificació de trajectòries en una cel.la de soldadura a la industria de l'automoció: Cal planificar acuradament les trajectòries de tots els manipuladors evitant en tot moment la col.lisió amb obstacles de l'entorn. Figura adaptada de [21].

certesa dels models assumits, o la cerca de solucions que optimitzin algun criteri. El gruix de treball realitzat fins ara, tanmateix, s'ha concentrat en l'obtenció d'algorismes només vàlids per sistemes robotitzats que no presentin cadenes cinemàtiques tancades. És a dir, sistemes on sigui quin sigui el moviment efectuat, mai cal mantenir la clausura cinemàtica d'una cadena cíclica de sòlids, articulats dos a dos entre sí. Tals cadenes apareixen, per exemple, en la situació mostrada a la Figura 1.2, on degut al pes excessiu de l'objecte a transportar, els dos robots l'han de manipular coordinadament per dur-lo des d'un punt a un altre de l'espai de treball. La programació punt a punt d'aquesta tasca resultaria molt difícil, o com a mínim altament tediosa i subjecte a possibles errors, ja que caldria garantir la premsió simultània de l'objecte en tot moment. Aquest projecte pretén desenvolupar un planificador que sí permeti aquest tipus de manipulacions, per a un sistema format

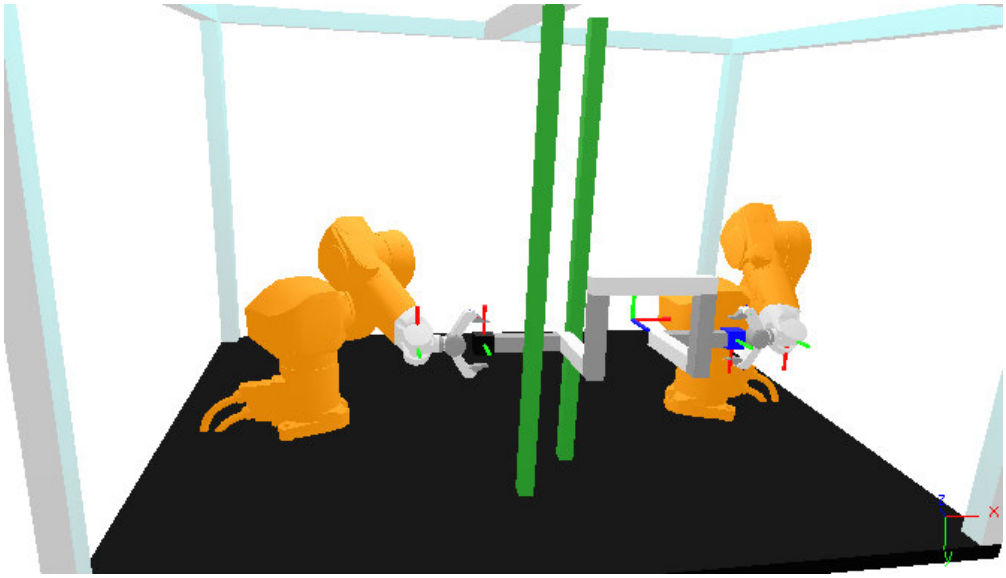


Figura 1.2: Manipulació coordinada d'un objecte. En tot moment cal mantenir tancada la seqüència cíclica de sòlids formada pels dos robots, l'objecte transportat, i el terra.

per dos robots Stäubli RX60 instal.lats a la cel.la robotitzada experimental de l'Institut de Robòtica i Informàtica Industrial de Barcelona.

1.3 Abast del projecte

El planificador desenvolupat disposarà de funcionalitat per:

- Especificar el problema a resoldre, definit per un conjunt de obstacles, l'objecte a transportar, i les posicions inicial i final del mateix.
- Introduir i manipular els models geomètrics de tots els sòlids en joc, permeten la seva simulació cinemàtica, i la detecció de col·lisions entre ells.
- Planificar la trajectòria articular que han de seguir els dos robots per transportar l'objecte des de la posició inicial a la final, subjectant-lo simultàniament.
- Simular cinemàticament la trajectòria obtinguda, mostrant-la en un entorn de visualització 3D.

El projecte es desenvoluparà tenint en compte les següents hipòtesis i requisits:

- S'assumirà que els dos robots tenen la seva base fixada al terra i que els obstacles de l'entorn són estàtics i de posició i geometria totalment conegudes. Aquesta hipòtesi permet l'ús d'estratègies de planificació basades en l'aprenentatge *off-line* de l'entorn.

- Tant l'objecte a transportar com els obstacles de l'entorn podran tenir geometria arbitraria, però s'assumirà que es disposa sempre de models polièdrics prou acurats de les seves superfícies.

- En la mesura que sigui possible, es procurarà obtenir trajectòries que minimitzin el desplaçament de les articulacions dels robots. A causa de la complexitat del problema, no es requerirà que les trajectòries siguin òptimes en aquest sentit, però si prou bones.

- Com a sortida, el planificador proporcionarà la trajectòria que cada robot a de seguir, parametritzant els angles de les seves sis articulacions en funció del temps.

- El planificador haurà de disposar d'una interfície gràfica que faciliti la definició dels problemes a resoldre, així com la visualització i anàlisi de les trajectòries calculades. Es procurarà implementar aquesta interfície de manera que, en un futur, si cal, es pugui substituir la llibreria de gestió gràfica en que es basi per alguna altre que resulti més convenient.

1.4 Estructura de la memòria

En el capítol 2 s'introdueix el problema de la planificació de moviments, presentant una versió simplificada del mateix, i es veu com es tradueix al formalisme de l'espai de configuracions, que permet fer-ne un tractament matemàtic rigurós.

El capítol 4 descriu diversos mètodes de planificació i en la darrera secció es justifica quin s'adequa més als requeriments del present projecte.

Els capítols 4 i 5 constitueixen el nucli de la memòria. En el primer d'ells s'explica amb detall el mètode de planificació que s'ha seleccionat i en el segon el cas d'un sistema format per cadenes cinemàtiques tancades, i més concretament la cadena que formen dos robots RX60 transportant un objecte subjectat per les seves pinces.

Finalment en el capítol 6 es resolt el problema de planificació per a dos experiments de complexitat considerable.

Capítol 2

L'espai de configuracions

En aquest capítol exposarem els trets característics de tot problema de planificació de moviments. Ho farem tot partint d'una versió simplificada del problema, l'anomenat *problema bàsic*. Això facilitarà al lector la comprensió de problemes de planificació de més complexitat, com el de la planificació de robots cooperants, que és l'objecte d'aquest projecte. Per altra banda, introduirem la noció d'espai de configuracions, que permetrà fer un plantejament matemàtic rigorós del problema i simplificar la seva anàlisi. En aquest espai, a més, molts problemes de planificació que són diferents en termes de geometria i cinemàtica queden traduïts a una formulació comuna.

2.1 Problema bàsic de la planificació de moviments

Hom considera la següent com la formulació més bàsica del problema de la planificació de moviments:

> Sigui $\mathcal{A}$ un sòlid rígid movent-se en un espai Euclidià $\mathcal{W}$, representat com $\mathbb{R}^N$, amb $N = 2$ o 3. El sòlid $\mathcal{A}$ s'anomena *robot*, i l'espai $\mathcal{W}$ s'anomena *espai de treball*.
>
> Sigui $\mathcal{B}_1, ..., \mathcal{B}_q$ una col·lecció de sòlids rígids fixats i distribuïts en $\mathcal{W}$. Els $\mathcal{B}_i$ s'anomenen *obstacles*.
>
> Assumim que la geometria del robot $\mathcal{A}$ i dels obstacles $\mathcal{B}_1$,...,$\mathcal{B}_q$ és coneguda, així com la localització dels segons dins l'espai $\mathcal{W}$.
>
> També fem la hipòtesi de que no tenim restriccions en el moviment del robot $\mathcal{A}$. Es diu que $\mathcal{A}$ és un robot *lliure-volant* perquè en absència d'obstacles pot ocupar qualsevol posició i orientació dins l'espai de treball, i pot desplaçar-se amb qualsevol velocitat.
>
> El problema a resoldre és el següent (figura 2.1): Donada una posició i orientació inicials i una posició i orientació finals de $\mathcal{A}$ en $\mathcal{W}$, trobar un algorisme que generi una *trajectòria* τ especificant una seqüència contínua de posicions i orientacions de $\mathcal{A}$ que eviti col·lisions amb els obstacles $\mathcal{B}_i$, comenci a la posició i orientació inicials, i acabi a la posició i orientació finals. En cas que no existeixi tal trajectòria, l'algorisme ho haurà de detectar.

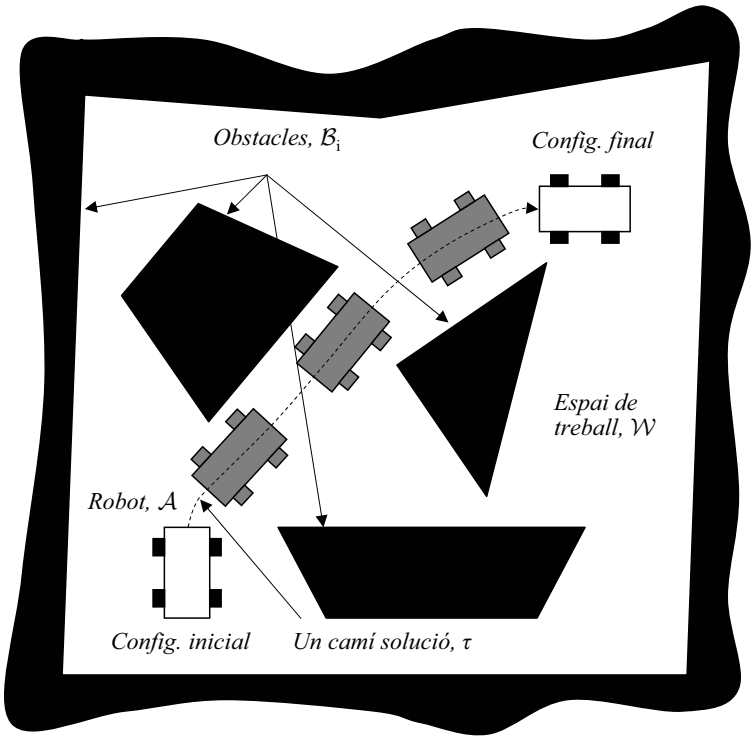


Figura 2.1: Elements del problema bàsic de la planificació de moviments.

La formulació bàsica anterior és una simplificació substancial del problema de la planificació de moviments. Per exemple, s'han ignorat les propietats dinàmiques del robot, no es consideren com a possibles els moviments amb contacte robot-obstacle, i s'assumeix que el robot és un únic sòlid, i que aquest és rígid. No obstant això, veurem que es tracta malgrat tot d'un problema difícil, d'interès pràctic, i que algunes de les solucions que s'hi han donat són aplicables a problemes més complexos com el que ens ocupa en el present projecte.

Encara que la formulació anterior sembli clara, hi ha aspectes que han de ser precisats. Una manera més rigorosa d'estudiar el problema, és a partir del que és conegut com a formulació en l'espai de configuracions, que repassarem tot seguit.

2.2 Formulació en l'espai de configuracions

La idea fonamental d'aquesta formulació és representar el robot com un punt en un espai $\mathcal{C}$, anomenat *l'espai de configuracions* del robot, i projectar els obstacles en aquest nou espai. Aquest plantejament transforma el problema de planificar el moviment d'un robot que es mou en $\mathcal{W}$, en el problema de planificar el moviment d'un punt que es mou en $\mathcal{C}$. La dimensió d'aquest espai, com veurem, correspon al nombre de graus de llibertat del robot.

2.2.1 Noció d'espai de configuracions

Introduirem aquí la noció d'espai de configuracions per al problema bàsic. La seva extensió al cas tractat en aquest projecte s'introduirà a la secció 5.2.

Sigui $\mathcal{A}$ el robot que volem desplaçar dins d'un espai de treball $\mathcal{W} = \mathbb{R}^N$, $N = 2$ o 3, i siguin $\mathcal{B}_1, ..., \mathcal{B}_q$ els obstacles a considerar dins de $\mathcal{W}$. Anomenem $\mathcal{F}_\mathcal{A}$ i $\mathcal{F}_\mathcal{W}$ els sistemes de referència cartesians associats a $\mathcal{A}$ i $\mathcal{W}$ respectivament. $\mathcal{F}_\mathcal{A}$ és una referència que es mou, ja que el robot té associats graus de llibertat, mentre que $\mathcal{F}_\mathcal{W}$ és a una posició fixa. Per definició, com que $\mathcal{A}$ és un sòlid rígid, cada punt a de $\mathcal{A}$ té una posició fixa respecte d' $\mathcal{F}_\mathcal{A}$. Però la posició d'a en $\mathcal{W}$ depèn de la posició i orientació d' $\mathcal{F}_\mathcal{A}$ relativa a $\mathcal{F}_\mathcal{W}$.

Una *configuració* q de $\mathcal{A}$ és una especificació de la posició $\mathcal{T}$ i orientació Θ d' $\mathcal{F}_\mathcal{A}$ respecte $\mathcal{F}_\mathcal{W}$. L'espai de configuracions de $\mathcal{A}$, denotat per $\mathcal{C}$, és l'espai de totes les configuracions de $\mathcal{A}$ (figura 2.2). El subconjunt de $\mathcal{W}$ ocupat per $\mathcal{A}$ a la configuració q és denotat per $\mathcal{A}(q)$. De la mateixa manera, el punt a de $\mathcal{A}$ a la configuració q és denotat per $a(q)$ en $\mathcal{W}$.

La configuració q de $\mathcal{A}$ s'especifica normalment mitjançant un vector d'm paràmetres reals. La tria dels paràmetres més adequats no és sempre senzilla i cal tenir en compte aspectes com la possible dependència dels mateixos, o la no injectivitat de la parametrització resultant.

Pel que fa al primer aspecte, per exemple, si hom descriu la posició $\mathcal{T}$ pel vector d'N coordenades de l'origen d'$\mathcal{F}_\mathcal{A}$ respecte $\mathcal{F}_\mathcal{W}$, i l'orientació Θ mitjançant una matriu $N \times N$, de canvi de base entre $\mathcal{F}_\mathcal{W}$ i $\mathcal{F}_\mathcal{A}$, llavors q és un vector de $N(N+1)$ paràmetres que, clarament, no són independents, ja que la matriu que descriu Θ ha de tenir les

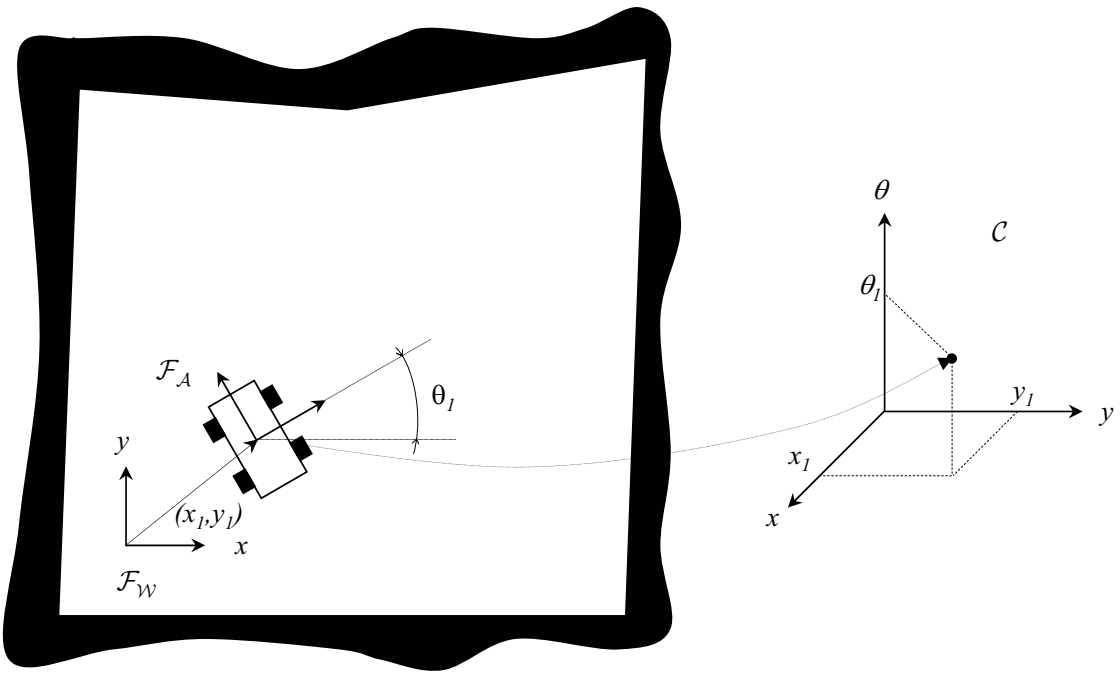


Figura 2.2: Generació de l'espai de configuracions per un robot mòbil en el pla.

columnes ortonormals i determinant igual a 1. En aquest cas, $\mathcal{C}$ és només un subconjunt de $\mathbb{R}^{N(N+1)}$ de mesura nul·la i, com veurem, això complica la resolució del problema bàsic de planificació.

Pel que fa al segon aspecte, hom podria pensar en descriure Θ per un vector de mínim nombre de paràmetres, com per exemple un angle θ, si $N = 2$ (per exemple, l'angle entre l'eix x de $\mathcal{F}_\mathcal{W}$ i $\mathcal{F}_\mathcal{A}$), i tres angles ϕ, θ, ψ, si $N = 3$ (per exemple, els angles d'Euler). El problema és que aquesta descripció no és injectiva, ja que una mateixa orientació pot ser descrita per diferents vectors d'angles. Un cas senzill és el que sorgeix en dues dimensions ($N = 2$), on dos angles en els que la seva diferència és un múltiple de 2π representen la mateixa orientació θ. En aquest cas el problema pot ser fàcilment solventat restringint el valor de l'angle θ a l'interval $[0,2\pi)$. En el cas 3-dimensional ($N = 3$) la dificultat és lleugerament més complicada de resoldre, però limitant els angles d'Euler als intervals $\phi \in [0, 2\pi]$, $\theta \in [0, \pi)$ i $\psi \in [0, 2\pi)$, i tenint en compte la singularitat que apareix per $\theta = 0$, aquest problema es pot solventar [19, pàg. 71].

Per acabar, podem dir que escollint una parametrització adient [19, pàg. 68] es pot descriure una configuració mitjançant un vector d'm paràmetres independents, amb $m = 3$ (si $N = 2$) i $m = 6$ (si $N = 3$).

2.2.2 Obstacles en l'espai de configuracions

En la definició de l'espai de configuracions s'ha considerat que no hi ha obstacles en l'espai de treball. Per tant, falta caracteritzar el subespai de punts de l'espai de configuracions que corresponen a situacions de col·lisió del robot amb els obstacles.
Cada obstacle $\mathcal{B}_i$ de l'espai de treball $\mathcal{W}$ es representa en $\mathcal{C}$ com una regió:

$$\mathcal{C}_{\mathcal{B}_i} = \{q \in \mathcal{C} \,/\, \mathcal{A}(q) \cap \mathcal{B}_i \neq \varnothing\}, \tag{2.1}$$

que anomenarem *C-obstacle*. La unió de tots els *C-obstacles*

$$\mathcal{C}_{obs} = \bigcup_{i=1}^{q} \mathcal{C}_{\mathcal{B}_i} \tag{2.2}$$

és anomenada la regió *C-obstacle* de $\mathcal{C}$, i el conjunt

$$\mathcal{C}_{lliure} = \mathcal{C} \backslash \mathcal{C}_{obs} \tag{2.3}$$

és anomenat l'*espai lliure* de $\mathcal{C}$. Qualsevol configuració de $\mathcal{C}_{lliure}$ és anomenada una *configuració lliure* de $\mathcal{C}$.

La figura 2.3 mostra un exemple d'obtenció de l'espai $\mathcal{C}_{\mathcal{B}_i}$ a partir de l'obstacle $\mathcal{B}_i$ en l'espai de treball $\mathcal{W}$. Es tracta d'un problema de dues dimensions on disposem d'un triangle $\mathcal{A}$, que és el sòlid rígid a desplaçar, i d'un rectangle $\mathcal{B}_i$, que és un obstacle fix a l'espai de treball $\mathcal{W}$. Per simplificar, considerem que el triangle només pot traslladar-se en x i y. D'aquesta manera l'espai de configuracions del triangle és $\mathcal{C} = \mathbb{R}^2$. La frontera de la regió $\mathcal{C}_{\mathcal{B}_i}$ és obtinguda resseguint el contorn de l'obstacle $\mathcal{B}_i$ mitjançant el triangle $\mathcal{A}$.

Es pot veure fàcilment que si permetem que el triangle $\mathcal{A}$, a més de traslladar-se en x i y, pugui rotar un angle θ, llavors el *C-obstacle* corresponent és delimitat per una superfície

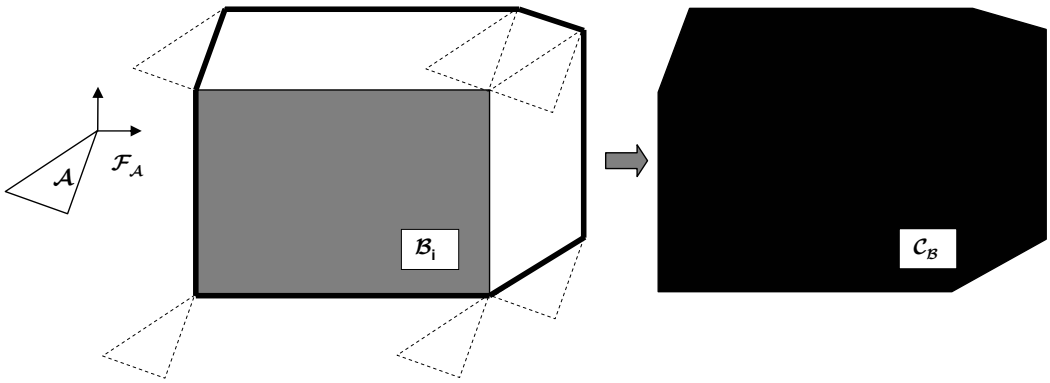


Figura 2.3: Generació de *C-obstacles* quan $\mathcal{A}$ és un polígon que es trasllada en el pla. Figura adaptada de [19].

reglada en l'espai $\{x, y, \theta\}$. De fet, cada tall a valor constant de θ d'aquesta superfície es pot generar amb el procediment anterior.

La representació explícita dels *C-obstacles* pot arribar a ser molt complexa en casos més generals. Pensem, per exemple, com de difícil pot resultar l'extensió del mètode anterior al cas que $\mathcal{A}$ i $\mathcal{B}_i$ siguin sòlids de forma arbitrària en $\mathcal{W} = \mathbb{R}^3$. Això fa que alguns dels mètodes de planificació de moviments evitin la representació explícita dels obstacles en $\mathcal{C}$. El mètode desenvolupat en aquest projecte n'és un exemple.

2.2.3 Noció de trajectòria

La noció de *continuïtat* és fonamental en la definició d'una trajectòria. La seva formalització requereix definir una topologia en $\mathcal{C}$. Una manera clàssica de fer-ho és especificar una funció de distància $d : \mathcal{C}\times\mathcal{C}\rightarrow\mathbb{R}$, i assumir que la topologia de $\mathcal{C}$ és la topologia mètrica induïda per d.

La distància d entre dues configuracions q i q', hauria de ser una funció tal que tendís a zero quan les regions $\mathcal{A}(q)$ i $\mathcal{A}(q')$ estiguessin molt pròximes entre sí. Una funció de distància que satisfà aquesta condició és:

$$d(q, q') = max\|a(q) - a(q')\| \quad \text{amb} \quad a \in \mathcal{A}, \tag{2.4}$$

on $\|x - x'\|$ és la distància Euclídia entre dos punts qualssevol x i x' en $\mathbb{R}^N$. En aquest projecte, la topologia en $\mathcal{C}$ serà la topologia induïda per aquesta distància.

Una *trajectòria* de $\mathcal{A}$ des de la configuració inicial q_{ini} a la configuració final q_{final} és una funció contínua:

$$\tau : [0, 1]\rightarrow\mathcal{C}, \tag{2.5}$$

amb

$$\tau(0) = q_{ini} \quad \text{i} \quad \tau(1) = q_{final}. \tag{2.6}$$

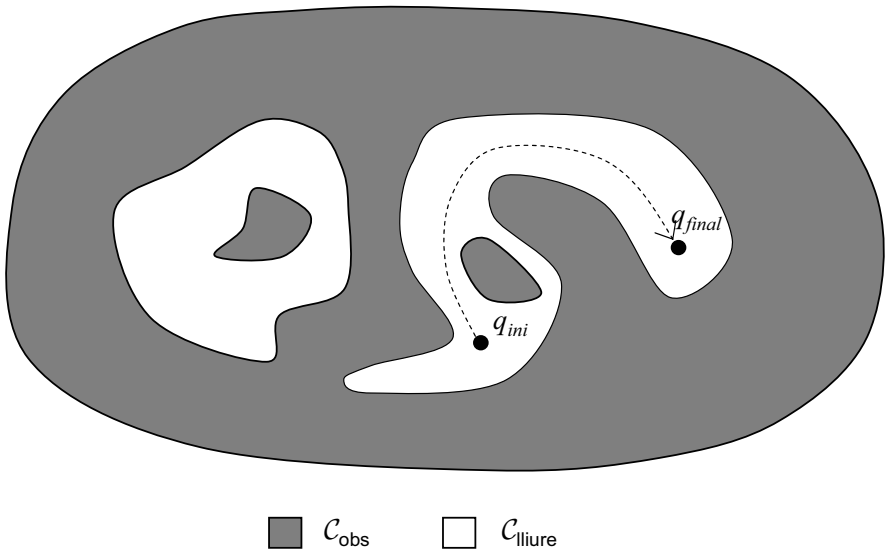


Figura 2.4: Problema bàsic en un espai de configuracions amb obstacles.

La continuïtat de τ implica que per tot $s_0 \in$ [0,1], $lim_{s \to s_0} max \|a(\tau(s)) - a(\tau(s_0))\| = 0$ per $a \in \mathcal{A}$, amb s prenent valors en [0,1]. Si $\mathcal{A}$ és un robot lliure-volant significa que, en absència d'obstacles, qualsevol trajectòria definida com s'ha dit anteriorment és factible.

Finalment, una *trajectòria lliure* entre dues configuracions q_{ini} i q_{final} és una funció contínua $\tau : [0,1] \to \mathcal{C}_{lliure}$, amb $\tau(0) = q_{ini}$ i $\tau(1) = q_{final}$. Aquesta definició, a més, indueix una partició natural de $\mathcal{C}_{lliure}$ en *components connexos*, on cada component és format per un conjunt de punts tals que qualssevol dos d'ells queden units per una trajectòria lliure.

Amb les anteriors definicions, i donada una configuració inicial i una de final, el problema bàsic de planificació de moviments queda traduït en generar una trajectòria lliure entre elles. Per a que hi hagi solució, ambdues configuracions hauran de pertànyer al mateix component connex de $\mathcal{C}_{lliure}$. Si no és així, aleshores no té solució. La figura 2.4 il·lustra tots aquests conceptes de forma esquemàtica.

2.2.4 Restriccions cinemàtiques

En el problema bàsic hem assumit que el sòlid rígid era un robot lliure-volant, i això ens portava a que les úniques restriccions de moviment que teníem eren a causa dels obstacles de l'espai de treball. En alguns problemes podria ser que interessés imposar restriccions cinemàtiques addicionals als moviments del robot. Aquestes restriccions poden ser bàsicament de dos tipus: *restriccions holonòmiques* i *restriccions no holonòmiques*. Les primeres sorgeixen, per exemple, en problemes de planificació com el tractat en aquest projecte, on es tenen cadenes cinemàtiques que cal mantenir tancades. Les segones apareixen en robots que tenen limitat l'espai de possibles moviments diferencials. Per constatar-ne la diferència, les repassarem tot seguit.

Restriccions holonòmiques. Assumint que una configuració és representada per una llista de paràmetres de mínima cardinalitat (secció 2.2.1), una restricció holonòmica és una equació que relaciona aquests paràmetres, i pot ser resolta per a un d'aquests. Això fa que es redueixi la dimensió de l'espai de configuracions en una dimensió. En general, podem dir que k restriccions holonòmiques independents redueixen la dimensió de l'espai en k.

Els mecanismes articulats, per exemple, donen lloc a restriccions d'aquest tipus. Imaginem un mecanisme articulat en el pla, constituït per articulacions de revolució. Cada articulació introdueix dues restriccions holonòmiques. Això és així perquè només permet un gir entre els sòlids que uneix, quan, en canvi, en absència d'articulacions, cada sòlid podria a més traslladar-se en el pla.

Les restriccions holonòmiques certament afecten l'estructura de l'espai de configuracions del robot, i poden canviar la seva connectivitat global. Tot i així, si $\mathcal{A}$ és un mecanisme articulat sense cadenes cinemàtiques tancades, les restriccions holonòmiques no introdueixen nous conceptes fonamentals. De fet, bastants dels algorismes disponibles per resoldre el problema bàsic de la planificació de moviments continuen sent aplicables [19]. Tanmateix, quan $\mathcal{A}$ conté cadenes cinemàtiques tancades, les restriccions holonòmiques que se'n deriven donen lloc a sistemes d'equacions polinòmiques on l'espai solució és rarament parametritzable (secció 5.1) i el problema és substancialment més complex. De fet, encara no s'han trobat algorismes de planificació generals que puguin resoldre aquest cas eficientment.

Malgrat tot, hom pot desenvolupar mètodes eficients per cassos específics com el tractat en aquest projecte (capítol 5).

Restriccions no holonòmiques. Una restricció no holonòmica és una equació no integrable que involucra els graus de llibertat del robot i les seves derivades. Aquestes restriccions no disminueixen la dimensió de l'espai de configuracions atansable pel robot, però sí que redueixen la dimensió de l'espai de moviments diferencials, és a dir, l'espai de velocitats possibles des de qualsevol configuració.
que equival a imposar que la velocitat d'R sigui tangencial a la corba que defineix la trajectòria.

Considerem per exemple un robot mòbil $\mathcal{A}$, modelitzat com un sòlid rígid rectangular i que es mou com un cotxe en un espai $\mathcal{W} = \mathbb{R}^2$ (figura 2.5). Com que $\mathcal{A}$ és un sòlid rígid, l'espai de configuracions associat és de dimensió tres, dues translacions i una rotació. Representem una configuració de $\mathcal{A}$ per (x, y, θ), on x i y són les coordenades en la referència $\mathcal{F}_\mathcal{W}$ del punt R equidistant a les dues rodes del darrera, i $\theta \in [0, 2\pi)$ és l'angle entre l'eix x de $\mathcal{F}_\mathcal{W}$ i l'eix principal de $\mathcal{F}_\mathcal{A}$. Durant un moviment, assumim que en cap moment podem derrapar, i això significa que la velocitat de R sempre mantindrà la direcció de l'eix x de $\mathcal{F}_\mathcal{A}$. Aquesta restricció no holonòmica és caracteritzada per l'equació

$$-\sin\theta\, dx + \cos\theta dy = 0 \tag{2.7}$$

Hom pot veure que aquesta equació no és integrable, i que per tant és una restricció no holonòmica. A causa d'aquesta restricció, l'espai de moviments diferencial $(dx, dy, d\theta)$ del robot a qualsevol configuració (x, y, θ) és un espai de dues dimensions. Si el cotxe hagués estat un robot lliure-volant, aquest espai hauria estat de dimensió tres. El moviment

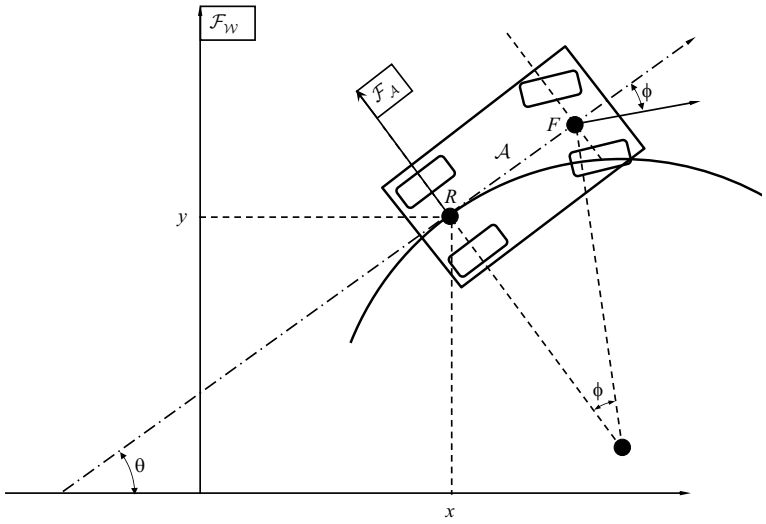


Figura 2.5: Model d'un cotxe girant. Figura adaptada de [19].

instantani del robot és determinat per dos paràmetres: la velocitat lineal al llarg del seu eix principal, i l'angle de direcció. De manera que quan el robot canvia l'angle de la direcció de les rodes, també canvia la direcció del vector velocitat, permetent que el robot pugui assolir qualsevol posició i orientació en el pla. En la figura 2.6 s'il·lustra una trajectòria generada per un robot mòbil d'aquest tipus, que realitza una operació d'aparcament. Una trajectòria diferent, segurament violaria alguna restricció no holonòmica. Podem concloure que les restriccions no holonòmiques restringeixen el vector de velocitats per cada configuració, però no canvien la dimensió de l'espai de configuracions. Això farà

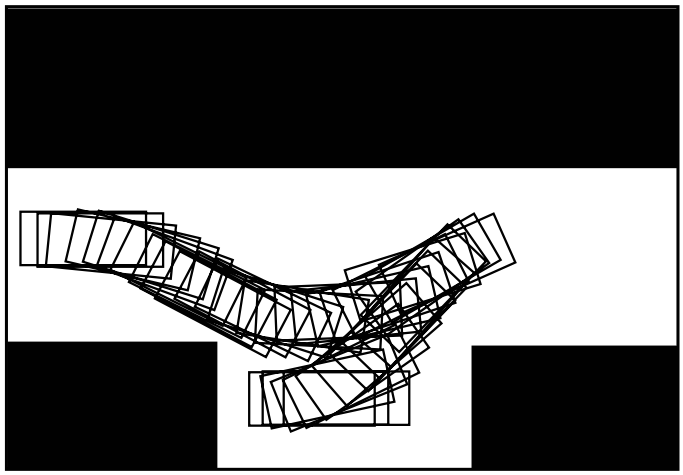


Figura 2.6: Exemple de planificació de moviments en un vehicle no holònom. Figura adaptada de [19].

que, de vegades, una trajectòria que teòricament seria factible entre dues configuracions, no es pugui realitzar perquè no compleix alguna de les restriccions no holonòmiques del problema tractat. En un planificador, les restriccions no holonòmiques són més difícils de tractar que les holonòmiques. En aquest projecte assumirem que el sistema robotitzat tractat no presenta restriccions d'aquest tipus.

Capítol 3

Antecedents

A la literatura trobem una gran varietat de mètodes per resoldre el problema de la planificació de moviments. Com veurem, mentre alguns fan èmfasi en l'obtenció d'algorismes complets (si hi ha solució aleshores la troben) i sovint són d'elevat cost computacional en espais d'alta dimensió, altres s'esforcen en l'obtenció ràpida de solucions tot renunciant a la completesa. El primer tipus d'algorismes té un elevat interès teòric, car la seva existència demostra que el problema és resoluble, per bé que amb un cost computacional molt elevat. El segon tipus d'algorismes, malgrat no ser capaços de trobar la solució en alguns casos, funcionen prou bé des d'un punt de vista pràctic en un gran ventall de problemes. Per tal d'escollir el que més s'adequa al desenvolupament d'aquest projecte, a continuació fem un repàs dels mètodes més estudiats. Hom els pot classificar en tres grans grups, d'acord amb el tipus d'estratègia utilitzada: mètodes basats en la descomposició cel·lular, en l'ús de camps potencials, o en la construcció de mapes de rutes.

3.1 Descomposició cel·lular

Aquesta és segurament una de les estratègies de planificació que més s'ha estudiat. Consisteix en descomposar l'espai lliure en regions, anomenades *cel·les*, de forma que una trajectòria entre dues configuracions d'una mateixa cel·la pugui ser generada fàcilment. A continuació, es construeix l'anomenat *graf de connectivitat*, on cada un dels seus nodes representa una cel·la, i on dos nodes són units per una aresta si i només si les dues corresponents cel·les són adjacents. Amb això, si sabem a quines cel·les pertanyen les configuracions inicial i final, només cal cercar aquest graf per mirar de trobar un *canal*: una seqüència de cel·les, adjacents dues a dues entre sí, que van des de la cel·la inicial a la final. A partir del canal hom pot llavors trobar una trajectòria lliure fàcilment. La figura 3.1 mostra un exemple de descomposició cel·lular d'un espai de configuracions de dues dimensions, i amb $\mathcal{C}$-obstacles poligonals. Els nodes ombrejats representen el canal existent entre dues configuracions donades q_{ini} i q_{final}.

Hom troba dues variants principals d'aquesta estratègia: el mètode *exacte*, i el mètode *aproximat*. El mètode exacte és complet, i per tant garanteix trobar una trajectòria lliure entre dues configuracions quan realment existeix. L'inconvenient d'aquest és que, excepte en casos simples, és computacionalment més costós que l'aproximat. Per aquest motiu, a la pràctica s'utilitza més el mètode aproximat que l'exacte, malgrat no ser complet.

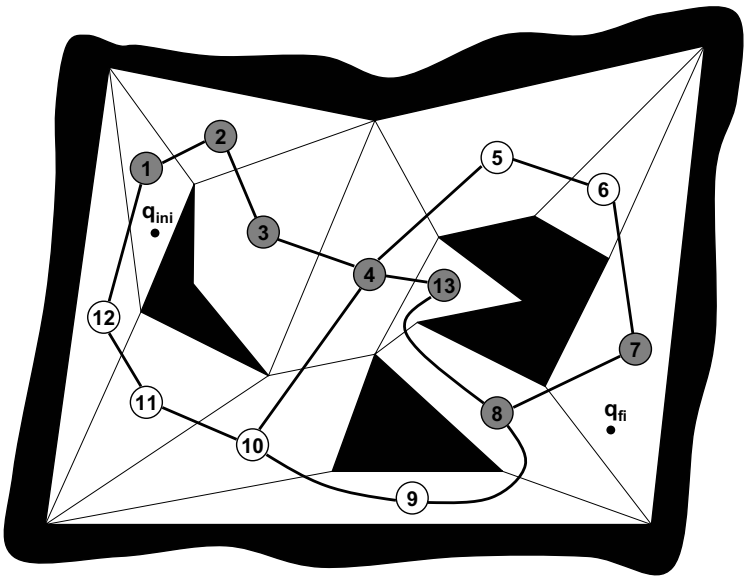


Figura 3.1: Descomposició cel·lular d'un espai bidimensional. Figura adaptada de [19].

3.1.1 Descomposició exacta

Aquest mètode descomposa l'espai lliure en un conjunt de cel·les no encavalcades entre sí, de manera que la seva unió recobreix exactament tot l'espai. La frontera de cada cel·la es tria de forma que correspongui a un canvi significatiu quan és travessada. En canvi, l'interior de la cel·la no és una regió crítica, i per tant es poden establir trajectòries lliures entre parelles de configuracions internes a ella fàcilment.

Hi ha una gran varietat d'algorismes que es basen en la descomposició cel·lular exacta. Per a espais de configuracions de dues dimensions i $\mathcal{C}$-obstacles poligonals tenim l'algorisme de *descomposició en trapezis*. La descomposició de l'espai lliure en trapezis, es pot realitzar mitjançant l'algorisme donat en [19, pàg. 134] que descomposa un polígon convex en temps polinòmic amb el seu nombre de vèrtexs. Malauradament, la presència de forats en el polígon (a causa dels $\mathcal{C}$-obstacles) fa que aquest problema sigui NP-dur [24]. Per paliar-ho, es pot fer una descomposició subòptima, consistent en subdividir l'espai emprant rectes verticals des de cadascun dels vèrtexs dels $\mathcal{C}$-obstacles. El resultat és una descomposició trapezoïdal de l'espai lliure, també anomenada *descomposició vertical* (figura 3.2), que de fet és també aplicable al cas en que $\mathcal{C} = \mathbb{R}^3$ i els $\mathcal{C}$-obstacles siguin polièdrics. El tractament complet amb aquesta última tècnica suposa un temps $O(n \log n)$, on n és el nombre total de vèrtexs dels $\mathcal{C}$-obstacles [31]. El mètode de descomposició vertical es pot estendre al cas en què l'espai de configuracions és $\mathcal{C} = \mathbb{R}^3$ amb els $\mathcal{C}$-obstacles polièdrics.

Construït el graf de connectivitat, la cerca d'un canal es pot fer bé aplicant-hi una cerca en profunditat estàndard, bé emprant l'algorisme A^* [15] [27]. En el primer cas, el temps de còmput requerit és $O(n)$, i en el segon és $O(n \log n)$.

Finalment, pel seu interès teòric, cal esmentar que es pot obtenir una descomposició cel·lular exacta d'un espai de configuracions arbitrari, amb els $\mathcal{C}$-obstacles definits com

conjunts semialgebraics. El mètode, batejat com la *descomposició axial cilíndrica*, fou desenvolupat per Schwartz i Sharir [32], i permet resoldre el problema bàsic de la planificació de moviments en el cas general, així com moltes de les seves extensions, incloent la planificació de sistemes multi-robot, o de braços articulats com els tractats en aquest projecte. El mètode, però, té un cost doblement exponencial amb la dimensió de $\mathcal{C}$, i rarament s'utilitza.

3.1.2 Descomposició aproximada

Aquest mètode subdivideix l'espai lliure en un conjunt de cel·les rectangulars, mitjançant un algorisme de descomposició. El quadre dret de la figura 3.3 mostra un exemple de descomposició aproximada a una certa iteració, per l'espai de configuracions de dues dimensions amb $\mathcal{C}$-obstacles poligonals del quadre esquerre. Cada cel·la es classifica aleshores com

- *plena*: si la cel·la és continguda en un $\mathcal{C}$-obstacle.
- *buida*: si no intersecta amb cap $\mathcal{C}$-obstacle.
- *mixta*: si intersecta parcialment amb algun $\mathcal{C}$-obstacle.

Les cel·les plenes s'eliminen i es construeix un graf de connectivitat amb les restants, representant les relacions d'adjacència entre cel·les de tipus buit o mixt. Com veiem, la unió d'aquestes cel·les no representa exactament l'espai lliure, sinó un subespai del mateix, i ara la frontera d'una cel·la no caracteritza una discontinuïtat de cap tipus, a diferència del mètode exacte.

Com en el cas general, donades q_{ini} i q_{final}, cal cercar un canal que les uneixi. Ara però el canal pot ser de dos tipus:

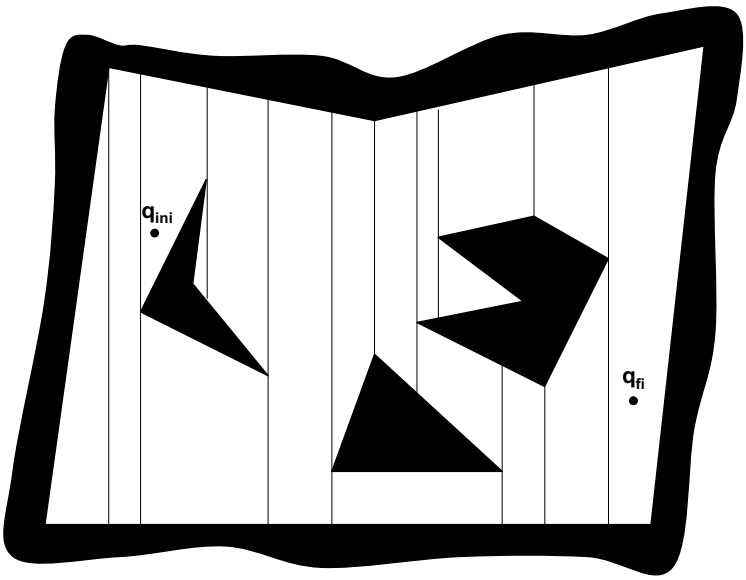


Figura 3.2: Il·lustració d'una descomposició vertical. Figura adaptada de [19].

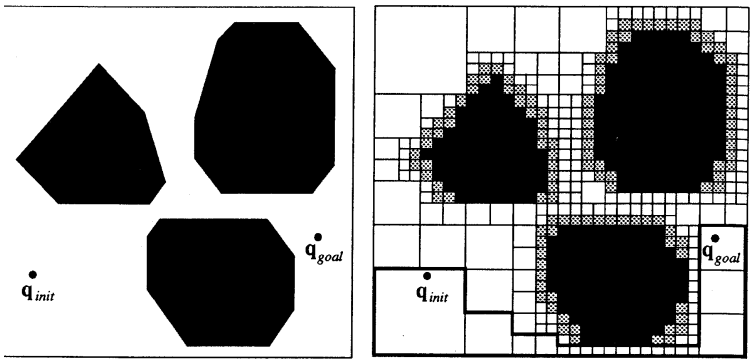


Figura 3.3: Exemple d'aplicació del mètode aproximat de descomposició en cel·les. Figura adaptada de [19].

- *B-canal*: si les seves cel·les són totes de tipus buit.
- *M-canal*: si alguna cel·la del canal és de tipus mixt.

Si trobem un B-canal, hi ha la garantia que existeix una trajectòria entre les dues configuracions. Ara bé, si només podem trobar un M-canal, no tenim la certesa de que tal trajectòria lliure existeixi i cal descomposar els rectangles de tipus mixt en d'altres més petits aplicant una nova iteració de l'algorisme de descomposició.

El procés anterior es repeteix fins que s'estableix un B-canal entre les configuracions q_{ini} i q_{final}, o fins que s'arriba a un llindar mínim quant a les dimensions dels rectangles generats. Un cop trobat un B-canal, podem establir una trajectòria lliure entre les configuracions q_{ini} i q_{final} de forma fàcil. Si $\mathcal{C} = \mathbb{R}^2$, per exemple, només cal unir els centres de cada cel·la amb un punt situat a la frontera entre dues cel·les adjacents, obtenint una trajectòria com la indicada a la figura 3.4.

La descomposició cel·lular aproximada és un mètode general, i en teoria es pot aplicar al problema bàsic de la planificació de moviments, així com a moltes de les seves extensions. Tanmateix, a la pràctica, el temps de còmput que requereix creix ràpidament a mida que la complexitat de l'espai de configuracions augmenta i més alta és la seva dimensió. Si es desitja, es pot acotar el temps de còmput limitant la mida de les cel·les, però això fa que l'algorisme perdi completesa. És en aquest sentit que es diu que el mètode és *complet a la resolució fixada*: el mètode garanteix que, si hi ha una solució descomposant l'espai lliure a la resolució fixada, aleshores la troba. Tot i així, aquest mètode és realment aplicable només quan la dimensió de $\mathcal{C}$ és prou petita.

3.2 Camp de potencials

En aquesta estratègia es tracta el robot com si fos una partícula sota la influència d'un camp de potencials artificial $\mathbf{U}$, on les seves variacions locals indiquen l'estructura de l'espai lliure.

La funció de potencial és freqüentment definida com la suma d'un potencial *d'atracció* $\mathbf{U}_{atrac}$, que arrossega el robot cap a la configuració final, i un de *repulsió* $\mathbf{U}_{rep}$, que empeny

el robot lluny dels obstacles:

$$\mathbf{U}(q) = \mathbf{U}_{atrac}(q) + \mathbf{U}_{rep}(q). \tag{3.1}$$

En la suma anterior $\mathbf{U}_{atrac}$ és independent dels $\mathcal{C}$-obstacles mentre que $\mathbf{U}_{rep}$ és independent de la configuració final. Normalment $\mathbf{U}_{atrac}$ es defineix com una funció parabòlica que pren valors positius arreus, excepte a la configuració final on el valor és nul. En canvi, la funció $\mathbf{U}_{rep}$ és definida com una funció positiva que tendeix a l'infinit per configuracions pròximes a la regió dels $\mathcal{C}$-obstacles, i pren valors nuls per configuracions que estan allunyades d'aquests per damunt d'una distància llindar.

Un cop determinat el camp de potencial resultant, calculem el camp de forces artificial induït fent:

$$\vec{F}(q) = -\vec{\nabla}\mathbf{U}(q). \tag{3.2}$$

Per a cada configuració, aquesta força indica la direcció i sentit de la màxima disminució del camp de potencials i, per tant, indica la millor direcció a seguir per tal que el robot vagi cap a la configuració final evitant col·lisionar amb obstacles intermedis. Per determinar la trajectòria lliure a partir d'aquest camp de forces, podem recórrer a diversos mètodes de cerca. Resumirem els dos que s'utilitzen més sovint: la cerca *guiada per gradient* i la cerca basada en *arbres d'exploració*.

En la cerca guiada per gradient [19, pàg 311], a cada iteració es busca una configuració q_{i+1} propera a la configuració actual q_i seguint la direcció de la força $\vec{F}(q_i)$. Matemàticament, el procés es pot expressar recursivament com

$$q_{i+1} = q_i + \delta_i \frac{\vec{F}(q_i)}{\| \vec{F}(q_i) \|}, \tag{3.3}$$

on δ_i és un real positiu prou petit de forma que a la configuració q_{i+1} la col·lisió sigui poc probable. Aplicant l'equació 3.3 iterativament, podem generar una trajectòria lliure on els

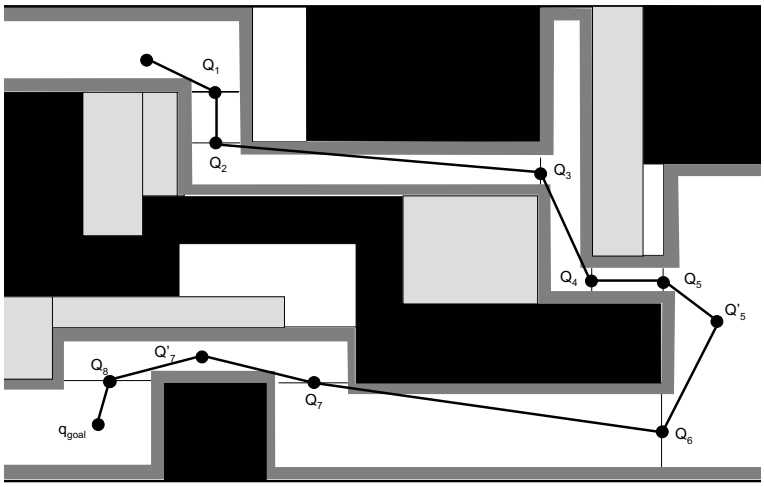


Figura 3.4: Exemple d'una trajectòria lliure en un B-canal. Figura adaptada de [19].

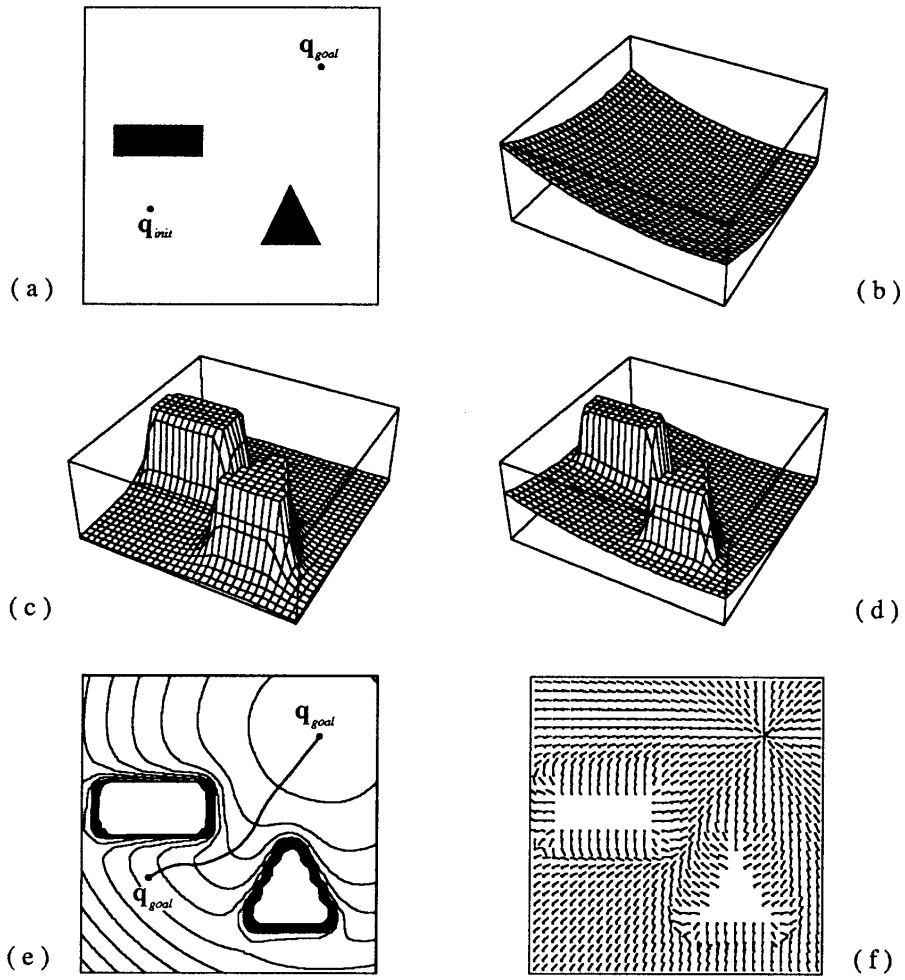


Figura 3.5: Exemple d'aplicació del mètode de camp de potencials en el cas translacional. Figura adaptada de [19].

valors de la funció potencial van descendint fins que convergim a la configuració final. La figura 3.5 il·lustra aquest procés en un espai de configuracions de dues dimensions i amb $\mathcal{C}$-obstacles poligonals. En el quadre (b) es representa una funció de potencial atractiva i en el (c) una de repulsiva. En el quadre (d) es representa la funció de potencial resultant, que és la suma de les anteriors. Al quadre (e) es representen les seves corbes de nivell, i una trajectòria lliure que uneix la configuració inicial amb la final tot emprant el camp de forces artificials del quadre (f).

Com es pot intuir, aplicant una estratègia de cerca tan simple com l'anterior, el planificador pot quedar atrapat en un mínim local de la funció $\mathbf{U}(q)$ que sigui diferent al de la configuració final. Evitar aquests mínims locals no és senzill. Primer, pel fet que hem de reconèixer quan ens trobem en un d'ells: com que els desplaçaments del robot són discrets, el planificador no s'atura exactament quan la força val zero i es manté oscil·lant en l'entorn del mínim local. En segon lloc, el planificador ha de ser capaç d'escapar d'aquests mínims. Una manera d'aconseguir-ho és desplaçar el robot en una certa direcció triada

aleatòriament, allunyant-lo del mínim local, i després reprendre l'algorisme novament, però això no garanteix que el robot arribi finalment al mínim absolut.

El problema dels mínims locals es pot evitar en gran mesura utilitzant una estratègia de cerca basada en arbres d'exploració [2]. Aquest mètode construeix primer una xarxa de configuracions lliures $G_{\mathcal{C}}$, que s'obté discretitzant cadascun dels m eixos de l'espai de configuracions. A continuació, de forma iterativa es construeix un arbre T, els nodes del qual són configuracions en $G_{\mathcal{C}}$ i on l'arrel és la configuració inicial q_{ini}. A cada iteració, l'algorisme considera la fulla de T amb menor potencial, examina quin dels veïns d'aquesta té el valor de potencial més petit, i reté els altres que tenen un valor de potencial menor que una certa cota. El veí amb menor potencial i els veïns retinguts es pengen llavors com a successors de la fulla considerada. L'algorisme finalitza quan s'assoleix la configuració final o quan el subconjunt $G_{\mathcal{C}}$ accessible des de q_{ini} s'ha explorat sense èxit. Aquest algorisme garanteix retornar una trajectòria lliure, si n'existeix una en la xarxa $G_{\mathcal{C}}$, en temps $O(mr^m \log r)$, essent r el nombre de subdivisions al llarg de cada eix.

3.3 Mapes de rutes

En aquesta categoria s'hi inclouen tots els mètodes que capturen la connectivitat de l'espai lliure en una xarxa de corbes unidimensionals anomenada *mapa de rutes*. Un cop construït aquest mapa, el problema es redueix a connectar q_{ini} i q_{final} a dues configuracions del mapa q'_{ini} i q'_{final} , i veure si existeix un camí que les uneix[1].

Repassarem tot seguit els tipus principals de mapes que hom troba a la literatura: *els grafs de visibilitat, els mapes obtinguts per retracció, els mapes de siluetes i els mapes probabilístics*. Convé notar aquí que les descomposicions cel·lulars no es consideren habitualment com a mètodes basats en mapes. Si bé és veritat que d'una descomposició cel·lular es pot obtenir fàcilment un mapa de rutes, la informació continguda en una descomposició és molt més rica que la que ofereix un mapa. (Cada canal cel·lular, per exemple, representa una infinitud de possibles trajectòries que podrien ser utilitzades per evitar obstacles inesperats sense necessitat de fer una nova cerca.)

3.3.1 Grafs de visibilitat

Els grafs de visibilitat s'apliquen essencialment a espais de configuracions de dues dimensions amb $\mathcal{C}$-obstacles poligonals. El mapa de rutes és un graf on els seus nodes emmagatzemen les configuracions que es troben als vèrtexs dels $\mathcal{C}$-obstacles, i dos nodes estan connectats per una aresta si el segment recte que uneix les seves configuracions no intersecta amb cap $\mathcal{C}$-obstacle (figura 3.6). Per generar una trajectòria entre dues configuracions, caldrà primer connectar-les a aquest graf. Si la connexió té èxit es busca el camí mínim que les uneixi. En cas que s'hagi trobat un camí, aquest donarà lloc a una

[1]És important aclarir els significat que donarem als termes camí i trajectòria, ja que apareixeran al llarg de l'exposició, i poden portar al lector a confusió. En terminologia de grafs, el terme camí és reservat per indicar la seqüència de vèrtexs travessats en anar d'un node a un altre del graf. En canvi, el terme trajectòria es refereix a la seqüència de configuracions que el robot adopta en passar d'una configuració inicial a una de final (secció 2.2.3). Tot i així, un camí portarà implícitament una trajectòria associada, com veurem més endavant.

trajectòria poligonal que connecta q_{ini} i q_{final} a través dels vèrtexs dels $\mathcal{C}$-obstacles. Això fa que el robot estigui en contacte amb els obstacles en alguns punts de la seva trajectòria.

L'algorisme més simple per a la construcció de grafs de visibilitat fou donat per Wesley i Lozano-Pérez [26] i corre en temps $O(n^3)$ on n és el nombre de vèrtexs dels $\mathcal{C}$-obstacles. Versions millorades d'aquest algorisme foren posteriorment donades per Welzl [37] i Edelsbrunner [9], que aconseguien construir el graf en temps $O(n^2)$. Determinar el camí mínim del graf que uneix els nodes inicial i final no és una tasca senzilla, perquè poden existir múltiples camins factibles entre aquests dos nodes. L'algorisme A^* [15, 27] és el que dóna millors resultats per aquest propòsit, amb un temps de còmput de $O(n^2)$. En resum, el temps total requerit, incloent el procés de construcció del graf i la cerca del camí mínim, és $O(n^2)$, si s'assumeix que la frontera de C_{lliure} ja ve donada.

Fins ara hem tractat un problema específic en l'espai de configuracions de dues dimensions. Hi ha vàries extensions del problema, que es poden classificar en dos grups: l'extensió a espais de configuracions de dues dimensions i amb polígons generalitzats, i l'extensió a espais de configuracions de més dimensions. En el primer cas, el mètode és semblant a l'anterior amb la diferència que els $\mathcal{C}$-obstacles són *polígons generalitzats*: regions delimitades per línies rectes i arcs circulars. Amb aquest plantejament, Laumond demostra que el problema és resoluble en temps polinòmic [20]. Quan al segon grup, el mètode té més limitacions del que es podria imaginar car: (1) el graf pot no contenir el camí mínim entre dos nodes (figura 3.7), (2) Canny va demostrar que el problema ja és NP-dur per a espais de configuracions de 3 dimensions [4], i (3) el mètode ni tant sols generalitza el cas d'un robot traslladant-se i rotant en el pla, ja que en aquest cas els $\mathcal{C}$-obstacles són superfícies reglades i no pas políedres de cares planes.

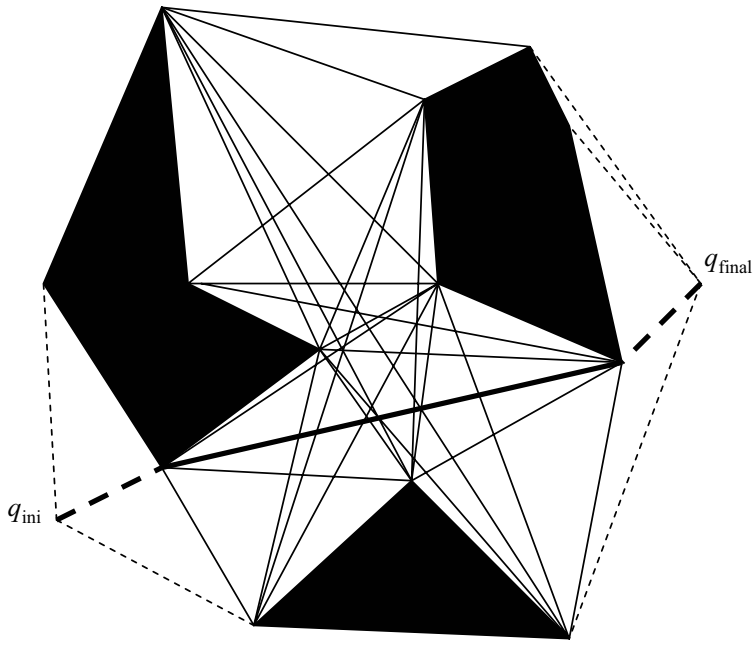


Figura 3.6: Graf de visibilitat en un espai de configuracions de dues dimensions. Figura adaptada de [19].

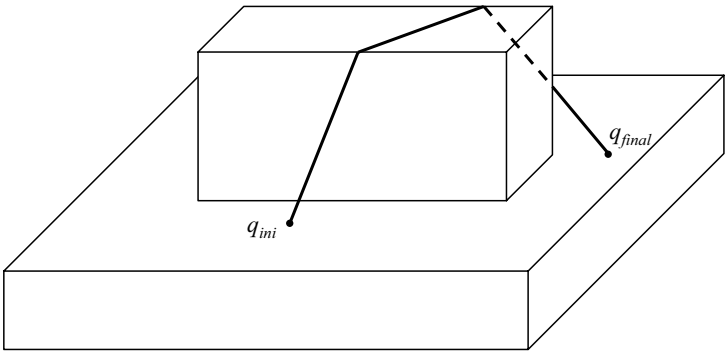


Figura 3.7: Quan $\mathcal{C} = \mathbb{R}^3$ i els $\mathcal{C}$-obstacles són polièdrics, els mètodes de visibilitat no poden trobar aquest camí mínim. Figura adaptada de [19].

3.3.2 Mètodes basats en retracció

Aquests mètodes consisteixen en definir una funció de retracció ρ que transforma $\mathcal{C}_{lliure}$ en un subconjunt de corbes unidimensionals d'ell mateix, tal que (1) ρ és contínua i injectiva, (2) la restricció de $\mathcal{C}$ a aquest subconjunt és l'aplicació identitat, i (3) el subconjunt preserva la connectivitat de $\mathcal{C}_{lliure}$. Per exemple, en un espai de configuracions $\mathcal{C} = \mathbb{R}^2$ on $\mathcal{C}_{lliure}$ sigui l'interior d'una regió poligonal tancada, hom el pot retraure cap al seu *diagrama de Voronoi*: el lloc geomètric dels cercles inscrits en $\mathcal{C}_{lliure}$ que fan contacte amb més d'un punt de la seva frontera. Al ser l'interior de $\mathcal{C}_{lliure}$ una regió poligonal tancada, el diagrama és format per una col·lecció finita de rectes, i de corbes parabòliques (figura 3.8). Kirkpatrick demostra que si $\mathcal{C}_{lliure} = \mathbb{R}^2$, aquest diagrama es pot construir en temps $O(n \log n)$, on n és el nombre de vèrtexs dels $\mathcal{C}$-obstacles.

Ara, donades q_{ini} i q_{final}, es pot trobar una trajectòria lliure emprant el diagrama de Voronoi. Si apliquem la funció de retracció ρ a aquests punts tenim que $\rho(q_{ini})$ i $\rho(q_{final})$ són dos punts del diagrama de Voronoi, i podem cercar a través d'ell una trajectòria que els uneixi. La trajectòria lliure resultant, per tant, serà la concatenació de tres trajectòries: una entre q_{ini} i $\rho(q_{ini})$, una altra entre $\rho(q_{ini})$ i $\rho(q_{final})$ seguint el diagrama, i finalment una tercera entre $\rho(q_{final})$ i q_{final}. La primera i la tercera d'aquestes trajectòries sempre existiran, ja que com hem dit anteriorment ρ és injectiva.

Finalment cal esmentar que les tècniques de retracció són essencialment només aplicables a espais de configuracions de dues o tres dimensions ja que, en general, és molt costós calcular el diagrama de Voronoi induït per un conjunt de $\mathcal{C}$-obstacles.

3.3.3 Mapes de siluetes

La construcció de mapes de siluetes fou estudiada per Canny el 1988 [4]. Es tracta d'un mètode general, ja que permet planificar el moviment d'un punt en l'espai, quan aquest és un conjunt compacte semialgebraic arbitrari espai. Molt breument, donat un conjunt semialgebraic compacte de dimensió m, el mètode obté el seu mapa de rutes representatiu tot calculant la silueta del conjunt des d'un punt situat a l'infinit en algun dels eixos de referència. A continuació, el mètode connecta les corbes resultants tot

aplicant recursivament el mateix procediment a seccions de dimensió $m-1$, $m-2$, ... fins a obtenir un mapa totalment connex sobre el qual es fa la cerca d'un camí que uneixi la configuració inicial i la final.

El mètode és complex i utilitza eines avançades de Geometria Diferencial (les "estratificacions") i de Teoria de l'Eliminació ("els polinomis resultants"). Una explicació detallada del mètode queda fora de l'objectiu d'aquest treball, però es fa esment aquí de la seva existència perquè:

- És un mètode general: permet resoldre el problema bàsic de la planificació de moviments i moltes de les seves extensions, incloent el problema d'aquest projecte.
- És complet: si existeix solució la troba, i en cas contrari ho detecta.
- És l'únic mètode general i complet que té un cost simplement exponencial en la dimensió a $\mathcal{C}$. El mètode de la descomposició axial cilíndrica és l'únic mètode general i complet alternatiu, però té un cost doblement exponencial en aquesta dimensió.

Malgrat que els mapes de siluetes tenen un elevat interès teòric, a la pràctica rarament s'utilitzen perquè la seva construcció és complexa i té un cost computacional molt elevat.

3.3.4 Mapes probabilístics

Un seriós inconvenient que tenen molts dels mètodes anteriors és que assumeixen que es disposa d'una representació explícita dels $\mathcal{C}$-obstacles. Tanmateix, aquesta representació

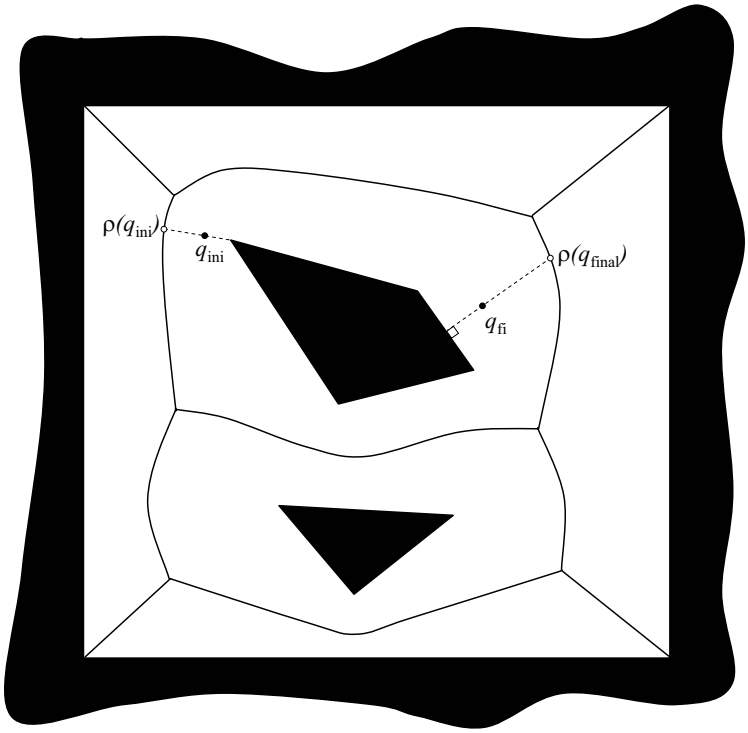


Figura 3.8: Diagrama de Voronoi quan $\mathcal{C}_{lliure}$ és poligonal. Figura adaptada de [19].

és sovint difícil de construir, especialment si l'espai de configuracions té moltes dimensions. Els mapes probabilístics, en canvi, eviten aquesta construcció tot fent un ús intensiu d'algun detector de col·lisions prou eficient, i això fa que siguin especialment fàcils d'implementar i raonablement ràpids en temps de resposta [17].

La construcció del mapa es fa tot mostrejant aleatòriament l'espai de configuracions. Breument, per cada configuració generada, (1) es verifica si cau en l'espai lliure comprovant si el robot col·lisiona amb algun $\mathcal{C}$-obstacle, i (2) s'intenta connectar la configuració amb d'altres generades anteriorment tot emprant un *planificador local*. Es procura que aquest planificador sigui senzill i no gaire costós, i per això només es demana que sigui capaç de connectar dues configuracions quan aquestes són properes entre sí.

Iterant l'anterior procés, s'obté un graf on els nodes emmagatzemen configuracions lliures del robot, i dos nodes queden connectats per una aresta si el planificador local ha pogut trobar una trajectòria connectant les seves configuracions. Finalment, com en la resta de mètodes basats en mapes, la cerca d'un camí entre q_{ini} i q_{final} es fa en dues etapes. Primer es connecten q_{ini} i q_{final} a dues configuracions properes del mapa, i després es comprova si el mapa conté algun camí que uneixi.

Aquesta estratègia fa un èmfasi especial en l'eficiència: es desitja que el planificador funcioni "prou bé" en la majoria de casos pràctics, i que pugui respondre en un temps prou curt. L'eficiència, però, es guanya a costa de perdre completesa, car és impossible garantir que el mapa obtingut representi acuradament tots els components connexos de l'espai de configuracions. Tot i així, s'ha demostrat que el mètode és probabilísticament complet [16]. És a dir, la probabilitat de no trobar un camí entre q_{ini} i q_{final}, quan realment existeix, tendeix a zero a mida que el nombre de mostres obtingudes de $\mathcal{C}_{lliure}$ tendeix a infinit.

3.4 Discussió

Per tal d'escollir l'estratègia de planificació més adient cal tenir en compte les dues característiques principals del problema a resoldre en aquest projecte. En primer lloc, l'espai de configuracions de dos robots Stäubli RX60 treballant coordinadament és de dimensió sis, perquè una configuració específica queda determinada pels sis angles d'un dels dos robots i la configuració en la que treballa l'altre (secció 5.2). En segon lloc, l'espai lliure té una estructura de conjunt semialgebraic, perquè correspon al conjunt de configuracions lliures que són a la vegada solucions d'un sistema d'equacions polinòmiques, com veurem amb detall a la secció 5.1.

Davant dels requeriments anteriors, hom s'adona ràpidament que cal renunciar a disposar d'un mètode complet pel problema plantejat ja que, com hem vist, els dos únics mètodes capaços de tractar amb conjunts semialgebraics (el basat en mapes de siluetes i la descomposició axial cilíndrica) són extremadament ineficients en espais d'elevada dimensió. D'entre la resta de mètodes, només tres s'han pogut adaptar al cas de robots articulats amb molts graus de llibertat: la descomposició cel·lular aproximada [11, 25], el mètode dels camps potencials [2], i els mapes probabilístics [17]. Malgrat tot, d'entre aquests només la darrera estratègia s'ha generalitzat al cas en que els robots considerats formen alguna cadena cinemàtica tancada [38, 14]. Aquest darrer fet, unit a la facilitat

amb què hom pot implementar un planificador basat en mapes probabilístics, ha fet que aquesta hagi estat l'estratègia seleccionada per al desenvolupament d'aquest projecte.

Tanmateix convé dir que, malgrat no es té constància que s'hagi intentat prèviament, sí sembla factible una particularització de la descomposició cel·lular aproximada o del mètode dels camps potencials al cas de dos robots cooperants estàndard com els aquí considerats. Hom podria, per exemple, partir del treball de Lozano-Pérez [25] on es dóna un procediment per efectuar una descomposició cel·lular aproximada de l'espai lliure d'un manipulador sèrie arbitrari, tot subdividint els seus graus de llibertat en petits intervals i analitzant per a quins d'aquests el robot entra en col·lisió amb els obstacles o amb sí mateix. El mateix esquema sembla directament aplicable al cas d'aquest projecte car, encara que un segon robot hauria de mantenir la clausura cinemàtica amb el robot analitzat, això es podria assegurar en tot moment aprofitant que es disposa de solució tancada per la seva cinemàtica inversa. Malgrat tot, el mateix treball reconeix que aquesta descomposició cel·lular és costosa en espais de moltes dimensions, i que dóna lloc a una discretització excessivament fina de l'espai lliure que obliga a un segon pas de compactació per tal de fer-la utilitzable en temps real [19, pàg. 396], complicant excessivament el procés.

També podríem optar per particularitzar el mètode dels camps potencials al nostre cas, tot partint del treball de Barraquand, Langlois i Latombe [2], que el generalitza al cas de robots articulats. Novament, es podria aprofitar el fet que es disposa de solució explícita per la cinemàtica inversa dels robots per mantenir la clausura cinemàtica requerida. Tanmateix, per assegurar que el mètode no caigui mai en mínims locals, caldria o bé generar una funció de potencial amb un únic mínim, o bé implementar una tècnica per escapar dels mínims locals. Els treballs de Rimon i Koditschek [18, 29, 30] demostren que la primera opció és teòricament factible, però el procés de construcció resulta molt complex a la pràctica i rarament s'aplica. La segona opció, per bé que possible si emprem una estratègia com l'explicada a la secció 2.2, no aporta completesa al mètode, i per tant aparentment cap avantatge respecte els mètodes basats en mapes probabilístics.

Capítol 4

Planificació basada en mapes probabilístics

Aquest capítol descriu en detall l'estratègia de planificació basada en mapes probabilístics. El mètode és aplicable a qualsevol robot lliure-volant i també a braços articulats amb molts graus de llibertat, sempre que no formin cap cadena cinemàtica tancada. Més endavant, al capítol 5 , veurem que el mètode és generalitzable al cas en què hi hagi tals cadenes, i com s'ha adaptat al problema d'aquest projecte.

La planificació basada en mapes probabilístics consta habitualment de dues fases: *la fase d'aprenentatge* i *la fase d'interrogació*. Durant la fase d'aprenentatge es construeix un mapa de rutes, emmagatzemat com un graf $R = (N, E)$. Els nodes en N són configuracions lliures del robot, i les arestes en E corresponen a trajectòries lliures entre aquestes configuracions, obtingudes mitjançant un planificador local simple i ràpid. Durant la fase d'interrogació, les configuracions inicial i final es connecten a dos nodes del mapa, i tot seguit s'explora el mapa per tal de trobar un camí que els uneixi (figura 4.1).

Com que treballarem en un entorn totalment estructurat (es coneixen perfectament els models geomètrics de tots els objectes) i estàtic (assumim que cap obstacle canvia de posició) l'estratègia anterior permet obtenir respostes de forma molt ràpida, car la fase d'aprenentatge es pot fer fora de línia, i el mapa resultant no haurà de ser recalculat a menys que canvi l'entorn o l'objecte a transportar.

4.1 La fase d'aprenentatge

La fase d'aprenentatge es subdivideix normalment en una etapa de *construcció* i una etapa d'*expansió*. L'etapa de construcció té dos objectius. Per una banda, obtenir un graf raonablement connex amb nodes que cobreixin uniformement l'espai lliure, i per altra banda que les regions de més "dificultat" continguin almenys alguns nodes. L'etapa d'expansió, en canvi, té com objectiu millorar la connectivitat del graf R. Per aconseguir-ho, es seleccionen els nodes de R que estan en zones d'alta dificultat de $\mathcal{C}$ i es generen nodes addicionals en les seves proximitats. Notem que la cobertura de l'espai lliure no és uniforme al mapa final, ja que depèn de quants nodes s'han expandit durant aquesta etapa.

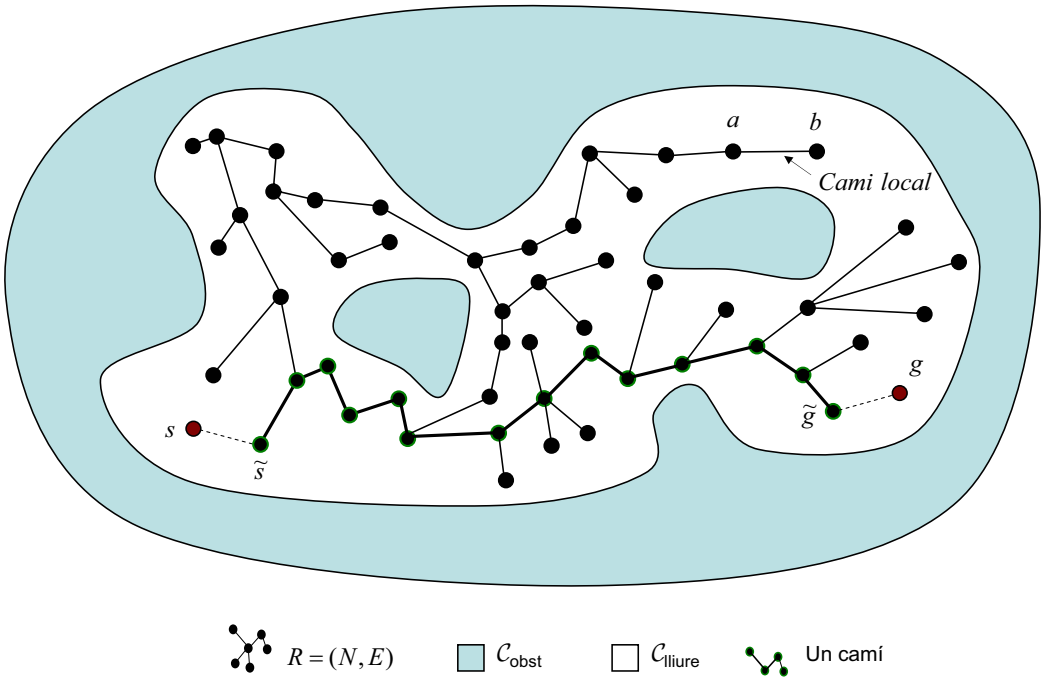


Figura 4.1: Planificació basada en mapes probabilístics. Els nodes de R corresponen a parelles de nodes per les que el planificador local ha trobat una trajectòria de connexió. Durant la fase d'interrogació, primer es connecten q_{ini} i q_{final} a configuracions properes en R, q'_{ini} i q'_{final}, i després es busca un camí en R que uneixi aquestes dues (indicat en negreta a la figura).

4.1.1 L'etapa de construcció

Inicialment el graf $R = (N, E)$ és buit. Aleshores, repetidament, generem configuracions lliures de forma aleatòria i les afegim a N. Per a cada nou node c generat, seleccionem un nombre determinat de nodes de N i intentem connectar c a cadascun d'ells emprant el planificador local. Sigui n un dels nodes seleccionats. Cada vegada que la crida al planificador local entre c i n té èxit, l'aresta (c, n) és afegida a E.

La selecció dels nodes que intentem connectar amb c es fa com segueix. Primer, es tria un conjunt N_c de nodes veïns de c. Aquest conjunt el formen aquells nodes que estan a una certa distància de c, per alguna mètrica D. Aleshores, anem agafant els nodes de N_c de forma que la seva distància a c sigui creixent. Cal remarcar, però, que només intentem connectar el node n amb el c, si n no pertany al mateix component connex que c. D'aquesta forma el graf que s'obté és una col·lecció d'arbres, car no conté cicles. Es procedeix així, i no d'altra forma, per tal d'estalviar crides al planificador local (que, com veurem, és la part més costosa de tot el procés). A més, si n i c ja pertanyen a un mateix component connex, vol dir que ja es coneix un camí per passar de l'un a l'altre, i intentar trobar una connexió directa no aporta cap informació addicional essencial.

Per fer la nostra presentació més formal, sigui Δ una funció simètrica $\mathcal{C}_{lliure} \times \mathcal{C}_{lliure} \rightarrow \{0, 1\}$, que retorna 1 si i només si la crida al planificador local entre dues

configuracions té èxit, i D una funció $\mathcal{C}_{lliure} \times \mathcal{C}_{lliure} \rightarrow R^+ \cup \{0\}$, anomenada *funció de distància*, que defineix una pseudo-mètrica a $\mathcal{C}$. Aquesta funció de distància D, a més, haurà de complir que sigui una funció simètrica i no degenerada. Amb aquestes definicions, l'algorisme de l'etapa de construcció pot ser expressat així:

ETAPA DE CONSTRUCCIÓ()

```
1  N ← ∅
2  E ← ∅
3  mentre ¬ condició_de_fi
4     fer
5        c ← una configuració aleatòria, lliure de col.lisions
6        N_c ← un conjunt de nodes veïns de c escollits de N
7        N ← N ∪ {c}
8        per tot n ∈ N_c, en odre incremental de D(c, n)
9           fer
10             si ¬ son_del_mateix_component_connex(c, n) ∧ △(c, n)
11                llavors
12                      E ← E ∪ {(c, n)}
13                      actualizar components connexos de R
```

Moltes instruccions de l'algorisme anterior estan encara sense especificar. En particular, cal definir com generarem les configuracions aleatòries a (5), proposar un planificador local per (10), aclarir la noció d'un node veí en (6), i triar una funció de distància D en (8). Ho fem tot seguit.

Generació de configuracions aleatòries. Els nodes de R haurien de constituir un mostreig suficientment uniforme de l'espai lliure. Suposant que una configuració del robot es pot descriure per un conjunt de paràmetres, i que cada un d'ells té un interval de valors de treball, cada nova configuració s'obté prenent valors aleatoris sobre cada un d'aquests intervals amb probabilitat uniforme. Un cop generada la configuració, es mira si es troba lliure de col·lisions. Si és així, aleshores és afegida a N; en cas contrari, és descartada. El detector de col·lisions haurà d'inspeccionar si qualsevol part del robot col·lisiona amb un obstacle, o bé si dos cossos diferents del robot ho fan entre sí. Tret d'aquest darrer requisit, la implementació del mètode és independent del detector de col·lisions utilitzat, tot i que l'eficiència global del mètode dependrà en gran mesura de l'eficiència d'aquest mòdul.

El Planificador local. El planificador local és una funció Booleana que, aplicada entre dues configuracions, determina si existeix o no una trajectòria lliure entre elles. Pot ser determinista, o no. És determinista quan aplicat entre dues configuracions prefixades sempre dóna idèntiques solucions. En canvi quan és no determinista les solucions poden canviar. Una altra característica del planificador local és la seva eficiència, o nombre d'èxits que dóna respecte als que donaria si fos complet. Normalment, els planificadors locals molt eficients tenen un cost computacional més alt que els planificadors locals poc eficients, ja que fan més inspeccions per determinar si existeix una trajectòria lliure entre dues configuracions.

Cal tenir en compte també com afecten les propietats del planificador local en les fases d'aprenentatge i d'interrogació. Si uséssim un planificador no determinista, les trajectòries locals haurien de ser emmagatzemades en el mapa de rutes. Això faria que el mapa ocupés molt més espai, i per aquest motiu s'utilitzarà un planificador local determinista. Pel que fa a la rapidesa del planificador local durant la construcció del mapa, hi ha un compromís entre el temps utilitzat en cada crida individual d'aquest i el nombre de crides totals. Si empréssim un planificador local lent, sovint seria capaç de en trobar un trajectòria lliure quan realment existís, i per tant serien necessaris pocs nodes per capturar la connectivitat de l'espai lliure. Tanmateix, aquest planificador seria força lent en cada crida, tot i que quedaria compensat per la disminució de crides necessàries en la construcció del mapa. Per altra banda, si empréssim un planificador local ràpid, requeriríem moltes més configuracions al mapa per aconseguir cobrir acceptablement l'espai lliure, i per tant el planificador seria cridat molt més sovint, encara que cada crida tardaria menys temps.

On realment la tria d'un planificador lent o ràpid té conseqüències és en la rapidesa de la fase d'interrogació. L'objectiu de construir un mapa de rutes és poder generar trajectòries ràpidament (gairebé instantàniament) davant d'una interrogació. Per tant, és important poder connectar q_{ini} i q_{final} al mapa de rutes, o detectar que tal connexió no és possible, molt ràpidament. Això requereix que el mapa sigui molt dens, de manera que sigui fàcil connectar-hi les configuracions d'inici i finalització durant la fase d'interrogació. Sembla doncs preferible usar un planificador local ràpid, inclòs si no és massa eficient. Cal afegir que si el planificador és determinista no cal memoritzar les trajectòries lliures entre configuracions, ja que recalcular-les a la fase d'interrogació suposa molt poc temps.

Un planificador local determinista bastant general, que és aplicable a tots els robots holònoms, o braços articulats sense cadenes cinemàtiques tancades, connecta simplement qualsevol parella de configuracions donades per un segment recte en l'espai de configuracions. Per comprovar que aquesta connexió és possible es seleccionen m configuracions $q_1, ..., q_m$ (figura 4.2) de manera que per cada parella de configuracions consecutives q_i, q_{i+1}, cap punt del robot, quan es troba posicionat a la configuració q_i, té una distància superior a ϵ respecte del mateix punt a la configuració q_{i+1} (ϵ és una constant positiva predeterminada). Per tal de verificar que el camí és lliure, s'utilitza un model geomètric "inflat" del robot, el resultat de convolucionar el model real amb una esfera de radi ϵ. Amb això, es pot considerar que el camí cau totalment a l'espai lliure si cadascuna de les configuracions $q_1, ..., q_m$ es troba dins els rangs establerts pels graus de llibertat i no dóna lloc a col·lisions.

Els nodes veïns. Una tria important en l'algorisme és determinar el conjunt N_c de nodes veïns del node c insertat. El planificador local intenta connectar cada un d'aquests nodes veïns amb el node c insertat, i això farà que gran part del temps de la fase d'aprenentatge sigui dedicada a fer aquestes connexions.

Per decidir quins nodes considerem dins N_c ens basarem en la següent hipòtesi. Existirà una distància d_{max} entre dues configuracions, per damunt de la qual serà poc probable que una crida al planificador local tingui èxit. Així doncs, per evitar crides innecessàries del planificador, formaran part de N_c aquells nodes que estiguin a una distància inferior

a d_{max} respecte de la configuració c. És a dir,

$$N_c = \{\tilde{c}\,\epsilon\, N | D(c,\tilde{c}) < d_{max}\}. \tag{4.1}$$

A més, per tal d'intentar aquelles connexions que a priori tenen major probabilitat d'èxit, primer intentarem connectar c als nodes que estiguin a menys distància de c. Per tant serà bo que abans d'intentar fer les connexions disposem dels nodes veïns N_c ordenats a distància creixent respecte de c.

Finalment, cal esmentar que per tal d'aconseguir que el procés de connexió de c als seus veïns més propers tingui un cost acotat durant la fase de construcció del mapa, s'aconsella que la quantitat de nodes considerats en N_c no superi un valor màxim, que [17] situa al voltant de trenta. D'aquesta forma s'aconsegueix que el nombre de crides del planificador local sigui aproximadament lineal amb el nombre de nodes del graf construït.

La figura 4.3 mostra un exemple de connexió d'un vertex c amb els seus nodes veïns. Inicialment el graf conté quatre components connexos. Després d'insertar la configuració c, trobem cinc candidats a nodes veïns que ordenem per distància creixent a c. Fixem-nos que en *(b)* la crida entre el node c i el node veí n_1 ha fracassat, i en canvi entre el node c i el node veí n_2 ha tingut èxit. Finalment destacar que en *(b)*, el node n_3 que inicialment no estava connectat al node c ara si que ho està.

La funció de distància. La funció de distància D és emprada tant per construir el conjunt de nodes veïns N_c com per ordenar-lo per cada nou node c insertat.

Una possibilitat, és definir $D(c,n)$ com una mesura (àrea o volum) de la regió de l'espai de treball escombrat pel robot quan aquest es mou al llarg d'un trajectòria calculada pel

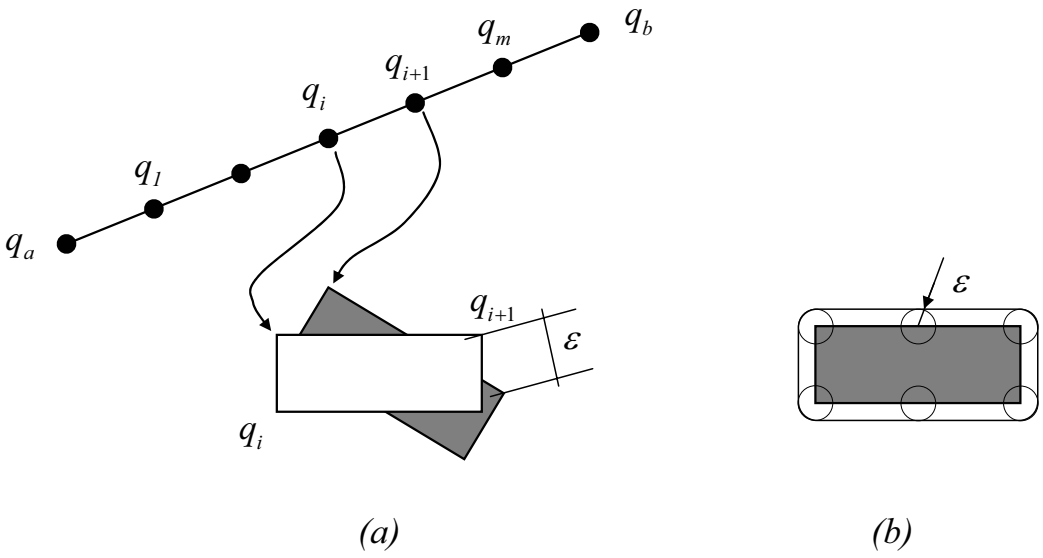


Figura 4.2: (a) El planificador local selecciona m configuracions de manera que entre dues de consecutives la distància màxima que s'ha desplaçat un mateix punt del robot és ϵ. $b)$ Model geomètric del robot inflat un valor ϵ per verificar la validesa del camí.

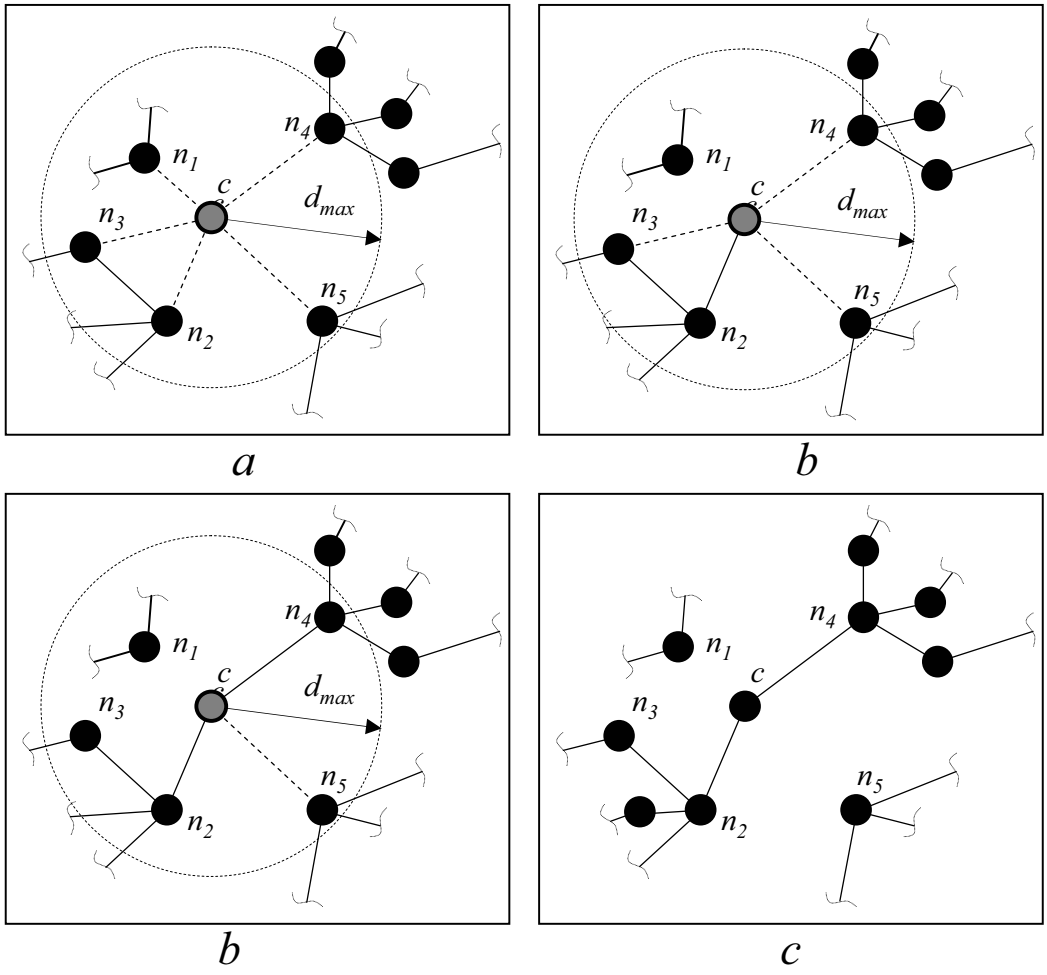


Figura 4.3: Exemple de connexió amb nodes veins.

planificador local entre dues configuracions c i n en absència d'obstacles. Com que calcular exactament l'àrea o el volum té un elevat cost computacional, moltes vegades s'utilitzen funcions més simples. Per exemple, es pot considerar la funció de distància ja definida en l'apartat 2.2.3,

$$D(c,n) = \max \parallel x(n), -x(c) \parallel \quad x \in robot, \tag{4.2}$$

on x representa un punt del robot, $x(c)$ és la posició de x en l'espai de treball quan el robot es troba a la configuració c, i $\parallel x(n) - x(c) \parallel$ és la distància Euclídia entre $x(c)$ i $x(n)$.

4.1.2 L'etapa d'expansió

Un cop acabada l'etapa de construcció ens podem trobar amb les següents situacions. En espais de treball simples, el graf R està ben connectat i és representatiu de l'espai lliure. Però en espais de treball més complexos, a on, per exemple, l'espai lliure constitueix un

únic component connex contenint passadissos estrets, el graf R sovint està format per alguns components connexos grossos i uns quants de petits. D'aquí es dedueix que l'etapa de construcció no ha estat prou eficient per capturar la connectivitat de l'espai lliure. Això s'esdevé, per exemple, quan els $\mathcal{C}$-obstacles defineixen passadissos estrets en l'espai lliure on és difícil que generem una configuració obtinguda per mostreig aleatori dels graus de llibertat del robot. Per aquest motiu, normalment el graf R està desconnectat en aquestes regions.

Una manera de millorar la connectivitat de R és mostrejar més densament l'espai de configuracions fins a obtenir una representació uniforme de l'espai lliure. Aquesta estratègia, però, pot suposar un cost computacional massa elevat, i per aquest motiu s'aplica una tècnica d'expansió com la següent.

La idea general és seleccionar un nombre de nodes de N que estan en regions d'elevada "dificultat", i després expandir-los. Entenem per expansionar un node el procés de: (1) generar una configuració lliure propera a ell, (2) afegir-la a N, i (3) intentar connectar-la a altres nodes de N tal com es fa en l'etapa de construcció . Per tant, l'etapa d'expansió incrementa la densitat de nodes en aquelles regions de l'espai lliure que hem considerat dificultoses. Com que els "forats" entre els components connexos del graf R resideixen en aquestes regions, la connectivitat del graf R es veu millorada. Per determinar quins nodes cal expandir, assignem un pes positiu a cada node c en N que doni una indicació de "la dificultat" de la regió que envolta c. Un pes elevat és indicatiu de que l'entorn de c és "difícil" i es trien els nodes de pes més elevat per ser expandits.

Hi ha vàries formes de definir $w(c)$, totes elles heurístiques. Vegem-ne algunes de les més habituals:

1. Comptabilitzar el nombre de nodes de N allunyats de c per sota d'una distància prefixada. Si aquest nombre és baix, probablement els obstacles ocupen bona part de les proximitats de c. Això suggereix que $w(c)$ pot ser definit inversament proporcional a aquest nombre de nodes.

2. Buscar quin és el component connex més proper a c i que no el conté. Per veure-ho, necessitem calcular la distància d_c del node c a qualsevol node d'altres components connexos, i escollir el valor mínim. Si aquesta distància és petita, llavors c es troba en una regió on dos components connexos no s'han pogut connectar, i això és un indicador que aquesta regió pot ser de dificultat elevada. El pes $w(c)$ es tria inversament proporcional a aquesta distància.

3. Definir $w(c)$ observant el comportament del planificador local. Per exemple, si el planificador local sovint fracassa en connectar c amb altres nodes, això és un indicador que c és en una regió d'elevada dificultat. D'acord amb això, hom defineix

$$r_f(c) = \frac{f(c)}{n(c)+1}, \tag{4.3}$$

on $n(c)$ és el nombre total de vegades que el planificador local intenta connectar c amb un altre node, i $f(c)$ és el nombre de vegades que fracassa. Cal afegir que, sempre que el planificador local fracassa en connectar dos nodes c i n, aquest fracàs

ha de ser comptabilitzat tant en $r_f(c)$ com en $r_f(n)$. Amb aquesta definició, el pes $w(c)$ es calcula com:

$$w(c) = \frac{r_f(c)}{\sum_{a \epsilon N} r_f(a)}. \tag{4.4}$$

Per expandir un node c, hem de calcular una *trajectòria aleatòria* partint de c i que sigui curta. Una trajectòria aleatòria es pot generar tot prenent repetidament una direcció aleatòria de l'espai de configuracions i desplaçant el robot en aquesta direcció fins que topa amb un obstacle (figura 4.4). Quan es produeix aquesta col·lisió, es torna enrere fins la darrera configuració lliure, i des d'aquesta triem una nova direcció aleatòria i així successivament. La configuració final n assolida així des de c s'emmagatzema com un nou node en N. L'aresta entre els nodes c i n s'afegeix a E i (a diferència de les arestes afegides durant l'etapa de construcció) s'hi desa la trajectòria obtinguda per anar de c a n, ja que aquesta, en ser aleatòria, no es podria recalcular.

Finalment, intentem connectar n a un altre component connex del graf R de la mateixa manera a com ho fèiem en l'etapa de construcció. Podem concloure doncs que la fase d'expansió mai crea un nou component connex en R. En el pitjor dels casos fracassa, i no es disminueixen el nombre de components connexos. Normalment, però, aquest nombre queda substancialment reduït.

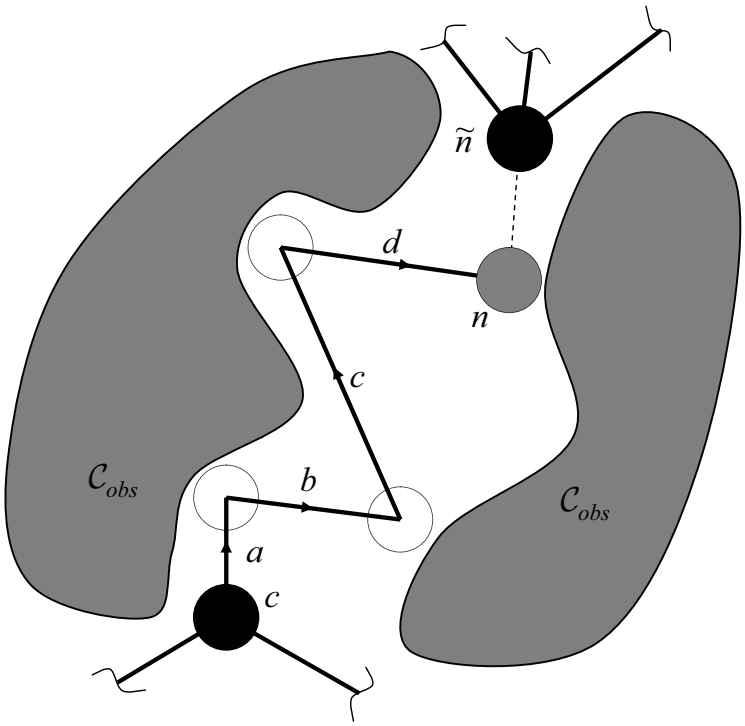


Figura 4.4: Trajectòria aleatòria generada en expansionar un node c. La trajectòria està formada pels trams a, b, c, d que són emmagatzemats en l'aresta (c, n). Finalment, el node n s'intenta connectar a altres nodes del graf, tal com es fa en l'etapa de construcció.

Una vegada la fase d'expansió ha finalitzat, s'eliminen els components connexos de R que són petits. Considerem que un component connex és petit si el seu nombre de nodes és inferior a un cert percentatge del nombre total de nodes en N (típicament 0.01%).

Per acabar, el lector pot preguntar-se sobre quins són els temps idonis, tan per l'etapa de construcció, com per la d'expansió. Naturalment, el rang de valors adequat per a cada un d'ells depèn de l'espai de treball, i haurien de ser determinats experimentalment. Tot i així, s'ha establert una relació entre els temps de construcció i els d'expansió que donen bons resultats per una gran part de problemes [17]. S'aconsella que la fase de construcció i expansió durin dos terços i un terç, respectivament, del temps total dedicat a la d'aprenentatge.

4.2 La fase d'interrogació

En la fase d'interrogació s'utilitza el mapa de rutes per determinar trajectòries lliures entre dues configuracions arbitràries. Per simplificar la descripció del procediment, suposarem que el graf R té un únic component connex, i que representa bé l'estructura de l'espai lliure.

La fase d'interrogació s'implementa normalment en tres etapes. Una primera que es redueix a connectar q_{ini} i q_{final} a dos nodes de N i buscar un camí entre aquests nodes. Una segona que reconstrueix la trajectòria lliure, si n'hi ha, com la concatenació de tres trajectòries: una trajectòria que connecta la configuració d'inici al mapa, una trajectòria continguda al mapa, i una trajectòria connectant el mapa amb la configuració final. I finalment la tercera, que s'encarrega mitjançant tècniques de suavització, d'escurçar la longitud de la trajectòria lliure calculada.

En la primera etapa, donada una configuració q_{ini} i una configuració final q_{final}, es tracta d'intentar connectar q_{ini} i q_{final} a dos nodes de N, anomenats q'_{ini} i q'_{final} respectivament, amb trajectòries lliures $P_{q_{ini}}$ i $P_{q_{final}}$. Si alguna d'aquestes connexions fracassa, el planificador respon que no ha pogut efectuar la connexió. En cas contrari, es calcula el camí existent P entre q'_{ini} i q'_{final}. Aquest camí es únic ja que R és un arbre i podem utilitzar qualsevol algorisme de cerca de camins per determinar-lo.

La qüestió principal és ara com calcular $P_{q_{ini}}$ i $P_{q_{final}}$. Com que les interrogacions haurien de tenir una resposta gairebé instantània, caldrà utilitzar un algorisme de baix cost computacional. Una estratègia habitual és seleccionar el conjunt $N_{q_{ini}}$ de nodes veïns a q_{ini} (seccio 4.1.1) i intentar connectar q_{ini} a cadascun d'ells fins que una connexió sigui exitosa. Si totes les connexions fracassen, generem una o unes quantes trajectòries aleatòries des de q_{ini}, però en lloc d'afegir el node del final de cada trajectòria aleatòria al mapa, intentem connectar-lo a N, tal com hem intentat inicialment per q_{ini}. Un cop connectat q_{ini} a N, fem el mateix procediment per connectar q_{final} a N.

Suposant que hem trobat $P_{q_{ini}}$, $P_{q_{final}}$ i P, ens trobem que $P_{q_{ini}}$ i $P_{q_{final}}$ són trajectòries lliures i en canvi P és només una seqüència de nodes i cal per tant transformar P en una trajectòria lliure. La reconstrucció de la trajectòria lliure a partir de P es redueix a la concatenació de trajectòries lliures entre nodes adjacents de P. Algunes d'aquestes trajectòries poden haver estat produïdes durant l'etapa d'expansió, i per tant poden estar emmagatzemades explícitament en arestes de E. Per altra banda, les arestes afegides

durant l'etapa de construcció no contenen la trajectòria lliure explícitament i aquesta ha de ser recalculada. Això és possible perquè el planificador local que hem considerat és determinista i produirà les mateixes trajectòries cada vegada que sigui cridat per les mateixes configuracions d'entrada.

Un cop concatenades les tres trajectòries $P_{q_{ini}}$, P, i $P_{q_{final}}$, obtenim una trajectòria lliure entre la configuració inicial q_{ini} i final q_{final}, que pot ser millorada aplicant una tercera etapa de suavització. Es tracta d'aplicar tècniques de suavització de trajectòries, com per exemple seleccionar segments aleatoris de la trajectòria i unir-los emprant el planificador local, o bé iterativament retallant les cantonades que es formen en algunes parts de la trajectòria, entre d'altres. Normalment, aquestes tècniques són desenvolupades per a classes específiques de robots, i per aquest motiu explicarem amb més detall les que s'ajusten al present projecte en capítols posteriors.

Fins ara hem descrit la fase d'interrogació considerant que el graf R consistia en un únic component connex. En general, però, el graf R pot consistir en uns quants components connexos R_i, $i = 1, 2, .., p$. Aquest és el cas quan l'espai lliure no es troba connectat. En aquestes situacions s'ha de tenir en compte un altre factor en la primera etapa. Suposant que les configuracions d'inici q_{ini} i q_{final} s'han connectat al graf R a dos nodes q'_{ini} i q'_{final} respectivament, ens hem d'assegurar que aquests últims pertanyen al mateix component connex, ja que si no és així no hi haurà solució, i per tant la fase d'interrogació fracassarà.

Per acabar, si les interrogacions fracassen freqüentment, és possible que el mapa no capturi adequadament la connectivitat de l'espai lliure. Per tant, s'hauria de dedicar més temps a la fase d'aprenentatge. Com que aquesta fase es pot fer incrementalment, simplement podem estendre el mapa actual reprenent sobre ell l'etapa de construcció i/o d'expansió.

4.3 Completesa probabilística

Un dels principals punts febles de la planificació probabilística és la no completesa de l'algorisme resultant: no es pot garantir que l'algorisme trobi una trajectòria lliure sempre que aquesta existeixi. Tanmateix, demostrarem a continuació que la probabilitat que l'algorisme no trobi tal trajectòria tendeix a zero a mida que el nombre de mostres obtingudes de $\mathcal{C}_{lliure}$ tendeix a infinit. Es diu així que l'algorisme és almenys *probabilísticament complet*. Per demostrar aquesta propietat, determinarem una cota superior de tal probabilitat per a un espai de configuracions de dues dimensions. La demostració per espais de més dimensions és de fet una generalització trivial de la que donarem a continuació [16].

Suposem que tenim un espai de configuracions de dues dimensions $\mathcal{C} = [0, 1]^2$, amb m mostres distribuïdes aleatòriament sobre $\mathcal{C}_{lliure}$ on cada mostra correspon a una configuració lliure del robot. A més, suposem que podem conèixer la distància des d'un punt x qualsevol de $\mathcal{C}_{lliure}$ als $\mathcal{C}$-obstacles, denotada per $r(x)$. És a dir:

$$r(x) = inf\,|x - y|, \quad \text{amb} \quad y \in \mathcal{C}\text{-obstacles}. \tag{4.5}$$

El nostre propòsit és trobar una cota de la probabilitat de fracàs en connectar dos punts a i b de $\mathcal{C}_{lliure}$ a partir de les m mostres i la geometria de l'espai, tot sabent que existeix una trajectòria γ entre aquests dos punts,

$$\gamma : [0, L] \rightarrow \mathcal{C}_{lliure}, \quad \text{a on} \quad \gamma(0) = a \quad \text{i} \quad \gamma(L) = b, \tag{4.6}$$

on tots els seus punts estan separats almenys per una certa distància R dels $\mathcal{C}$-obstacles (figura 4.5). És a dir,

$$R = inf\{r(\gamma(t))\}, \quad \text{amb} \quad 0 \leq t \leq L. \tag{4.7}$$

Suposarem també que la corba γ bé parametritzada per la longitud del seu arc i que per tant es compleix que

$$d(s,t) = \int_s^t \|\gamma'(t)\| \, d\bar{t} = |s - t|, \quad \forall s, t \in [0, L]. \tag{4.8}$$

on $d(s,t)$ és la longitud de l'arc de corba comprès entre $\gamma(s)$ i $\gamma(t)$.

Per obtenir la cota de probabilitat desitjada, en primer lloc podem establir una partició de la corba γ en n trossos, sent $n = \lceil \frac{L}{R/2} \rceil$, delimitats per $n+1$ punts $x_0 = a, x_1, \cdots, x_n = b$, de paràmetres $t_o, \cdots, t_n$, respectivament. Aquests punts es trien de manera que:

$$d(t_j, t_{j+1}) \leq \frac{R}{2} \quad \text{per} \quad j \in [0, n-1]. \tag{4.9}$$

A la inequació 4.9, la desigualtat és produeix quan la fracció $\frac{L}{R/2}$ no és un valor sencer i aleshores s'arrodoneix pel natural per excés mitjançant l'operador $\lceil \ \rceil$.

Si $B_R(x)$ denota la bola de radi R centrada en $x \in \mathcal{C}$, la inequació 4.9 implica la següent relació:

$$B_{\frac{R}{2}}(x_{j+1}) \subseteq B_R(x_j), \quad \text{per} \quad j = 0, \cdots, n-1, \tag{4.10}$$

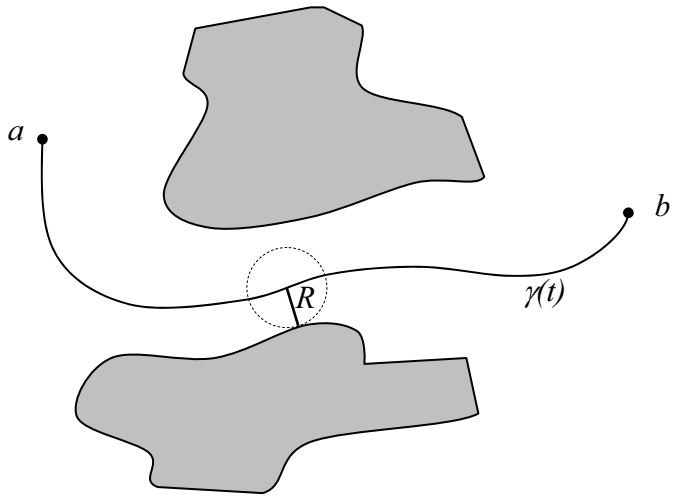


Figura 4.5: Trajectòria existent entre dos punts a i b de $\mathcal{C}_{lliure}$.

que es demostra tot veinet que, gràcies a la desigualtat triangular, $\forall\, p \in B_{\frac{R}{2}}(x_{j+1})$ tenim

$$\|x_j - p\| \leq \underbrace{\|x_j - x_{j+1}\|}_{c_1} + \underbrace{\|x_{j+1} - p\|}_{c_2},$$

però de la inequació 4.9 és:

$$c_1 = \|x_j - x_{j+1}\| \leq d(t_j, t_{j+1}) \leq \frac{R}{2},$$

i per ser $p \in B_{\frac{R}{2}}(x_{j+1})$ tenim:

$$c_2 \leq \frac{R}{2}.$$

Així doncs, escollint la cota superior de c_1 i c_2 arribem a $\|x_j - p\| \leq \frac{R}{2} + \frac{R}{2} = R$ que prova la relació 4.10.

Si ara prenem $c \in B_{\frac{R}{2}}(x_j)$ i $d \in B_{\frac{R}{2}}(x_{j+1})$, observem que el segment recte $\bar{cd}$ és lliure, ja que ambdós estan continguts en la mateixa bola lliure $B_R(x_j)$ (figura 4.7).

De l'afirmació anterior deduïm que per a que el nostre algorisme pugui connectar a i b, cal almenys que en cada bola $B_{\frac{R}{2}}(x_j)$, contingui una de les m mostres generades aleatòriament. D'aquesta forma podem assegurar que els punts de connexió entre aquestes boles també estaran inclosos en $\mathcal{C}_{lliure}$. Utilitzant una relació d'àrees podem deduir la probabilitat de que un punt q_i no estigui contingut en $B_{\frac{R}{2}}(x_j)$ és $\left(1 - \frac{|B_{R/2}|}{|\mathcal{C}_{lliure}|}\right)$ sent $|B_{\frac{R}{2}}|$ i $|\mathcal{C}_{lliure}|$ les àrees dels respectius conjunts. Com que les m mostres s'han generat de forma independent, la probabilitat de que cap mostra q_i estigui continguda en una determinada bola $B_{\frac{R}{2}}(x_j)$ és $\left(1 - \frac{|B_{R/2}|}{|\mathcal{C}_{lliure}|}\right)^m$.

Finalment la probabilitat de fracàs buscada és la suma de les probabilitats de que cadascuna de les boles $B_{\frac{R}{2}}(x_j)$ estigui buida, per $j = 1, \cdots, n-1$:

$$\begin{aligned} P_r(\text{fracàs}) &\leq P_r(\text{alguna bola sigui buida}) \\ &\leq \underbrace{\left(1 - \frac{|B_{R/2}|}{|\mathcal{C}_{lliure}|}\right)^m + \cdots + \left(1 - \frac{|B_{R/2}|}{|\mathcal{C}_{lliure}|}\right)^m}_{n-1} \\ &= (\lceil \frac{2L}{R} \rceil - 1)\Big(1 - \frac{|B_{R/2}|}{|\mathcal{C}_{lliure}|}\Big)^m. \end{aligned} \tag{4.11}$$

Com que l'espai de configuracions és de dimensió 2 tenim que $|B_{\frac{R}{2}}| = \frac{\pi R^2}{4}$ i arribem a:

$$P_r(\text{fracàs}) \leq (\frac{2L}{R})\Big(1 - \frac{\pi R^2}{4|\mathcal{C}_{lliure}|}\Big)^m. \tag{4.12}$$

Com que R és el radi d'una bola totalment continguda en l'espai $\mathcal{C}_{lliure}$, el valor de l'interior del parèntesi dret de la inequació 4.12 sempre serà més petit que 1 per tant

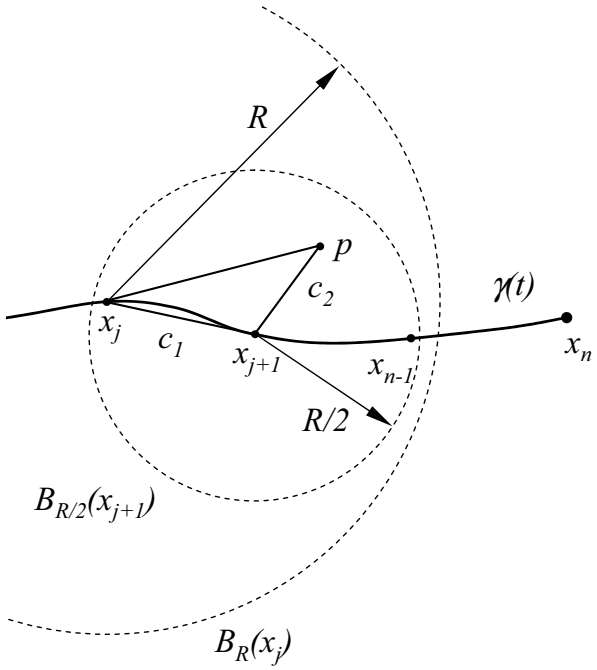


Figura 4.6: Relació de distàncies que s'estableix entre x_j, x_{j+1} i un punt p qualsevol de $B_{\frac{R}{2}}(x_{j+1})$.

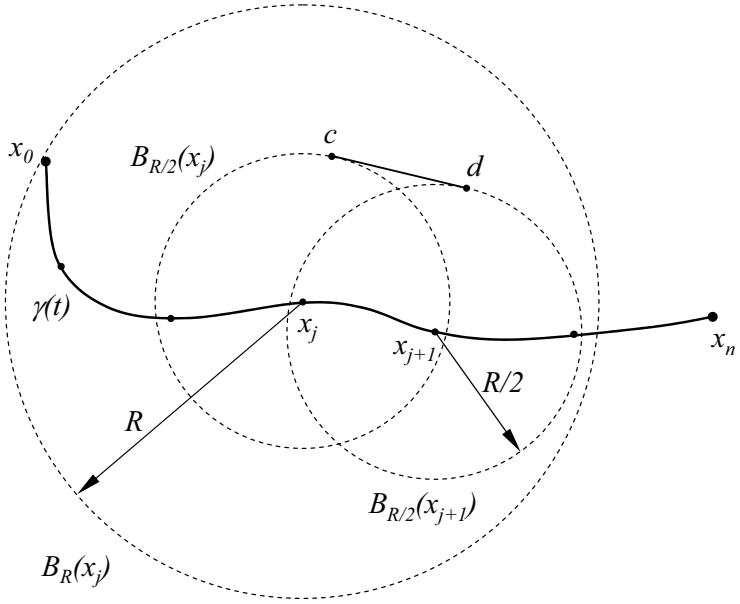


Figura 4.7: Representació gràfica de la relació 4.10

$$\lim_{m\to\infty}\left(1-\frac{\pi R^2}{4|\mathcal{C}_{lliure}|}\right)^m=0, \tag{4.13}$$

demostrant així que la probabilitat de que l'algorisme fracassi tendeix a zero quan m augmenta.

Cal dir que l'anterior cota de probabilitat té una utilitat pràctica relativa, ja que no es pot fer servir per calcular el nombre de mostres desitjables, en no saber-se els paràmetres R i L a priori. No obstant, la cota dóna almenys una idea de la relació entre la probabilitat de fracàs i R, L i m. Demostra per exemple que la probabilitat de fracàs tendeix ràpidament a zero, degut a la dependència exponencial amb m, i que l'existència de camins molt llargs té poca repercussió, ja que la dependència amb L és lineal.

Capítol 5

Mapes probabilístics per cadenes cinemàtiques tancades

El mètode desenvolupat al capítol anterior es pot aplicar directament per planificar els moviments de robots modelitzables com un únic cos, o els de sistemes robotitzats més complexos, formats per múltiples cossos articulats entre sí, però sense formar cadenes cinemàtiques tancades. Tanmateix, en aquest capítol veurem que el mètode es pot adaptar al cas d'aquest projecte, tot introduint algunes modificacions. Veurem primer quines dificultats afegeix el fet que els dos robots estiguin simultàniament agafant un objecte, quina estructura té l'espai de configuracions que en resulta, algunes estratègies que es poden adoptar per efectuar les modificacions esmentades, i la solució que finalment s'ha implementat.

5.1 Satisfacció d'equacions de clausura

En el problema bàsic del capítol 2 s'assumia que el robot era modelitzable com un sòlid rígid lliure-volant. Sovint, però, el robot està format per diversos sòlids rígids $\mathcal{A}_1, ..., \mathcal{A}_p$ connectats entre sí per articulacions, que poden ser de revolució, prismàtiques, esfèriques, etc. Cada articulació restringeix el moviment relatiu dels dos sòlids que connecta, i porta associada una coordenada generalitzada q_i.

Anomenem *coordenades generalitzades* les variables geomètriques q_i de posició i orientació emprades per descriure la configuració d'un sistema mecànic. Habitualment, aquestes coordenades són els angles relatius entre sòlids adjacents o bé els angles que formen els sòlids respecte una referència absoluta. Un exemple de mecanisme format per articulacions de revolució seria la cadena cinemàtica que es mostra a la figura 5.1. L'objectiu és trobar els valors d'aquestes coordenades que fan que el mecanisme mantingui tancada la cadena (figura 5.2).

Aquest conjunt de solucions es troba a partir de les equacions de clausura que defineixen la cadena tancada. Determinar-les no és una tasca fàcilment sistematitzable, excepte en el cas que s'utilitzin coordenades referencials per a cada membre (6 a l'espai i 3 en el pla). A continuació mostrem els passos que habitualment es segueixen per determinar aquestes equacions.

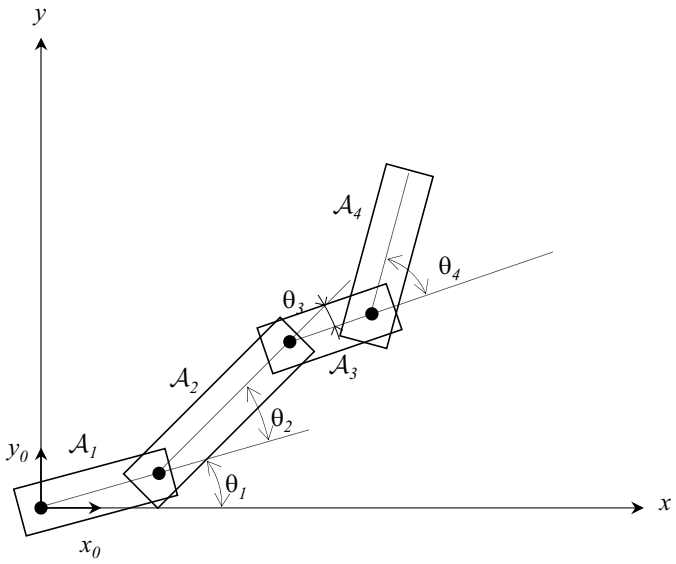


Figura 5.1: Un exemple de cadena cinemàtica. que està formada per articulacions de revolució. Els angles de les articulacions es mesuren entre sòlids consecutius de la cadena.

1. **Creació d'un graf de sòlids i restriccions**. Primer de tot hem de fer una abstracció del mecanisme articulat, i crear un graf G_M on cada node representa una referència solidària a un dels sòlids, i cada arc la transformació homogènia que relaciona els sistemes de referència de dos sòlids articulats entre sí (annex A). Per simplificar l'estudi, considerem que un dels sòlids del mecanisme es troba fix al món, i que per tant s'han suprimit les translacions i rotacions "rígides" del conjunt. Com a exemple, la figura 5.3 mostra el graf de restriccions del mecanisme de la figura 5.2.

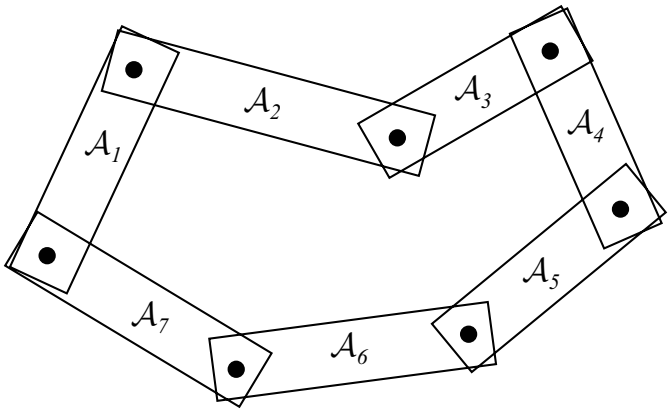


Figura 5.2: Mecanisme pla de barres articulades.

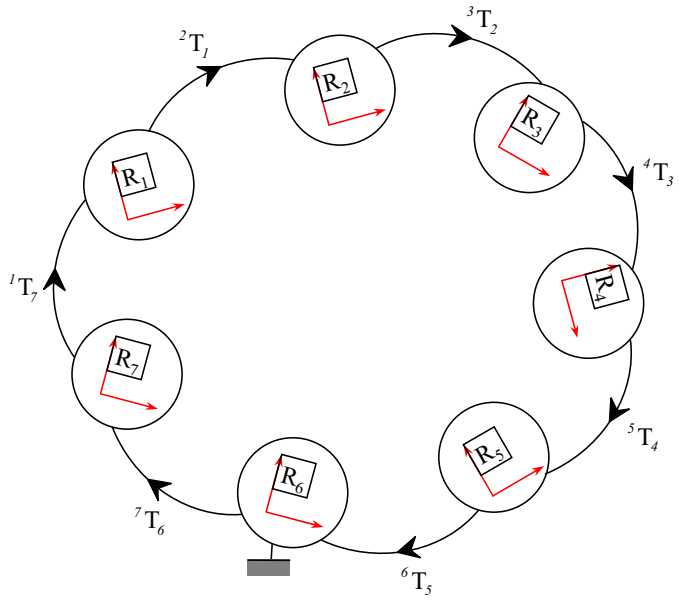


Figura 5.3: Graf de sòlids i restriccions del mecanisme de la figura 5.2.

Cada ${}^iT_{i-1}$ $(i = 2, .., 7)$ i ${}^1T_7$ és la següent matriu de transformació homogènia:

$$ {}^iT_{i-1} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & x_{ti} \\ \sin\theta_i & \cos\theta_i & y_{ti} \\ 0 & 0 & 1 \end{pmatrix} \tag{5.1} $$

2. **Determinació de l'equació matricial trigonomètrica**. Quan el mecanisme presenta una cadena tancada, es pot veure que si fem el producte de transformades partint d'una referència d'un sòlid fins a tancar el cicle, el producte esdevindrà la matriu identitat. Aquest sistema trigonomètric caracteritza el conjunt de configuracions que poden adoptar els sòlids involucrats per mantenir tancat el cicle. En l'exemple ha de ser:

$$ {}^2T_1\,{}^3T_2\,{}^4T_3\,{}^5T_4\,{}^6T_5\,{}^7T_6\,{}^1T_7 = I \tag{5.2} $$

3. **Algebrització**. Per resoldre l'anterior sistema, cal convertir-lo a un sistema d'equacions polinòmic, fent el canvi de variable de la tangent de l'angle mig, o substituint $x_i = sin(\theta_i)$, $y_i = cos(\theta_i)$, i afegint la restricció $x_i^2 + y_i^2 = 1$ per cada angle.

4. **Resolució del Sistema**. El sistema polinòmic resultant es pot resoldre finalment utilitzant tècniques de Geometria Algebraica, com les bases de Gröbner o mètodes d'eliminació, o tècniques numèriques de continuació o cerca per intervals.

5.2 L'espai de configuracions

Un robot lliure-volant pot ser posicionat i orientat de forma qualsevol en un espai de treball en absència d'obstacles. Utilitzant una correcta parametrització dels seus graus de llibertat, es pot comprovar que l'espai de configuracions $SE(3)$ és una varietat diferencial, una propietat que seria desitjable trobar en l'espai de configuracions de tot sistema a planificar. Intuïtivament, significa que a cada punt, l'espai de configuracions es comporta com si fos una "superfície" on la seva diferencial està definida.

Per a un mecanisme articulat amb n coordenades generalitzades, una configuració ve definida per un vector q de n components, pertanyent a un determinat espai n-dimensional $\mathcal{C}$. El vector determina de forma única la posició i orientació de tots els sòlids del mecanisme. L'espai de configuracions del mecanisme, denotat per $\mathcal{C}_{clausura}$ és només un subconjunt de C, consistent en el conjunt de punts q que són solució del sistema algebraic de la secció anterior. Formalment,

$$\mathcal{C}_{anell} = \{q \epsilon \mathcal{C} \,/\, f(q) = I\} \tag{5.3}$$

on $f(q) = I$ és el producte de transformacions homogènies igualades a la identitat a què fèiem esment.

Finalment, per ambdós tipus de configuracions, obertes o tancades, les configuracions no podran presentar col·lisions amb cap obstacle o qualsevol part del mateix robot. Així doncs anomenem $\mathcal{C}_{lliure}$ el conjunt de configuracions $q \in \mathcal{C}$ que no causen cap col·lisió en el sistema. Usant aquesta notació, per un mecanisme en cadena tancada el problema de planificació pot ser definit així: Donada les configuracions q_{ini} i q_{fi}, trobar una trajectòria $q(t)$, $t \in [0,1]$, tal que $q(0) = q_{ini}$, $q(1) = q_{fi}$, i $\forall\, t \in [0,1]$, $q(t) \in \mathcal{C}_{clausura} \bigcap \mathcal{C}_{lliure}$.

Encara que $\mathcal{C}$ és típicament una varietat diferencial, $\mathcal{C}_{clausura}$ té en general una estructura més complexa, ja que p ot contenir punts on la diferencial no és definida, anomenats singularitats. Una varietat algebraica, per tant, no és necessàriament diferencial, encara que pot ser subdividida en un nombre finit de varietats diferencials. El problema de la planificació es complica enormement en aquests casos, ja que, en general, pot no ser possible trobar una parametrització de $\mathcal{C}_{clausura}$

5.3 Estratègies de planificació

Per estendre el mètode explicat al capítol anterior al cas de dos robots cooperants, cal introduir modificacions en la generació de configuracions aleatòries i en la generació de camins locals. Dues són les estratègies seguides fins ara, degudes a Yakey, LaValle i Kavraki per una banda [38], i a Han i Amato per l'altra [14]. La primera és aplicable a mecanismes amb múltiples cadenes cinemàtiques tancades i s'assumeix que no es disposa de la solució inversa per cap subcadena. En la segona s'assumeix que sí es coneix aquesta solució, i està més orientada a sistemes amb una única cadena tancada.

5.3.1 Mètode de Yakey, Lavalle i Kavraki

En general, per a un mecanisme articulat M de q graus de llibertat, existeixen certs valors de q que fan que el mecanisme presenti cadenes tancades (figura 5.4). El conjunt

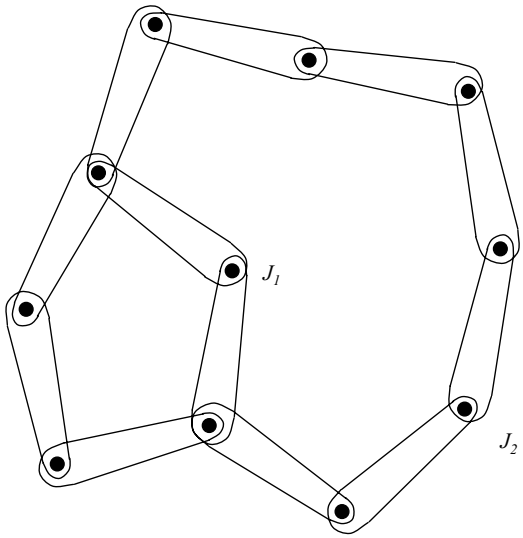


Figura 5.4: Mecanisme amb dues cadenes cinemàtiques tancades.

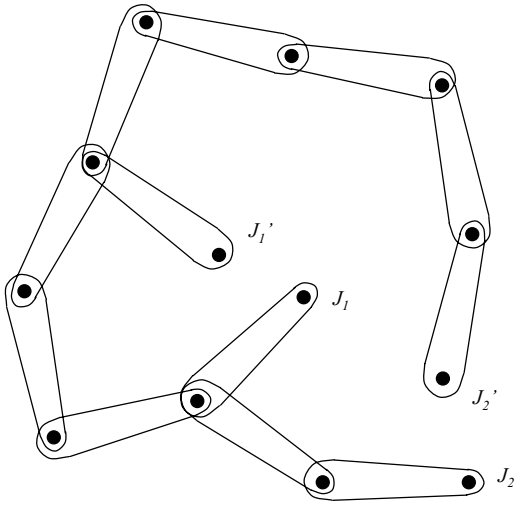


Figura 5.5: Mecanisme obert en dues articulacions.

$\mathcal{F} = \{f_1(q) = 0, ..., f_m(q) = 0\}$ està integrat per m equacions de clausura que haurien de satisfer cadascuna d'aquestes cadenes.

En lloc de treballar amb aquestes equacions de clausura, la idea és treballar amb una *funció cinemàtica d'error* de manera que s'aproximi l'espai $\mathcal{C}_{clausura}$ per un nou espai $\tilde{\mathcal{C}}_{clausura}$ de més fácil manipulació. Per obtenir-la cal trencar cada un dels anells per una articulació qualsevol, com es mostra a la figura 5.5. Com a resultat s'obtenen dues articulacions noves per a cada anell trencat, donant lloc a un nou mecanisme M' associat a M. Si analitzem el graf de restriccions $G_{M'}$ de M', veiem que és un arbre (figura 5.5).

Es satisfan totes les equacions de clausura de M quan J_k i J'_k tenen la mateixa posició en W. Si això es compleix per a totes les articulacions, J_k i J'_k de cada cadena tanca-

da, llavors les configuracions romandran en $\mathcal{C}_{clausura}$. Per a cada articulació J_k i J'_k del mecanisme M' es busca l'expressió cinemàtica $J_k(q)$ i $J'_k(q)$ a partir del producte de transformades partint d'una articulació de cada anell. La funció cinemàtica d'error es defineix com:

$$e(q) = \sum \|J_k(q) - J'_k(q)\| \tag{5.4}$$

on k indica l'índex d'una articulacions d'M que s'ha obert. Aquesta funció d'error defineix $C_{clausura}$ com:

$$\mathcal{C}_{clausura} = \{q \,\epsilon\, \mathcal{C} \,|\, e(q) = 0\} \tag{5.5}$$

Però com que treballem amb una certa tolerància $\epsilon > 0$ definim:

$$\tilde{\mathcal{C}}_{clausura} = \{q \,\epsilon\, \mathcal{C} \,|\, e(q) \leq \epsilon\} \tag{5.6}$$

Utilitzant aquesta tolerància el cost computacional es inferior al que suposaria resoldre directament el conjunt de restriccions $\{f_1(q) = 0, ..., f_m(q) = 0\}$. Ara, per generar configuracions aleatòries, només cal mostrejar l'espai $\mathcal{C}$ i mitjançant un algoritme iteratiu tipus Newton-Raphson refinar-les fins a convergir al conjunt $\mathcal{C}_{clausura}$ amb un error inferior a ϵ.

Per generar camins locals, podem buscar configuracions pròximes a una de donada utilitzant *l'espai tangent*: el conjunt de vectors tangents per algun $q \in \mathcal{C}_{clausura}$. Per determinar-lo cal calcular el Jacobià de les equacions de clausura $\{f_1(q) = 0, ..., f_m(q) = 0\}$. Si ara volem generar una nova configuració propera a una donada, en lloc de fer-ho de forma aleatòria podem utilitzar aquest espai tangent de forma que la probabilitat que pertanyi a $\tilde{\mathcal{C}}_{clausura}$ sigui més alta. El que fem resoldre el següent sistema d'equacions per a cada configuració q desitjada:

$$\begin{pmatrix} \frac{\partial f_1(q)}{\partial q_1} & \frac{\partial f_1(q)}{\partial q_2} & \cdots & \frac{\partial f_1(q)}{\partial q_n} \\ \frac{\partial f_2(q)}{\partial q_1} & \frac{\partial f_2(q)}{\partial q_2} & \cdots & \frac{\partial f_2(q)}{\partial q_n} \\ \vdots & \vdots & & \\ \frac{\partial f_m(q)}{\partial q_1} & \frac{\partial f_m(q)}{\partial q_2} & \cdots & \frac{\partial f_m(q)}{\partial q_n} \end{pmatrix} \begin{pmatrix} dq_1 \\ dq_2 \\ \vdots \\ dq_n \end{pmatrix} = 0 \tag{5.7}$$

Cal recordar que $m < n$. Si el rang de la matriu es $k \leq m$, aleshores $n - k$ coordenades generalitzades poden ser triades independentment i la resta es troben a partir de 5.7. Finalment a partir de la configuració diferencial solucionada $(dq_1, ..., dq_n)$, la nova configuració està a $(q_1 + dq_1, ..., q_n + dq_n)$. Aquesta nova configuració és vàlida si pertany al conjunt $\tilde{\mathcal{C}}_{clausura}$ i està lliure de col·lisions. Aquesta tècnica, però, suposa un cost computacional elevat, pel fet que s'han de calcular les derivades parcials per cada una de les nostres restriccions.

Finalment, només resta veure com es pot generar un camí local que uneixi dues configuracions q i q' properes. El planificador local utilitzat intenta la connexió emprant un mètode de descens guiat per gradient. Consisteix bàsicament en intentar reduir la distància $\rho(q, q')$, triant direccions de l'espai tangent adequades, i noves configuracions intermitges que mantinguin l'error cinemàtic dintre d'un marge ϵ. L'èxit de l'algorisme es basa en l'assumpció que els vèrtexs q i q' estan suficientment propers com per assegurar que problemes de mínims locals i de col·lisions seran bastant improbables.

El mètode anterior presenta com a inconvenient principal el seu elevat cost computacional, derivat principalment del fet que és un mètode general que no fa cap assumpció respecte del sistema robotitzat tractat. És fàcil veure que moltes de les mostres inicialment triades sobre $\mathcal{C}$ acaben no convergint cap a $\mathcal{C}_{clausura}$ perquè n'estan molt allunyades. Quan es disposa de la solució explícita del problema cinemàtic invers és millor recórrer a mètodes més simples, com el desenvolupat per Han i Amato [14], que expliquem a continuació.

5.3.2 Mètode de Han i Amato

Per tal de generar configuracions que satisfacin les condicions de clausura cinemàtica, aquest mètode trenca el mecanisme en dues subcadenes obertes, tal com mostra la figura 5.6. Els graus de llibertat per una cadena són anomenats coordenades actives, i per l'altra coordenades passives. Ara només cal generar valors aleatoris per les coordenades actives, i per cinemàtica inversa trobar les solucions de les coordenades passives. De fet, per cada conjunt de coordenades actives podem trobar diferents solucions que satisfan la clausura cinemàtica. Cada una d'aquestes solucions està associada a un tipus de configuració diferent de la cadena.

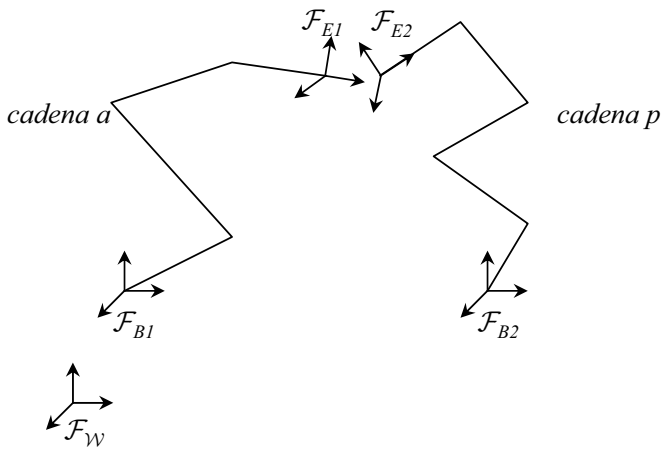


Figura 5.6: Trencament d'un cadena tancada en dues cadenes obertes. La referència $\mathcal{F}_{\mathcal{W}}$ és l'absoluta de l'espai de treball, les referències $\mathcal{F}_{Bi}$ es diuen de base, perquè són solidàries al primer element de cada una de les cadenes que s'han obtingut. Finalment les referències $\mathcal{F}_{Ei}$ sòn les solidàries a l'element final de cada cadena.

Aquest mètode suposa un notable estalvi computacional en sistemes on la cinemàtica inversa té sol·lució tancada, ja que per una generació de coordenades actives podem trobar molt ràpidament diferents configuracions per la part passiva. Per a cada configuració s'ha de comprovar si és o no lliure de col·lisions. Si ho és, s'emmagatzema com un node del mapa de rutes, obtenint múltiples configuracions classificades segons el tipus de solució de la cadena passiva.

Per altra banda, el procés per generar connexions locals entre nodes no difereix massa de l'utilitzat en l'esquema de la planificació probabilística estàndard. Es crida el plani-

ficador local per a la cadena associada a les coordenades actives i per cada configuració intermitja del camí generat es resol la cinemàtica inversa de la cadena passiva. Si hi ha solució es mira si la configuració tancada és lliure de col·lisions. El planificador local retorna "èxit" si ha pogut establir la connexió sense col·lisionar amb cap obstacle. A l'hora de fer les connexions cal tenir en compte que només podem intentar connectar configuracions que corresponguin a solucions cinemàtiques del mateix tipus per la cadena passiva.

5.4 Implementació per a dos manipuladors simples

L'estratègia de Han i Amato, convenientment posada en el context dels algorismes explicats en el capítol 4, és la que millor s'escau per a desenvolupar el planificador requerit en aquest projecte, ja que permet aprofitar el fet que es coneix l'expressió tancada de la cinemàtica inversa dels dos robots considerats. L'estructura principal dels algorismes es dedueix fàcilment dels ja donats en aquell capítol, convenientment modificats per adaptar-los a l'estratègia de la secció anterior. En aquesta secció detallarem tots els aspectes d'implementació que encara cal precisar, en cadascuna de les fases de què consta el mètode.

5.4.1 Sistema considerat

L'espai de treball consta dels següents elements: dos robots manipuladors RX60, un objecte a transportar, i els obstacles. Les bases dels robots són fixades al terra, així com els obstacles, i les seves posicions i orientacions respecte al sistema de referència absolut són conegudes. Desitgem transportar l'objecte amb l'ajuda dels robots, és a dir, quan aquests juntament amb l'objecte formen una cadena cinemàtica tancada (secció 5.1). En consonància amb la notació introduïda al capítol 2, denotarem el conjunt dels robots i l'objecte per $\mathcal{A}$, els obstacles individuals per $\mathcal{B}_i$. Sovint parlarem de configuracions de $\mathcal{A}$ i direm que són cinemàticament vàlides si els dos robots estan simultàniament agafant l'objecte a transportar, satisfent per tant la condició de clausura cinemàtica associada.

Cal esmentar que només intentarem resoldre el problema de transportar un objecte des d'una posició inicial a una final de l'espai de treball, subjecte simultàniament als dos robots, i que en cap moment resoldrem el problema de com cal desplaçar els robots des de les seves posicions prèvies a la situació de prensió inicial. Aquest problema ja es pot resoldre amb els planificadors per robots articulats estàndard que hom troba a la literatura.

5.4.2 Etapa de construcció

Recordem de la secció 4.1.1 que aquesta etapa consisteix en la construcció d'un graf $R = (N, E)$, on els nodes de N emmagatzemen configuracions lliures del robot, generades aleatòriament, i les arestes de E indiquen crides exitoses del planificador local entre parelles de configuracions. Afegim tot seguit els detalls necessaris per obtenir aquest graf.

5.4.2.1 Generació de configuracions aleatòries

S'han considerat dues maneres de generar configuracions aleatòries, cinemàticament vàlides. La primera consisteix en generar posicions i orientacions aleatòries de l'objecte, i després resoldre el problema cinemàtic invers per cada robot, i la segona consisteix en generar un vector de sis angles aleatòriament triats per un dels robots, i després tancar la cadena resolent la cinemàtica inversa pel robot restant.

La segona opció permet generar configuracions aleatòries del robot dins el seu espai cinemàtic, ja que coneixem els intervals de treball de les seves articulacions (taula C.2). En canvi, en la primera opció, la tria dels intervals dels paràmetres per l'objecte és bastant arbitrària. Si triem uns intervals massa grans podem estar obtenint configuracions aleatòries de l'objecte en zones on no es pot produir la premsió simultània. Per altra banda, triant intervals massa estrets, pot ser que només aconseguim mostrejar un subconjunt de tot l'espai cinemàtic dels robots.

Un avantatge addicional de la segona opció és que obté una probabilitat de tancament major que la primera, gràcies a que només s'ha de resoldre un problema cinemàtic invers. De fet, empíricament hem obtingut una probabilitat d'èxit pel segon cas de l'ordre del 5%, que és gairebé el doble que mostrejant amb la primera opció, on la probabilitat d'èxit es situa al voltant del 3%. Clarament, el segon mètode és preferible, ja que permet mostrejar l'espai lliure molt més ràpidament.

Seguint la nomenclatura de Han i Amato, assumirem que el robot actiu és l'esquerre, i el passiu el dret. Una configuració aleatòria del robot actiu ve donada per un vector format pels angles de les seves sis articulacions, triats aleatòriament. L'angle d'una articulació es tria aleatòriament, dins el rang d'angles permesos per la mateixa, mostrejant amb densitat de probabilitat uniforme.

Un cop obtingut el vector d'angles aleatoris pel robot actiu, hem de trobar els paràmetres de posició i orientació de l'objecte, de manera que quedi situat en la seva pinça. A tal fi, prèviament cal calcular la matriu de transformació ${}^{obj}T_R$ a partir de les diferents transformacions conegudes del sistema. De la figura 5.7 podem deduir que

$$ {}^{obj}T_R = {}^{0_a}T_R \, {}^{TCP_a}T_{0_a} \, {}^{obj}T_{TCP_a}. \tag{5.8} $$

Per posicionar l'objecte espaialment cal resoldre la cinemàtica inversa de l'objecte a partir de ${}^{obj}T_R$. Finalment, per establir el tancament total, hem de resoldre la cinemàtica inversa del robot passiu, i per tant prèviament haurem de calcular ${}^{TCP_p}T_{0_p}$, que es pot deduir de l'esquema inclòs a la figura 5.8:

$$ {}^{TCP_p}T_{0_p} = {}^{R}T_{0_p} \, {}^{obj}T_R \, {}^{TCP_p}T_{obj}. \tag{5.9} $$

Procedint com s'ha explicat acabem obtenint els angles de les 12 articulacions en joc, que necessàriament han de satisfer la condició de clausura cinemàtica del conjunt. Fet això, es comprova si tots els sòlids en joc col·lisionen o no entre ells o amb els obstacles de l'entorn i, en cas negatiu, la configuració es pot ja inserir al mapa de rutes. Aquesta darrera comprovació es farà tot utilitzant un detector de col·lisions estàndard ja implementat en el paquet de gestió gràfica utilitzat pel desenvolupament del projecte.

5.4.2.2 El planificador local

Tal com s'ha explicat a la secció 4.1.1, el planificador local intenta connectar cada nova configuració c amb les configuracions més properes ja inserides al mapa de rutes.

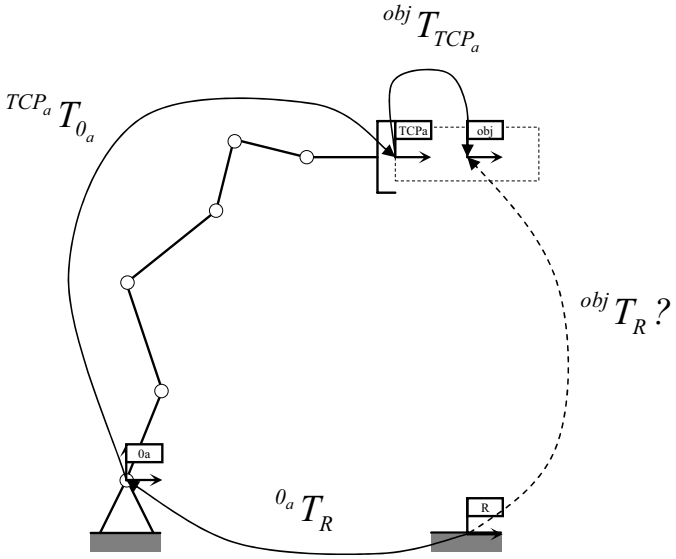


Figura 5.7: Relació de transformacions homogènies del robot actiu. ${}^{0_a}T_R$ és coneguda ja que coneixem la posició i orientació de les bases de cada un dels robots respecte la referència absoluta. ${}^{TCP_a}T_{0_a}$ es pot obtenir per cinemàtica directa a partir del vector d'angles del robot actiu i ${}^{obj}T_R$ també és una dada ja que l'usuari especifica com vol agafar l'objecte.

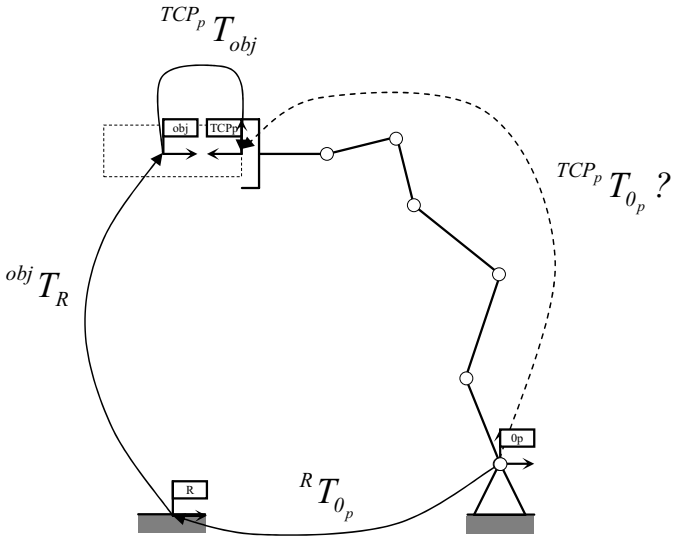


Figura 5.8: Relació de transformacions homogènies del robot passiu.

El planificador local adoptat té un comportament determinista. Donades dues configuracions c i n, el primer que fem és calcular l'equació vectorial de la recta que passa pels punts definits com: el vector d'angles del robot actiu en c, que denotem per $q_c^a = \vec{\theta}_c^a$ i el vector d'angles del robot actiu en n, denotat per $q_c^a = \vec{\theta}_n^a$.

$$\vec{\theta}_c^a = \begin{pmatrix} \theta_{c1}^a \\ \vdots \\ \theta_{c6}^a \end{pmatrix} \qquad \vec{\theta}_n^a = \begin{pmatrix} \theta_{n1}^a \\ \vdots \\ \theta_{n6}^a \end{pmatrix}$$

$$\vec{\theta}^a = \vec{\theta}_c^a + \mu(\vec{\theta}_n^a - \vec{\theta}_c^a) \qquad \text{amb } \mu \in [0,1] \tag{5.10}$$

El següent pas consisteix en discretitzar el segment recte definit per l'equació 5.10 en m configuracions del robot actiu equiespaiades, de paràmetres

$$\mu = \frac{i\,\lambda}{\|\vec{\theta}_n^a - \vec{\theta}_c^a\|}, \qquad \text{amb} \quad i = 1, \cdots, m \quad \text{sent} \quad m = \lfloor \frac{\|\vec{\theta}_n^a - \vec{\theta}_c^a\|}{\lambda} \rfloor,$$

essent λ un pas d'avanç especificat per l'usuari, que és la distància Euclidia entre dues configuracions q_i^a, q_{i+1}^a. Un cop discretitzat el segment, per a cada configuració q_i^a del robot actiu, mirem si podem tancar la cadena amb el robot passiu, i si és així comprovem si aquesta configuració es troba lliure de col·lisions. Si aquest procediment el repetim per totes les configuracions q_i^a del segment, i totes les configuracions q_i es troben lliures de col·lisió, aleshores el planificador local conclou que ha pogut connectar els nodes c i n. La figura 5.9 mostra que el segment recte en l'espai de configuracions del robot actiu es transforma en una corba de configuracions $q_i = (q_i^a, q_i^p)$ en l'espai de configuracions de la cadena tancada.

5.4.2.3 La funció distància

La funció distància intervé en la selecció dels nodes veïns al nou node c inserit (secció 4.1.1). Definirem la distància entre dues configuracions a i b com la longitud de la trajectòria γ de l'espai de configuracions que segueix el planificador local per unir-les:

$$d(\bar{s}, \bar{t}) = \int_{\bar{s}}^{\bar{t}} \|\nabla\gamma(\bar{t})\|\, dt = |\bar{s} - \bar{t}|, \quad \forall \bar{s}, \bar{t} \in \mathbb{R}^n. \tag{5.11}$$

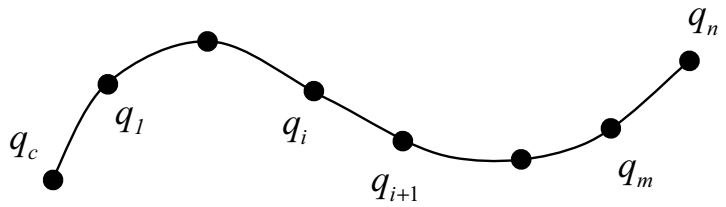


Figura 5.9: Trajectòria d'una cadena tancada en l'espai de configuracions.

Per simplificar-ne el càlcul, podem aproximar γ per la poligonal induïda pels punts de pas q_i (figura 5.10), i així la distància és

$$D(\bar{s},\bar{t}) = \sum_{i=0}^{m} \|q_i - q_{i+1}\|, \tag{5.12}$$

on

$$\|q_i - q_{i+1}\| = \sqrt{\|\bar{\theta}_a^i - \bar{\theta}_a^{i+1}\|^2 + \|\bar{\theta}_p^i - \bar{\theta}_p^{i+1}\|^2} \tag{5.13}$$

és la funció distància Euclidia en l'espai de les articulacions. Com que aquesta distància és un paràmetre que només depèn de la cinemàtica del sistema, per accelerar-ne el seu càlcul es pot simular l'execució de la trajectòria γ sense detectar la col·lisió amb cap obstacle.

5.4.3 Etapa d'expansió

En aquesta etapa s'intenta reduir el nombre de components connexos del mapa de rutes (secció 4.1.2). L'estratègia passa per seleccionar nodes que estiguin en regions difícils i expansionar-los tot seguit, generant una trajectòria aleatòria a partir d'ells i verificant si la connexió amb altres components és possible.

Sigui c el node a expandir i n el node amb el qui potencialment s'estableix connexió. Cal dir que, contràriament a com s'explica en la secció 4.1.2, en el planificador que desenvoluparem no emmagatzemarem el camí de connexió com una aresta $c-n$. Interpretarem la trajectòria com una seqüència de nodes de configuracions prèvies a la col·lisió (que s'afegiran a N) unides per arestes que indiquen crides exitoses del planificador local (i que s'afegiran a E). Així aconseguirem treballar amb un graf on el contingut de les arestes és sempre el mateix. Una altra diferència és que no només intentarem connectar l'últim node n amb altres components, sinó que ho farem per cada node de la seqüència. A continuació expliquem amb detall com es poden generar desplaçaments en direccions aleatòries per la cadena cinemàtica tancada.

1. Donat un node c, en primer lloc generem aleatòriament una configuració lliure a tal com es veu a la figura 5.11.

2. Tot seguit comprovem la validesa cinemàtica de la trajectòria que connecta c amb a mitjançant el planificador local amb el detector de col·lisions desactivat. Després,

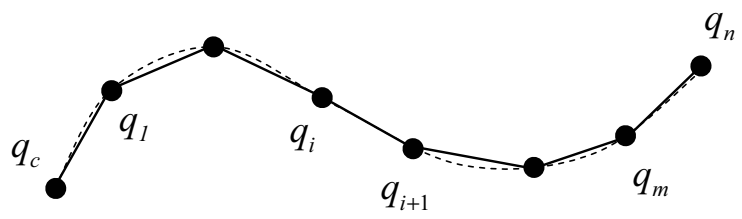


Figura 5.10: Trajectòria poligonal per aproximar la longitud de l'arc de la corba γ.

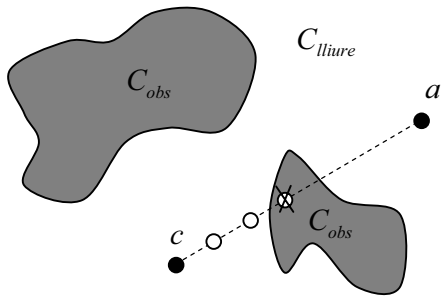


Figura 5.11: Generació d'una trajectòria aleatòria durant l'etapa d'expansió.

si la connexió és vàlida cinemàticament, cridem el planificador local entre aquestes dues configuracions tal com es fa en l'etapa de construcció.

3. El planificador que emprem és el de la secció 5.4.2.2 amb l'afegit que a cada pas d'avanç cal memoritzar la configuració del pas anterior. D'aquesta manera quan el planificador falli a la iteració i (amb $i > 1$), la configuració q_{i-1} esdevindrà lliure i constituirà el primer node de la trajectòria aleatòria. Després d'aquesta es procedeix de la mateixa forma fins que la trajectòria tingui un màxim de nodes o es connecti a un altre component connex.

Com veiem, procedint com s'esmenta, en lloc de cridar directament el planificador local entre c i a, es valida primer cinemàticament la connexió entre aquests dos nodes. D'aquesta maneta assegurem que si el planificador local falla es perquè ha trobat una configuració lliure propera a la paret, d'altra banda això no es podria assegurar i el cost computacional que suposa cada crida del planificador faria alentir el procés d'expansió.

Cal afegir que la distància entre els nodes de la trajectòria aleatòria no estarà per sota de la cota màxima, ja que empíricament hem comprovat que les trajectòries aleatòries calculades sense aquesta restricció tenen una probabilitat de connexió amb altres components més alta.

5.4.4 Fase d'Interrogació

Com esmentàvem a la secció 4.2 hi ha bàsicament dues possibilitats a l'hora de connectar les configuracions d'inici i final al mapa de rutes: o bé es connecten directament al mapa mitjançant el planificador local, o bé es genera explícitament una trajectòria aleatòria que parteixi de la configuració considerada per intentar la connexió amb el mapa. Si la configuració es troba en zones d'alta dificultat, utilitzarem una trajectòria aleatòria generada com a l'etapa d'expansió (secció 5.4.3). Ara bé si la configuració que hem de connectar al mapa es troba allunyada dels obstacles, farem una trajectòria aleatòria com la d'expansió però limitant la distància entre nodes adjacents a la cota màxima.

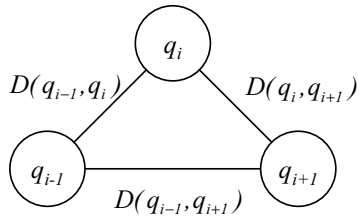


Figura 5.12: Triangle considerat en eliminar el node i.

5.4.5 Fase de Suavització

Si la interrogació entre dues configuracions és exitosa, com a resultat obtenim una seqüència de nodes que codifiquen la trajectòria a seguir entre la configuració inicial i final. Explícitament, aquesta trajectòria es pot trobar com la concatenació de cada una de les trajectòries entre nodes adjacents, calculades novament mitjançant el planificador local.

Com que s'ha usat un mètode heurístic, aquesta trajectòria molt probablement no serà òptima quant a distància recorreguda. De fet, com a conseqüència de mantenir un mapa de rutes que no presenta cicles, aquesta trajectòria serà sovint força llarga. Per escurçar la seva longitud hem implementat dos mètodes de suavitzat diferents que es poden aplicar de forma conjunta o combinada. La secció 5.4.5.3 discuteix els avantatges i inconvenients de cadascun d'ells.

5.4.5.1 L'optimitzador de la poligonal

Aquest mètode tracta la trajectòria obtinguda com una poligonal que es vol escurçar [3]. Malgrat que està pensat per optimitzar la trajectòria dels robots manipuladors treballant independentment, es pot adaptar fàcilment al nostre cas. La idea de l'algorisme és anar eliminant nodes de manera que es redueixi la longitud de la trajectòria inicial, procedint així:

1. En primer lloc es calcula la reducció de distància que suposa l'eliminació del node i de la trajectòria inicial. Per calcular-la, prèviament s'han de trobar les distàncies del triangle format pels nodes $i-1$, i, i $i+1$, com es veu a la figura 5.12. La reducció de distància que suposaria l'eliminació del node i és:

$$R_i(q_{ini}, q_{fi}) = D(q_{i-1}, q_i) + D(q_i, q_{i+1}) - D(q_{i-1}, q_{i+1}) \tag{5.14}$$

   on per calcular-la cal que $D(q_{i-1}, q_{i+1})$ sigui un camí lliure.

2. L'ordre en que esborrem els nodes influirà en la solució que obtindrem. L'algorisme proposa que la millor forma d'escurçar la trajectòria és eliminar primer els nodes de major R_i. Per tant calcularem els R_i i els ordenarem decreixentment.

3. Quan un node i és seleccionat com a candidat a ser eliminat, cal cridar el planificador local entre els nodes $i-1$ i i per veure si la trajectòria que els connecta és o no

5.4.5.3 Discussió

Comparant els dos mètodes de suavització, sens dubte el que dóna la trajectòria més escurçada és el segon, pel fet que fa una cerca exhaustiva entre totes les possibilitats que ofereix la trajectòria de partida. Ara bé, quan el resultat de la interrogació sigui un camí molt llarg, al voltant de més de 100 nodes, el temps de còmput de l'algorisme de camins mínims és força gran, impedint l'obtenció del resultat suavitzat de forma ràpida. En aquest cas, és convenient aplicar el primer mètode fins a obtenir un camí amb un nombre raonable de nodes, per després aplicar sobre aquest l'algorisme de camins mínims.

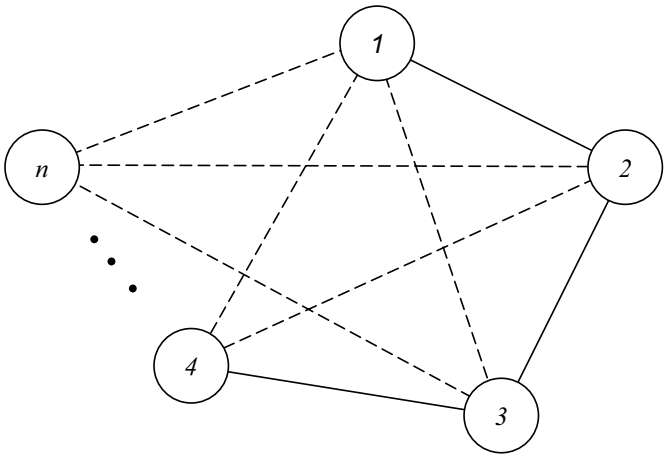


Figura 5.13: Graf construït per aplicar l'algorisme de Dijkstra.

lliure. Si ho és s'elimina el node. Altrament, es tria el següent node candidat a ser eliminat. Notem que cada vegada que eliminem un node i haurem de recalcular novament les reduccions de distància associades als nodes $i-1$ i $i+1$, i reordenar-les convenientment.

5.4.5.2 Obtenció del camí mínim

Aquesta tècnica de suavització fa una cerca exhaustiva, trobant el camí mínim restringit a passar pels nodes de la trajectòria no suavitzada considerada, procedint de la següent manera:

1. En primer lloc, donada una trajectòria solució involucrant n nodes, es construeix el graf T complet d'aquests nodes, que conté $n \cdot (n-1)/2$ arestes, tal com es mostra a la figura 5.13. Emmagatzemem a les arestes de T les distàncies entre els nodes que connecten i eliminem una aresta si la trajectòria directa entre els dos nodes no és possible. Cada aresta pot a més ser marcada en saber-se si els dos nodes que connecta poden ser units pel planificador local, limitant així les crides a aquest.

2. Un cop construït el graf T, es pot aplicar l'algorisme de Dijkstra [34, pàg 225-227] per trobar el camí mínim que uneix la configuració inicial amb la final, circulant a través d'aquest graf.

3. Calculat el camí mínim representem la trajectòria d'aquest per comprovar si és o no lliure de col·lisions. Si el resultat és positiu el procés haurà acabat, d'altra banda haurem d'eliminar del graf l'aresta on el planificador ha fracassat, i marcar les que corresponen a trams lliures, iniciant una nova cerca sobre el graf i iterant el procés fins a trobar una trajectòria lliure.

Capítol 6

Resultats experimentals

6.1 Modelització dels elements

Al llarg dels capítols 4 i 5 s'han descrit les estratègies per desenvolupar un planificador basat en mètodes probabilístics, però s'han omès aspectes secundaris com són: la modelització dels elements de l'espai de treball (robots, objecte i obstacles) a un programa de simulació i la dotació de moviments als que ho requereixin. Aquesta modelització s'ha fet mitjançant el software *Webots* [13], que a més d'incorporar un detector de col·lisions té rutines en c per interactuar amb els models. A continuació es donen unes breus explicacions per construir models mitjançant aquest software.

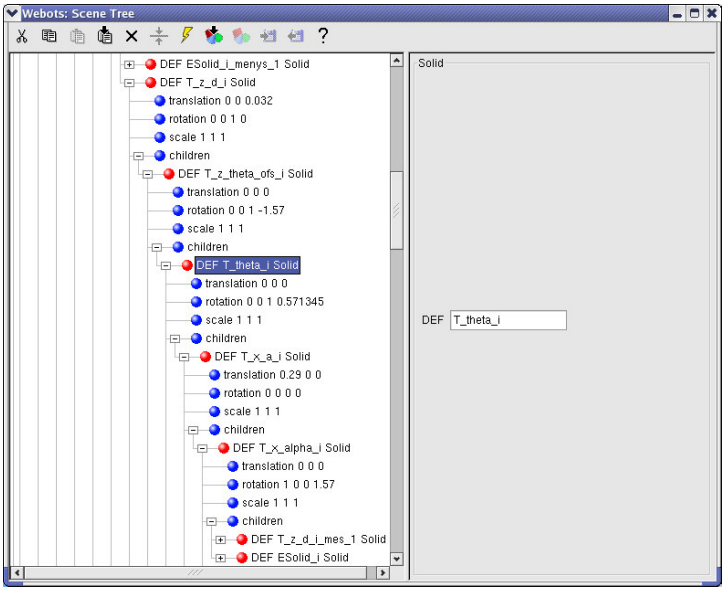
6.1.1 Webots

La modelització dels elements en Webots es fa amb tres etapes. La primera, externa a Webots, consisteix en convertir els fitxers de cada model en el format VRML 2.0, la segona seria la importació d'aquests fitxers a Webots i finalment la tercera establir les posicions i orientacions d'aquests models mitjançant una finestra anomenada arbre d'escena (*scene tree*) [13, pàg 32]. Aquesta finestra, basada en la programació VRML (Virtual Reality Modeling Language) [36], té la denominació d'arbre perquè de la modelització dels elements se'n obté un conjunt de nodes relacionats jeràrquicament. Els que presenten més interès són els següents:

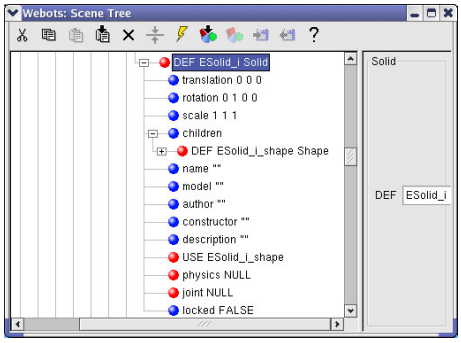
- Nodes de forma (*Shape*): Contenen totes les característiques geomètriques dels elements a representar així com altres atributs com el color, la brillantor, etc.

- Nodes de transformació (*Transform*): Permeten definir una translació i orientació respecte els nodes inserits en el seu camp fill (*children*).Un node que hereta les propietats d'aquest és el node *Solid* que conserva les propietats del*Transform* però a més pot detectar les col·lisions amb altres nodes *Solid* si es defineix la geometria del sòlid que s'està transformant en el seu camp *Bounding Box* (figura (b) de 6.1).

- Nodes d'agrupació (*Group*): Permeten inserir diferents nodes en el seu camp fill (*children*).



(a)



(b)

Figura 6.1: (a) Establiment de la transformació homogènia entre sòlids adjacents mitjançant nodes *Transform*. És d'interès notar que el node ombrejat representa l'articulació i. (b) Estructura del sòlid i.

Pel projecte, els robots RX60 són els elements que presenten més complexitat en quan a la modelització ja que constitueixen una cadena cinemàtica de sòlids. Per establir les posicions i orientacions entre els sòlids adjacents s'empra la transformació homogènia que els lliga, calculada com el producte de les transformacions associades als paràmetres de Denavit-Hartenberg (equació B.1). Aquestes transformacions parcials són les que afegim a l'arbre d'escena com a nodes Sòlid consecutius (figura (a) de 6.1). Cal notar que aquesta estructura jeràrquica a base d'afegir fills a cadascun dels nodes és ideal per a la representació d'aquestes cadenes cinemàtiques pel fet que, quan es mou una articulació, l'efecte es propaga a tots els sòlids que es troben a continuació d'aquesta.

6.1.2 Passos per modelitzar els robots RX60

Breument descrivim els punts més rellevants:

1. Obtenció del model global en CAD del RX60 subministrat per l'empresa Stäubli.

2. Conversió dels models de cada un dels sòlids del RX60 a VRML 2.0 mitjançant el software *Sòlid Works*. Els models convertits són nodes *Solid* amb el camp *children* format per múltiples nodes *Shape* que constitueixen diferents parts del model.

3. Tractament dels fitxers dels models convertits que consisteix en agrupar els nodes *Shape* de cada *Solid* en un de sol. D'aquesta manera podem fer un bon ús del detector de col·lisions de Webots.

4. Tractament dels fitxers dels models que consisteix en transformar les coordenades absolutes de cada model a relatives a la seva referència de Denavit-Hartenberg. Així podrem inserir els models seguint el patró de la figura 6.1.

5. Construcció de l'estructura jeràrquica de nodes en Webots seguint el patró de la figura 6.1.

6.2 Introducció dels experiments

A continuació procedirem a resoldre dos exemples de dificultat creixent amb el planificador desenvolupat.

El primer que s'ha de fer és seleccionar els paràmetres que intervenen en l'algorisme. Després de realitzar molts experiments hem vist que per la majoria de problemes els següents paràmetres,

num. veins: 30.

max. dist: 3.3,

aconsegueixen obtenir mapes força representatius de l'espai lliure amb un nombre no molt elevat de nodes generats. Així doncs, aquests seran els que triarem pels dos experiments. També afegir que seleccionarem com a robot actiu l'esquerra, tan en la generació de configuracions aleatòries com en les crides al planificador local, on per aquest últim a més triarem un valor de *pas_cami_local*: 0.045 que és prou petit com per assegurar la validesa de les connexions. Respecte als paràmetres associats al RX60 els rangs de treball de les articulacions seran els definits per l'empresa Stäubli (taula C.2), i el tipus de solució cinemàtica del robot passiu serà triat per a cada problema com veurem.

Cal de dir que tots els problemes tindran per defecte els obstacles terra, i una gàbia de barres prismàtiques tal com es presenta en la cel·la robotitzada de l'institut IRI. Així doncs els nous obstacles que introduirem per cada exemple seran referits com obstacles interiors (secció E.4.1).

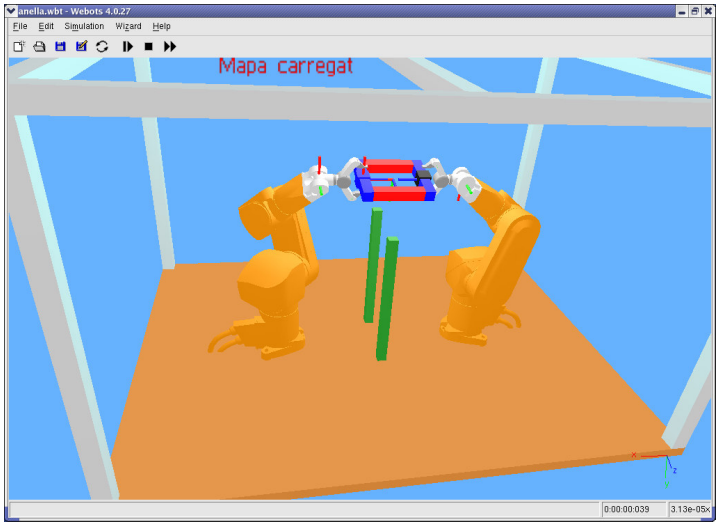


Figura 6.2: Elements de l'espai de treball per l'experiment 1.

6.3 Experiment 1

6.3.1 Introducció

En aquest experiment l'espai de treball estar format pels següents elements (figura 6.2):

- Els robots RX60 separats una distància de 900 mm.
- L'objecte a transportar que és una anella rectangular de colors blau i vermell.
- Els obstacles interiors: Dues barres verticals de color verd separades una distància de 300 mm.

Objectiu: L'objectiu és trobar trajectòries que portin l'anella de l'interior de la barra esquerra (configuracions d'inici) a la dreta (configuracions de fi).

6.3.2 Resultats de l'aprenentatge

Abans de generar el mapa de rutes hem d'escollir quina solució cinemàtica desitgem pel robot passiu en la generació de les configuracions aleatòries (taula C.4). Si generem un mapa sense obstacles podem veure que amb la solució *run* l'espai cinemàtic es troba dins la regió on volem trobar les trajectòries solució (figura 6.3). A continuació realitzarem les diferents etapes de la fase d'aprenentatge per aquest experiment.

1. Etapa de construcció

n. nodes generats: 15000.

n. components connexos: 78.

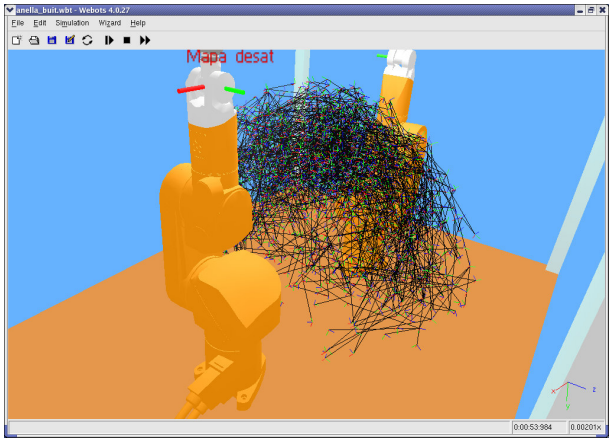


Figura 6.3: Cada un dels sistemes de referència corresponen als de l'objecte quan es troba formant cadena tancada amb els robots RX60, amb el passiu amb configuració *run*. Les arestes són connexions entre parelles de configuracions.

components connexos									
$\mathbf{c_1}$		$\mathbf{c_2}$		$\mathbf{c_3}$		$\mathbf{c_4}$		$\mathbf{c_{i\ i=5,\cdots,78}}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
6779	45,19	6735	44,90	742	4,95	640	4,27	< 6	$< 0,04$

Taula 6.1: Distribució dels components connexos del mapa finalitzada la construcció.

temps d'execució: $656,68$ s.

interrogacions: Fracassen.

Podem veure que generant 15000 nodes l'etapa de construcció ha donat un resultat força positiu, ja que dels 78 components realment 4 podem considerar-los importants com s'aprecia a la taula 6.1. Ara bé, com que les interrogacions per assolir l'objectiu fracassen haurem de realitzar l'etapa d'expansió per intentar disminuir el nombre de components connexos.

2. Etapa d'expansió

n. nodes expansionats: 300.

n. components connexos: 19.

temps d'execució: $333,89$ s.

interrogacions: Amb èxit.

Clarament l'etapa d'expansió ha estat molt eficaç, ja que hem aconseguit unir els quatre components connexos principals en un de sol (taula 6.2).

Sumant els temps emprats per realitzar l'etapa de construcció i d'expansió tenim que la fase d'aprenentatge ha tardat $990,57$ segons. Tot i que aquest temps no és molt gran, el seu anàlisi no és d'interès, ja que l'objectiu és trobar un mapa representatiu de l'espai lliure i que les interrogacions sobre aquest es realitzin de forma gairebé instantània.

components connexos									
$\mathbf{c_1}$		$\mathbf{c_2}$		$\mathbf{c_3}$		$\mathbf{c_4}$		$\mathbf{c_{i\ i=5,\cdots,19}}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
17701	99,13	16	0,09	12	0,067	12	0,067	< 8	$< 0,045$

Taula 6.2: Distribució dels components connexos del mapa finalitzada l'expansió.

6.3.3 Resultats de les interrogacions

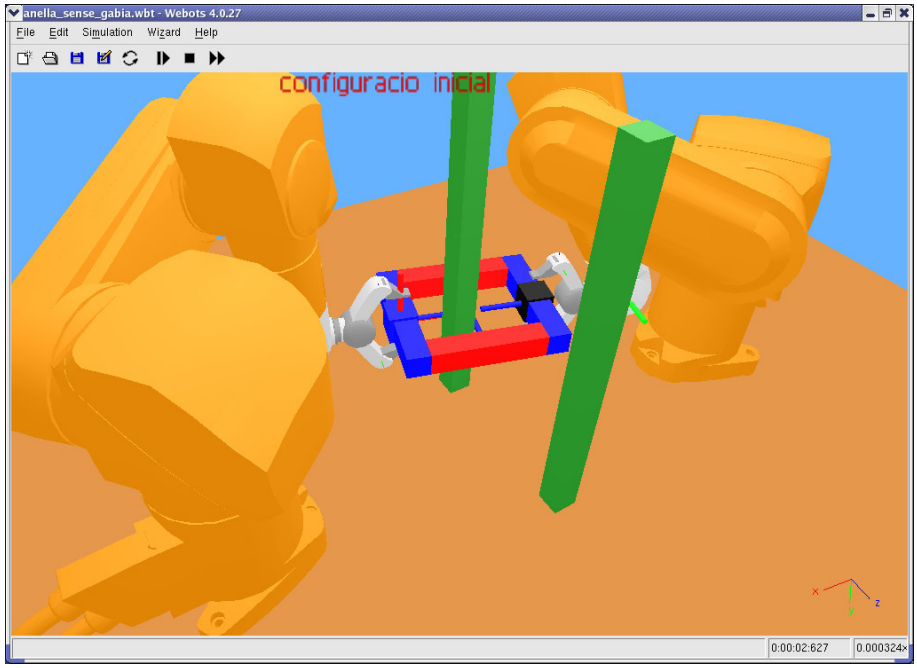
interrog.	*connexió*		*suavitzat*		
	temps [s]	dist. traject. [º]	temps [s]	dist. traject. [º]	% red. dist
1	3,23	82,21	0,99	43	47,69
2	0,23	24,65	0,22	16,23	34,16

Taula 6.3: Resultats de dues interrogacions sobre el mapa obtingut. Les configuracions d'inici i fi per a cada una són quan l'anella es troba dins la barra esquerra i dreta respectivament.

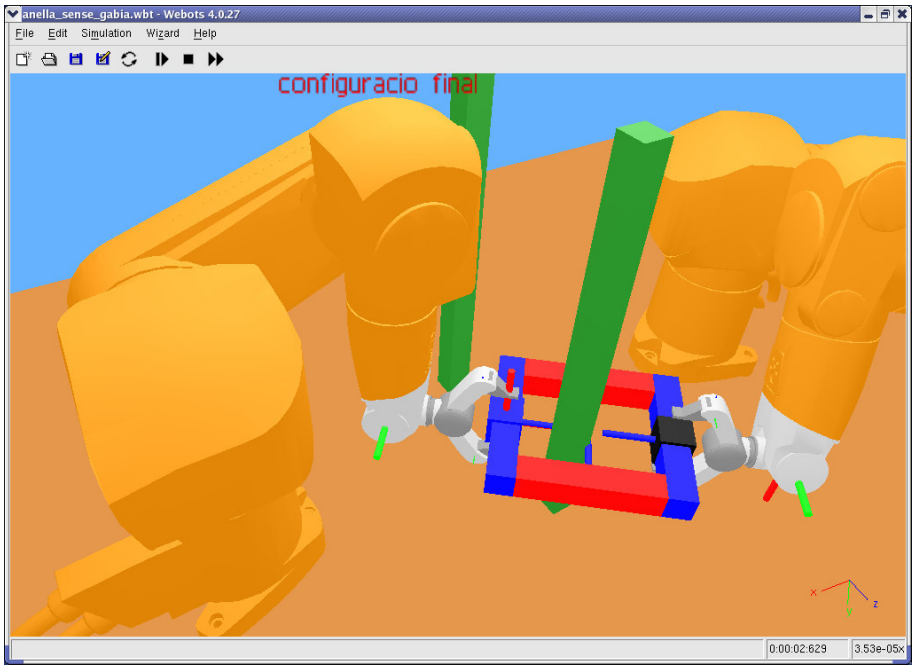
Els resultats obtinguts són prou engrescadors (taula 6.3), ja que les connexions s'han realitzat molt ràpidament (uns segons). A més, el suavitzador emprat, basat en l'obtenció del camí mínim (secció 5.4.5.2) ha estat capaç de reduir més d'un terç les distàncies de les trajectòries inicials. A continuació visualitzarem la seqüència d'imatges de la trajectòria suavitzada de la primera interrogació.

6.3.4 Una trajectòria solució

Aquest secció pretén mostrar la trajectòria solució que té com a configuració inicial i final les (a) i (b) de la figura 6.4. Cal dir que en les diferents seqüències d'imatges de la trajectòria s'han canviat els punts de vista, per tal que el lector comprovi que en tot moment la cadena tancada es troba lliure de col·lisions.

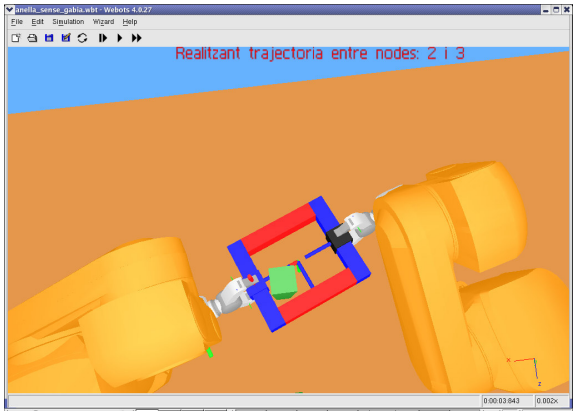


(a)

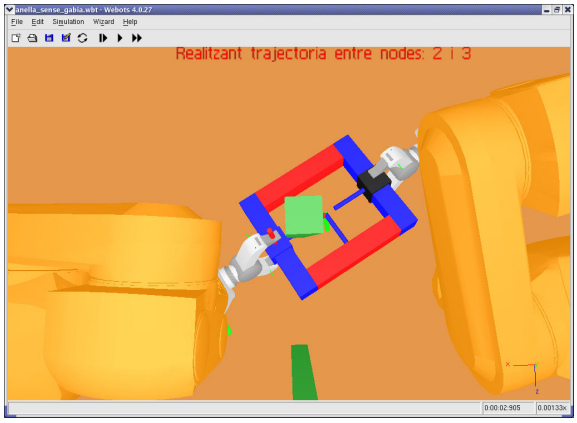


(b)

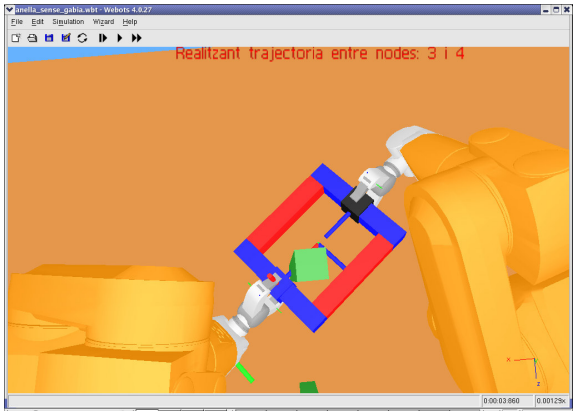
Figura 6.4: (a) Configuració inicial. (b) Configuració final.


Realitzant trajectoria entre nodes: 2 i 3

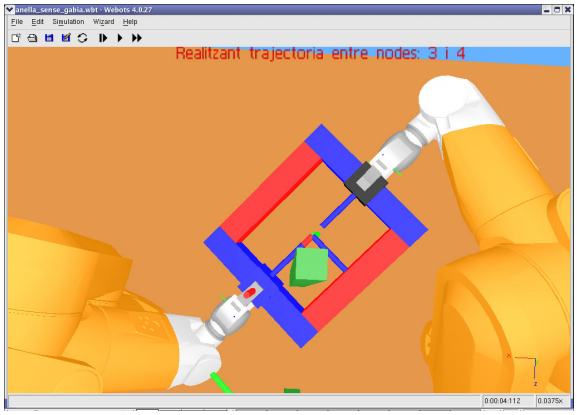
(a)


Realitzant trajectoria entre nodes: 2 i 3

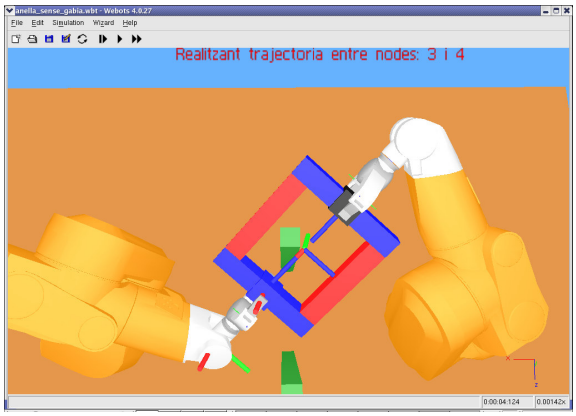
(b)


Realitzant trajectoria entre nodes: 3 i 4

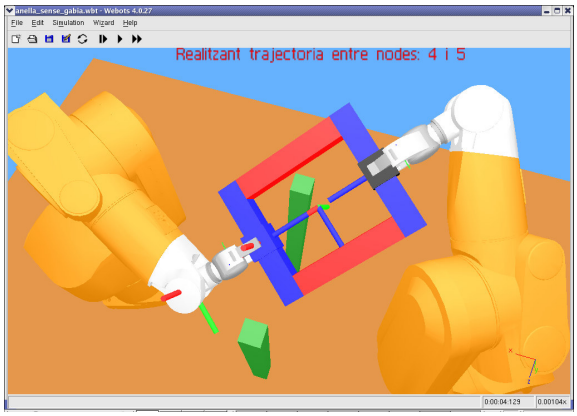
(c)


Realitzant trajectoria entre nodes: 3 i 4

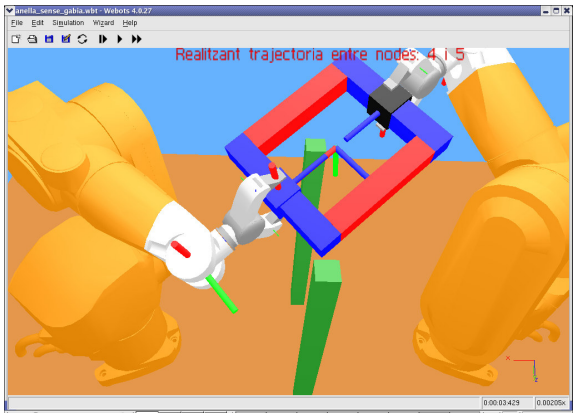
(d)


Realitzant trajectoria entre nodes: 3 i 4

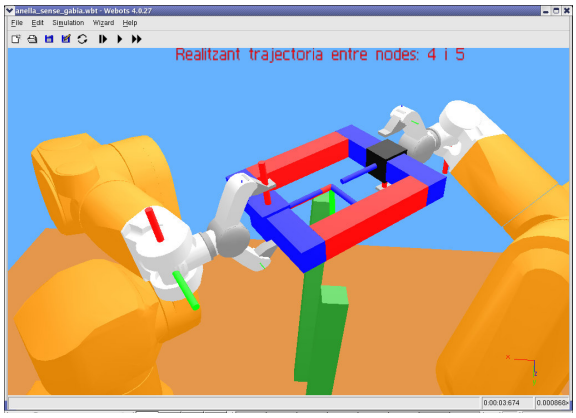
(e)


Realitzant trajectoria entre nodes: 4 i 5

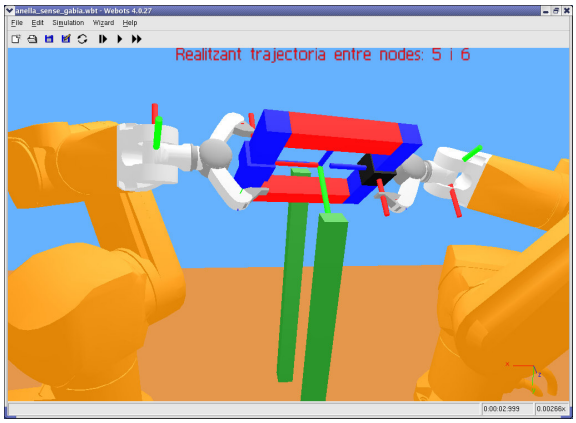
(f)



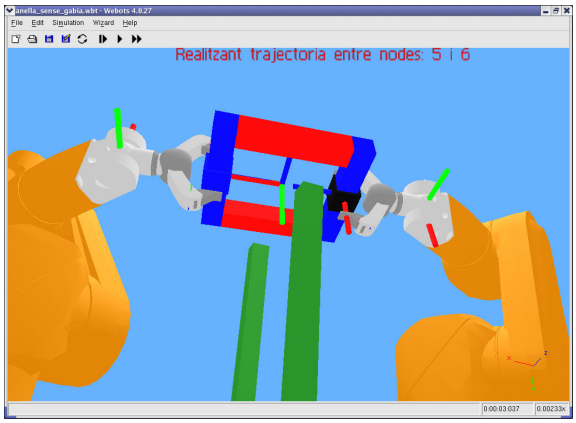
(g)



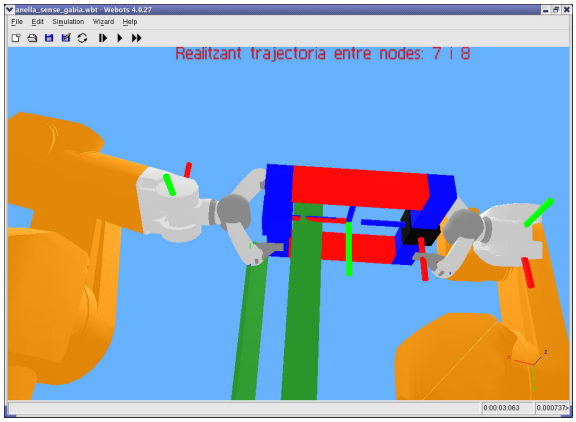
(h)



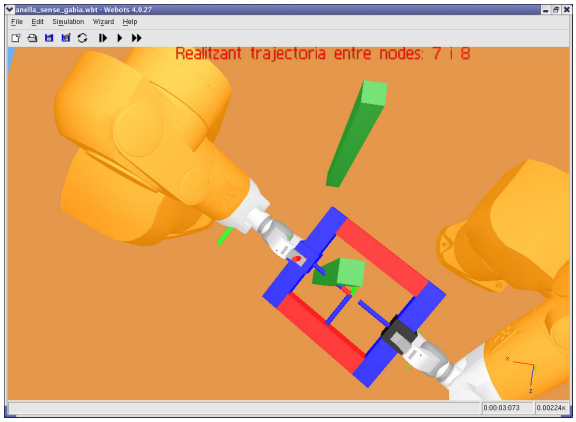
(i)



(j)



(k)



(l)

Figura 6.5: Seqüència d'imatges de la trajectòria solució.

6.4 Experiment 2

6.4.1 Introducció

Aquest experiment és més complex que l'anterior, i com veurem la seva resolució no ha estat tant sistemàtica. Tanmateix, hem aconseguit l'objectiu principal, trobar la trajectòria entre dues configuracions d'inici i fi (secció 6.4.4), i per tant és una bona prova que el planificador desenvolupat pot tenir cabuda en el sector industrial.
En aquest cas l'espai de treball estar format pels següents elements (figura 6.6):

- Els robots RX60 separats una distància de 1200 mm.
- L'objecte a transportar que és una barra prismàtica retorçada de colors blau i vermell.
- Els obstacles interiors: Dues barres verticals de color verd posicionades a les coordenades del pla $(90, -280)$ i $(-620, -450)$ en mil·límetres, sent x l'eix que va del robot esquerra al dret.

Objectiu: L'objectiu d'aquest experiment és trobar una trajectòria que connecti una configuració inicial i final com les (a) i (b) de la figura 6.9.

6.4.2 Resultats de l'aprenentatge

Abans de generar el mapa hem de determinar quin és l'espai cinemàtic de la cadena tancada que millor s'ajusta a la solució del problema. Si prenem la solució del tipus *run*

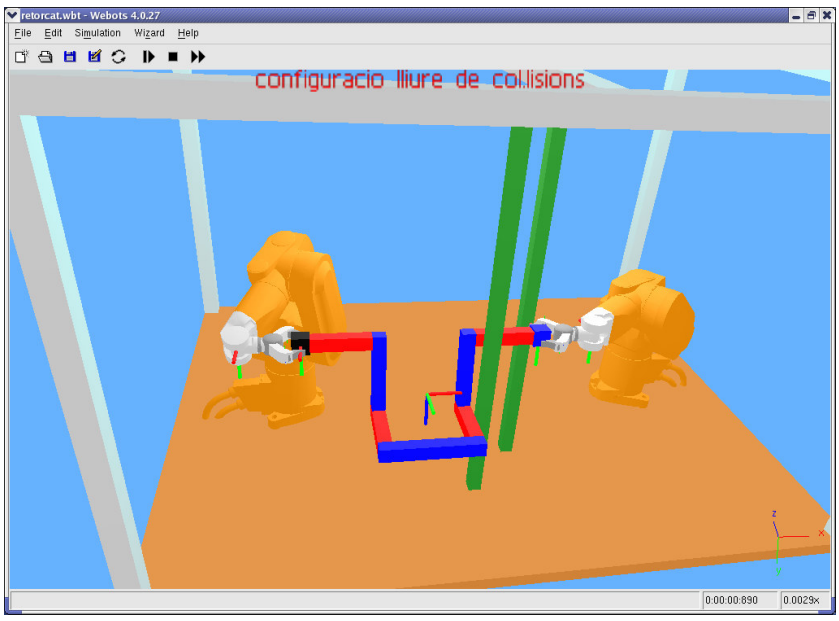


Figura 6.6: Elements de l'espai de treball per l'experiment 2.

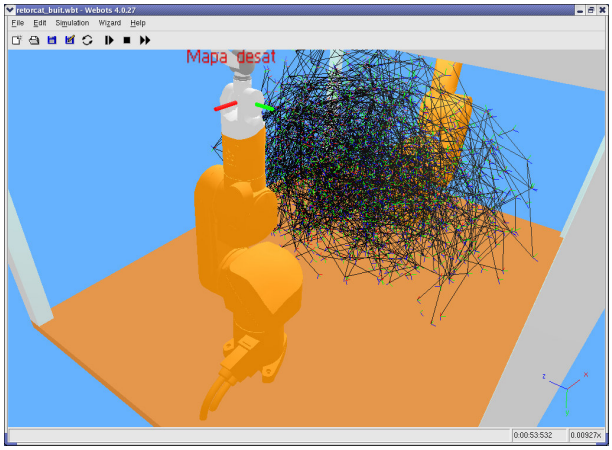


Figura 6.7: Aproximació de l'espai cinemàtic per la solució *run* del robot esquerra. Cada un dels sistemes de referència corresponen als de l'objecte quan es troba formant cadena tancada amb els robots RX60. Les arestes són connexions entre parelles de configuracions.

components connexos									
$\mathbf{c_1}$		$\mathbf{c_2}$		$\mathbf{c_3}$		$\mathbf{c_4}$		$\mathbf{c_5}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
5340	48,54	2465	22,41	395	3,60	318	2,90	287	2,61

components connexos									
$\mathbf{c_6}$		$\mathbf{c_7}$		$\mathbf{c_8}$		$\mathbf{c_9}$		$\mathbf{c_{i\ i=10,\cdots,204}}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
219	2,00	172	1,56	155	1,41	140	1,27	< 126	1,14

Taula 6.4: Distribució dels components connexos del mapa finalitzada la construcció.

(veure C.4) pel robot passiu comprovem que el mapa sense obstacles dóna configuracions a la regió on estan situades les barres, però també a la regió esquerra que no és d'interès. Per evitar aquest problema hem afegit dos obstacles rectangulars (figura 6.8) que eviten que es generin nodes en aquestes regions.

1. Etapa de construcció

n. nodes generats: 11000.

n. components connexos: 204.

temps d'execució: $598,95$ s.

interrogacions: Fracassen.

Veiem que generant 11000 nodes s'han obtingut 204 components connexos, on només 9 dels quals poden considerar-se principals (taula 6.4). Tanmateix, d'aquests no tots

són útils per trobar les trajectòries solució que busquem. Mitjançant la finestra d'anàlisi (secció E.3.4) podem veure una configuració representativa de cada un d'ells, i així saber quin d'ells hem d'eliminar. Per exemple en la figura 6.8 mostrem una configuració representativa pels sis primers components connexos, on només el c_3 és d'interès. Fent el mateix per la resta de components trobem que els d'utilitat són: c_3, c_7 c_8; els altres són filtrats (secció E.3.4). Com a conseqüència el mapa passa a tenir 965 nodes, i com que els components són encara força petits apliquem una ampliació de l'etapa de construcció. Aquesta ampliació es farà fins que l'últim node del graf tingui índex 120000 (s'introdueix al camp *max. nodes* de la finestra de paràmetres).

2. Ampliació de l'etapa de construcció 1

n. nodes generats: 119035.

n. components connexos: 284.

temps d'execució: 10438.7 s.

interrogacions: Fracassen.

Analitzant els components connexos trobem 20, formats per més de 100 nodes, que són útils. Si procedim novament a filtrar els que no necessitem, el graf queda reduït a 11705 nodes. Aplicant les interrogacions entre les configuracions inicial i final de la figura 6.9 obtenim també un resultat negatiu. Així doncs fem una nova ampliació de la construcció fins que l'índex de l'últim node sigui 180000.

3. Ampliació de l'etapa de construcció 2

n. nodes generats: 168295.

n. components connexos: 326.

temps d'execució: $11603,01$ s.

interrogacions: Fracassen.

D'aquests 326 components només 9 els considerem útils (taula 6.5), la resta els podem eliminar obtenint un nombre final de nodes de 29707. Com que els components són suficientment grans, decidim aplicar l'etapa d'expansió, i ho fem al 30% dels nodes que tenim.

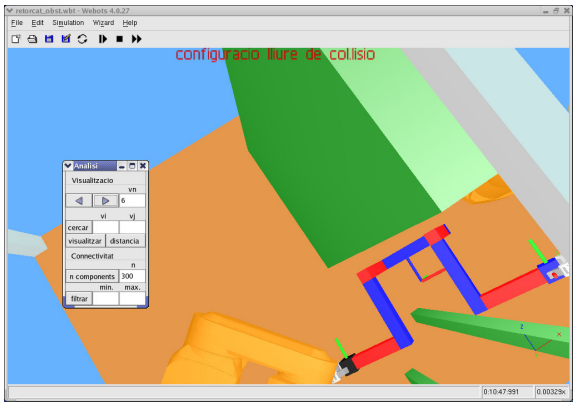
4. Etapa d'expansió

n. nodes expansionats: 8912.

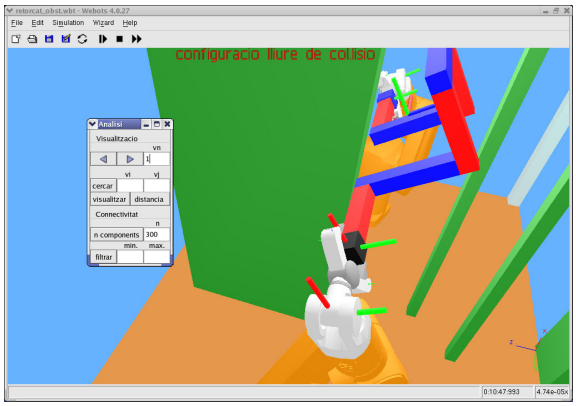
n. components connexos: 220.

temps d'execució: 10012 s.

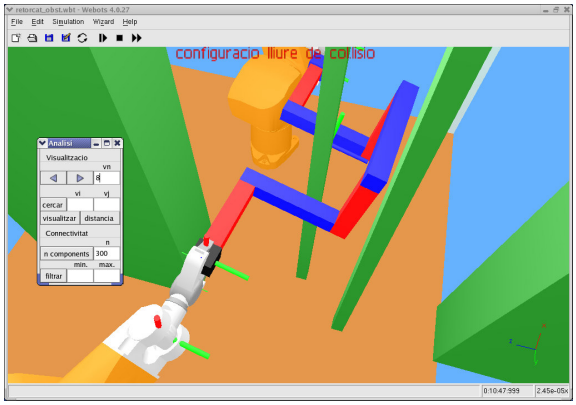
interrogacions: Amb èxit.



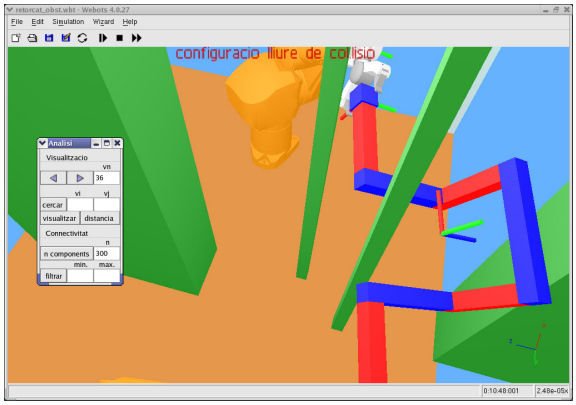
(a) node 6 de c_1



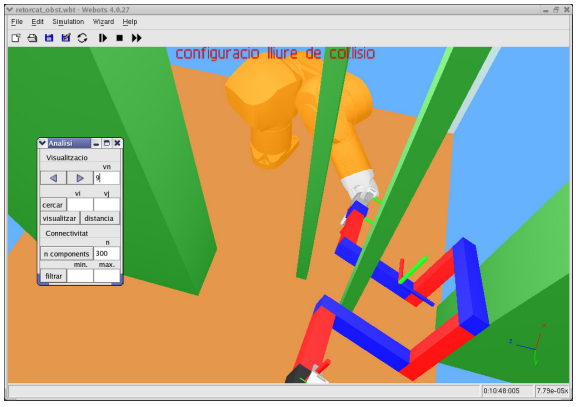
(b) node 1 de c_2



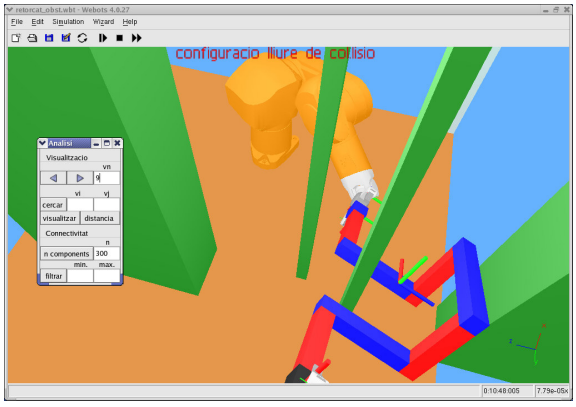
(c) node 8 de c_3



(d) node 36 de c_4



(e) node 33 de c_5



(f) node 9 de c_6

Figura 6.8: Nodes representatius dels sis primers components connexos.

components connexos									
$\mathbf{c_1}$		$\mathbf{c_2}$		$\mathbf{c_3}$		$\mathbf{c_4}$		$\mathbf{c_5}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
6335	29,70	5150	24,13	2304	10,80	2011	9,42	1572	7,36

components connexos									
$\mathbf{c_6}$		$\mathbf{c_7}$		$\mathbf{c_8}$		$\mathbf{c_9}$		$\mathbf{c_{i\ i=10,\cdots,314}}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
1029	1,55	1029	1,55	984	1,48	922	1,38	< 922	$< 4,32$

Taula 6.5: Distribució dels components connexos útils del mapa finalitzada la segona ampliació de la construcció.

D'aquests 220 components només els dos primers podem considerar-los de tamany destacable, (taula 6.6) la resta els podem descartar. Com a conclusió veiem que l'etapa d'expansió ha estat força exitosa, i tots els components no s'han reduït a un únic perquè els obstacles ho impedeixen. Tanmateix, quedant-nos amb el segon component connex podem trobar les trajectòries solució que connecten configuracions d'inici i fi del tipus de la figura 6.9. Els resultats finals amb el mapa expansionat són els següents:

num. nodes: 110157.

n. components connexos: 220.

temps execució total: $32652,66$ s.

components connexos									
$\mathbf{c_1}$		$\mathbf{c_2}$		$\mathbf{c_3}$		$\mathbf{c_4}$		$\mathbf{c_5}$	
n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.	n.	% tot.
53507	48,57	50622	45,95	4119	3,74	420	0,38	< 351	$< 0,32$

Taula 6.6: Distribució dels components connexos finalitzada l'expansió.

6.4.3 Resultats de les interrogacions

Els resultats obtinguts per diferents configuracions d'inici i fi del tipus de la figura 6.9 són els següents:

Clarament els resultats són bons, ja que com en el cas anterior obtenim les interrogacions amb pocs segons. La distància de reducció és també força bona, (un 70%), tot i que la suavització ,basada en l'algorisme de camí mínim (secció 5.4.5.2), tarda una mica més que en l'experiment anterior a causa de que en aquest cas els camins a suavitzar estan formats per molts nodes (uns 80). Una manera de reduir aquest temps, seria primer fer

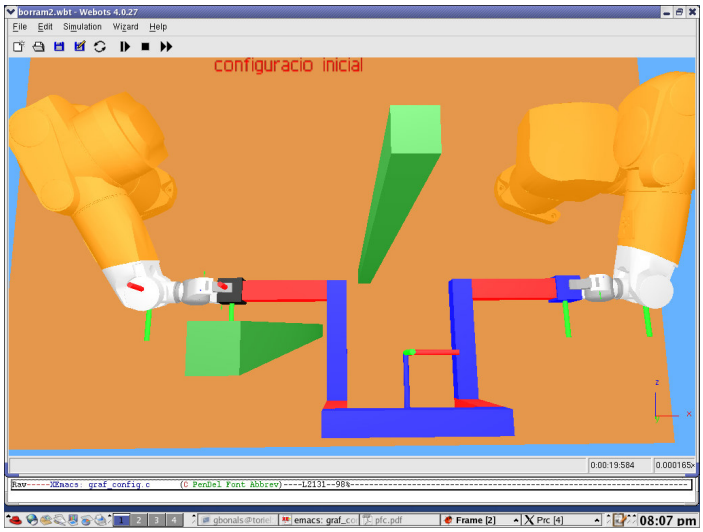
interrog.	*connexió*		*suavitzat*		
	temps [s]	dist. traject. [º]	temps [s]	dist. traject. [º]	% red. dist
1	0,31	125,37	7,16	38,88	72,88
2	0,33	127,8	6,5	38,74	69,68

Taula 6.7: Resultats de dues interrogacions sobre el mapa obtingut.

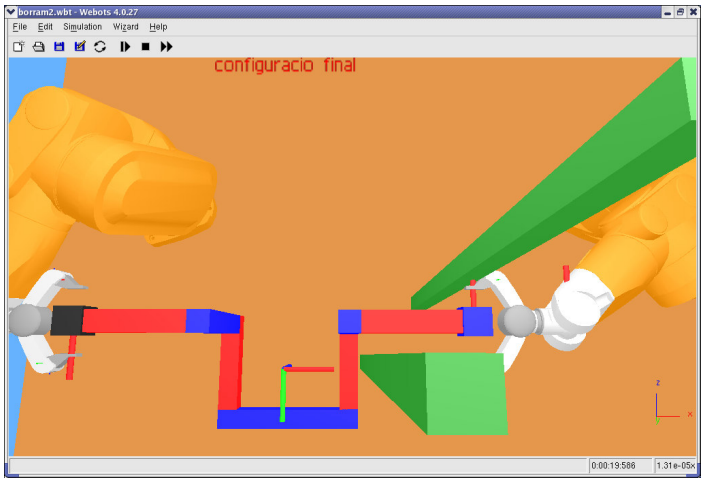
una suavització emprant l'optimitzador de la poligonal (secció 5.4.5.1), i de la trajectòria resultant aplicar l'algorisme de l'obtenció del camí mínim (secció 5.4.5.2). En aquest cas però milloraríem rapidesa de càlcul en la suavització a costa de no reduir tan la distància de la trajectòria. A continuació visualitzarem la seqüència d'imatges de la trajectòria suavitzada de la primera interrogació.

6.4.4 Una trajectòria solució

Aquest secció pretén mostrar la trajectòria solució que té com a configuració inicial i final les (a) i (b) de la figura 6.9. Tal com l'experiment 1, en les diferents seqüències d'imatges de la trajectòria s'han canviat els punts de vista, per tal que el lector comprovi que en tot moment la cadena tancada es troba lliure de col·lisions.

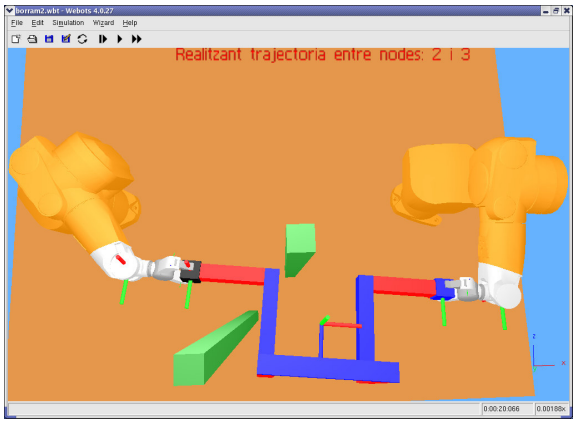


(a)

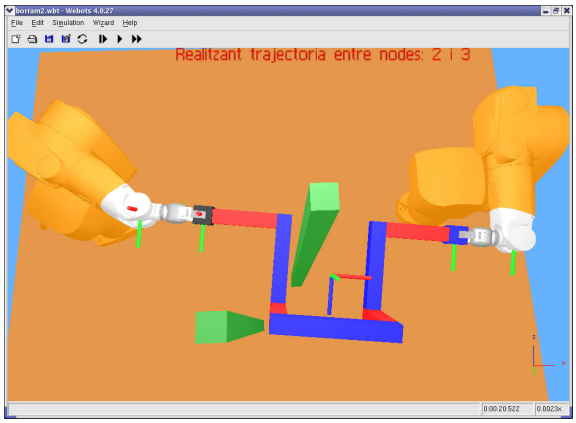


(b)

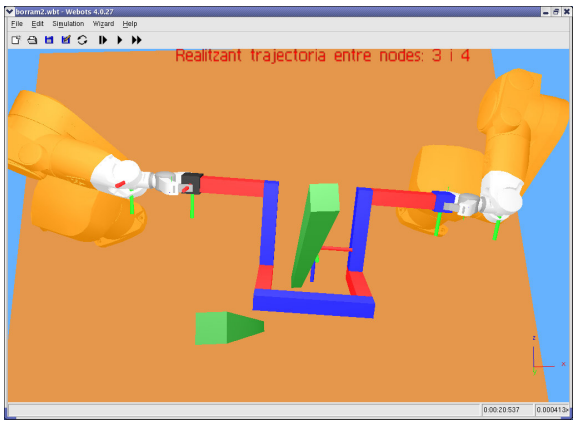
Figura 6.9: (a) Configuració inicial. (b) Configuració final.


Realitzant trajectoria entre nodes: 2 i 3

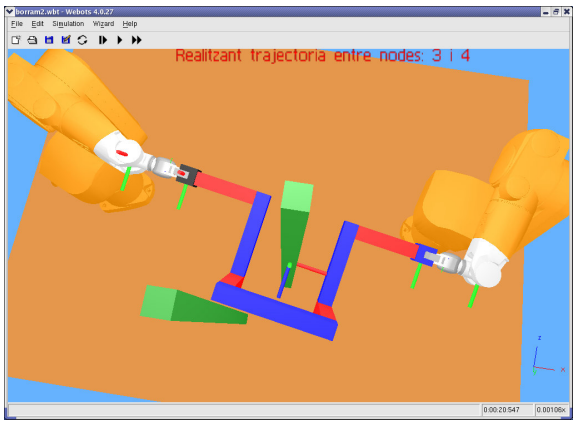
(a)


Realitzant trajectoria entre nodes: 2 i 3

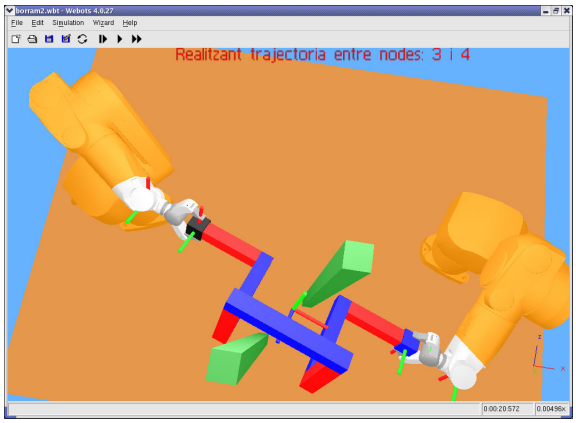
(b)


Realitzant trajectoria entre nodes: 3 i 4

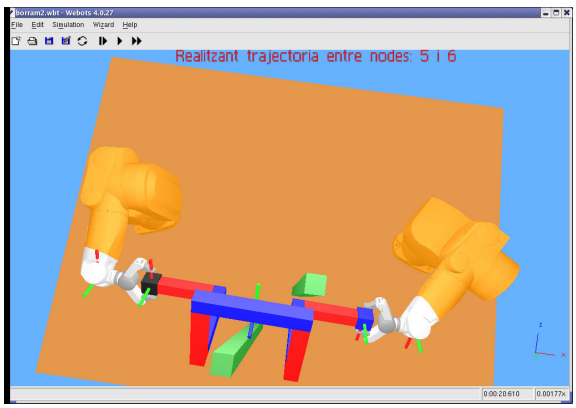
(c)


Realitzant trajectoria entre nodes: 3 i 4

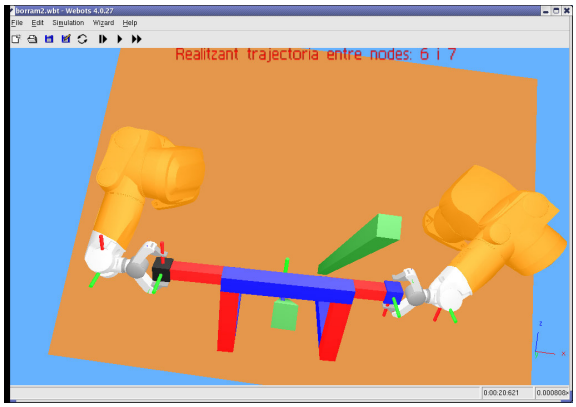
(d)


Realitzant trajectoria entre nodes: 3 i 4

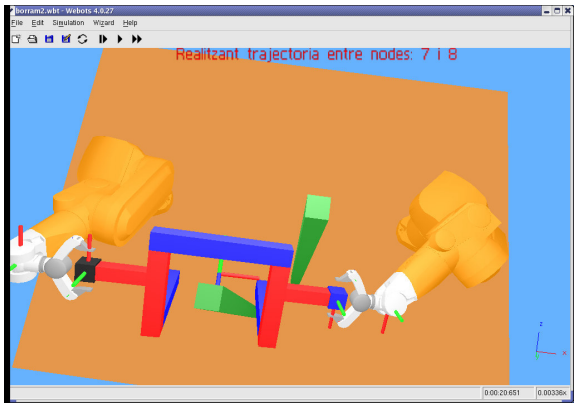
(e)


Realitzant trajectoria entre nodes: 5 i 6

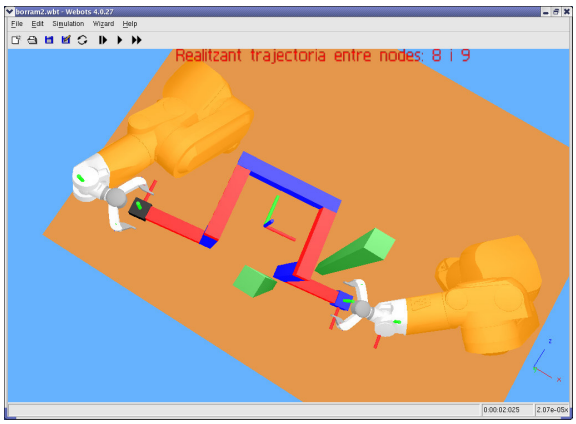
(f)


Realitzant trajectoria entre nodes: 6 i 7

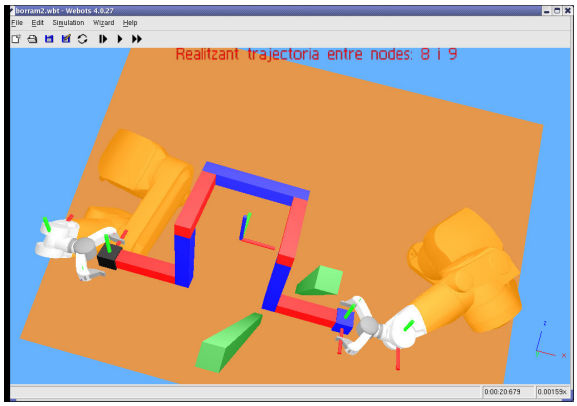
(g)


Realitzant trajectoria entre nodes: 7 i 8

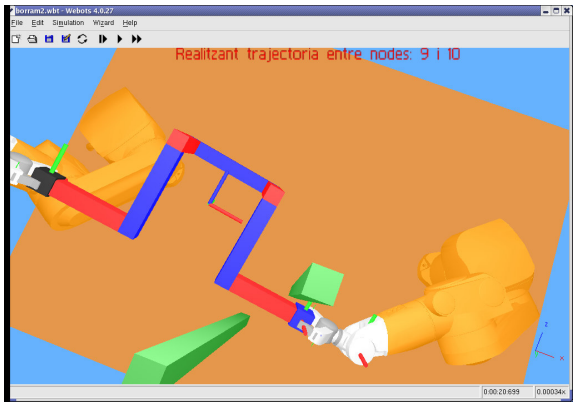
(h)


Realitzant trajectoria entre nodes: 8 i 9

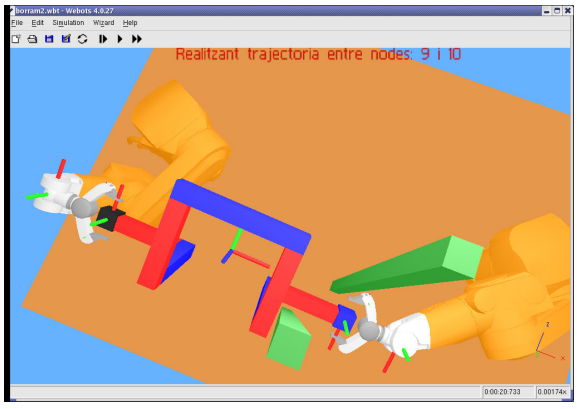
(i)


Realitzant trajectoria entre nodes: 8 i 9

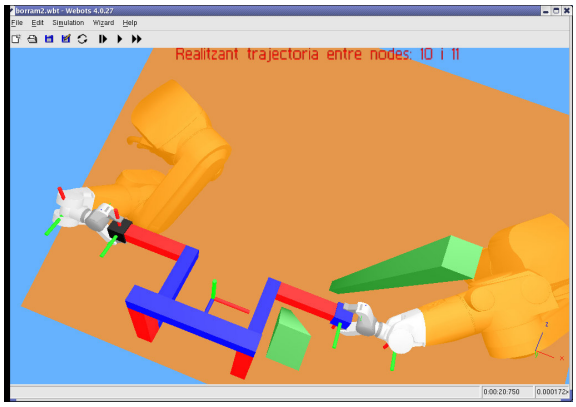
(j)


Realitzant trajectoria entre nodes: 9 i 10

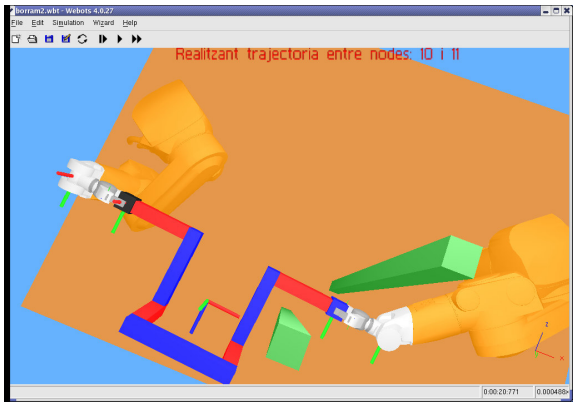
(k)


Realitzant trajectoria entre nodes: 9 i 10

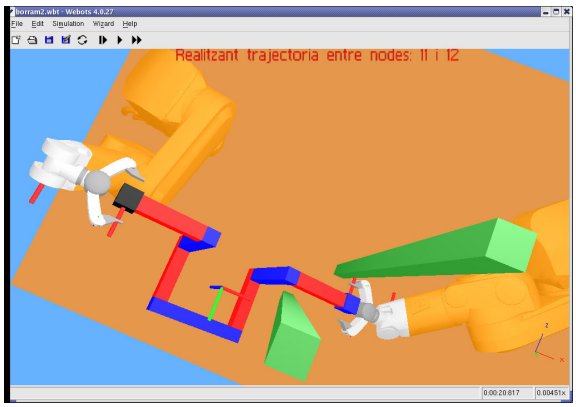
(l)



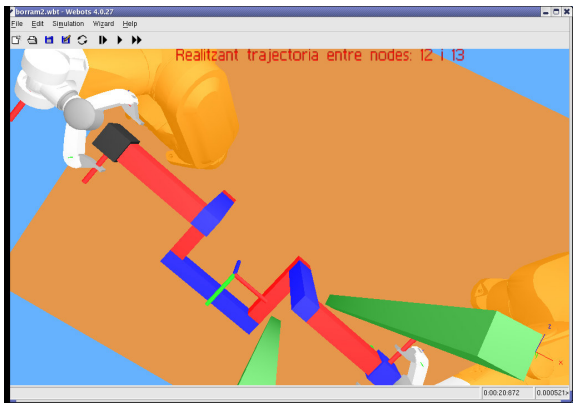
(m)



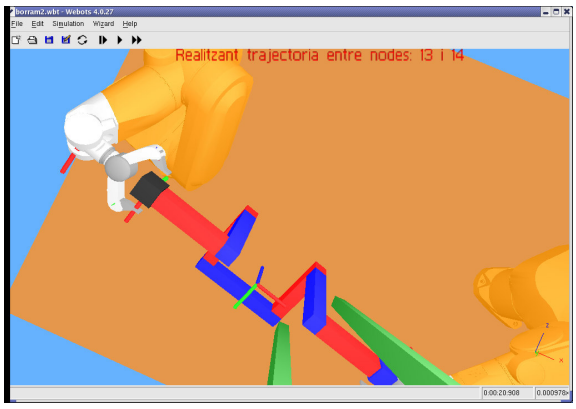
(n)



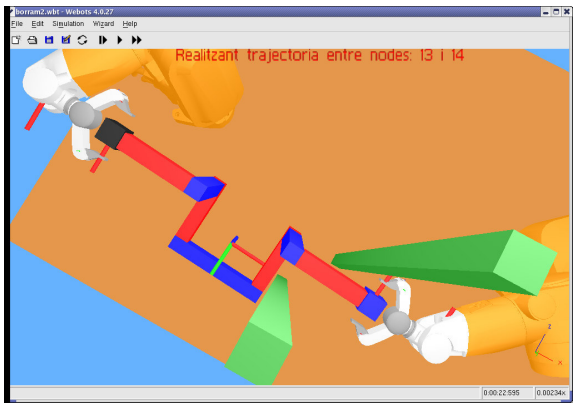
(o)



(p)



(q)



(r)

Figura 6.10: Seqüència d'imatges de la trajectòria solució.

Capítol 7

Conclusions i treball futur

Conclusions

Relacionem a continuació les principals conclusions del projecte.

a) Respecte l'assoliment dels objectius fixats

En tots els problemes analitzats l'algorisme ha estat capaç de satisfer l'objectiu marcat per aquest projecte: la generació d'una trajectòria que transporti un objecte mitjançant els dos robots des d'una configuració inicial a una final. Les solucions generades són també força curtes quant a la distància recorreguda en l'espai de configuracions, i s'obtenen de forma gairebé instantània una vegada s'ha generat el mapa de rutes de l'entorn de treball, satisfent així els requeriments inicials. S'han aconseguit resoldre dos problemes de notable complexitat que així ho demostren. Finalment, cal dir que la interfície gràfica desenvolupada té un ús prou còmode, i permet definir i resoldre problemes a un usuari que no tingui massa coneixements de planificació.

b) Respecte el mètode de planificació emprat

Tot i que el mètode de planificació utilitzat satisfà els requeriments del projecte, no podem assegurar que sigui la millor opció d'entre totes les possibles. Seria necessari implementar altres mètodes per contrastar-ne els resultats. La secció 3.4, però, ofereix una discussió de les raons que han dut a seleccionar un mètode basat en mapes probabilístics.

c Respecte al desenvolupament del projecte

El desenvolupament d'aquest projecte ens ha fet veure com és d'important el plantejament d'un problema i la seva bona definició des d'un principi. Per exemple, en algunes fases hem hagut de fer modificacions sobre resultats de fases prèvies que han suposat molta feina que no és visible pel lector. Tanmateix, donada la naturalesa del projecte, és pràcticament inevitable que això passi.

Treball futur

Per tal d'augmentar l'eficàcia del planificador, caldria treballar més en els següents punts.

a) Manteniment de les configuracions en arbres multidimensionals

Utilitzant estructures de dades com les proposades en [1], hom pot guardar les configuracions corresponents als nodes del mapa de manera que el càlcul de les configuracions més properes a una de donada sigui molt més ràpid. Aquesta millora incidiria notablement en el temps requerit per construir el mapa de rutes, tal com es demostra a [1].

a) Generació de mapes amb totes les configuracions del robot passiu

Actualment, al mapa de rutes generat només s'hi guarda la configuració corresponent a una de les solucions del problema cinemàtic invers pel robot passiu. Res impediria però guardar-les totes, obtenint així un mostrejat més complet de l'espai lliure. Per a que aquesta millora sigui d'utilitat, caldria disposar d'un planificador local que permeti connectar dues configuracions corresponents a solucions diferents d'aquesta cinemàtica inversa, o bé mostrejar directament les regions corresponents a singularitats del robot passiu i aprofitar el planificador local existent per connectar-les al mapa.

a) Utilització d'un detector de col·lisions més potent

Determinats detectors de col·lisions recentment desenvolupats [33] permeten no solament decidir si el robot està col·lisionant amb els obstacles, sinó també retornar una estimació conservativa del desplaçament màxim que es pot efectuar sense col·lisionar amb l'entorn. Això permetria implementar un planificador local de pas variable que, utilitzant aquesta informació, podria verificar si la connexió entre dos nodes propers és factible, amb un esforç de càlcul probablement molt inferior.

Capítol 8

Agraïments

Vull donar el més sincer agraïment al meu director de projecte, en Lluís Ros, per la seva incansable dedicació, i per ser capaç de transmetre'm un rigor i un mètode essencials per la realització d'aquest projecte.

A en Josep Maria Porta per la seva bona predisposició i eficiència en la resolució de problemes informàtics.

També a tots els companys del laboratori de l'Institut de Robòtica i Informàtica Industrial per la bona estada que m'han fet passar.

Finalment, he d'agrair als meus pares i amics el seu suport que m'han brindat en tot moment.

Part II

Annexos

Apèndix A

Posició i orientació d'un sòlid rígid

En l'estudi de la cinemàtica de robots manipuladors constantment necessitem conèixer la posició i orientació de cada sòlid rígid que intervé en l'espai de treball. Per identificar aquesta posició i orientació és necessari establir un sistema de referència fix al món, que denotem per A, i un altre de solidari al sòlid, que denotem per B. Si suposem que la posició de cada punt del sòlid rígid és coneguda respecte B, la seva posició i orientació queda definida per la posició de l'origen Q de B i l'orientació de B respecte d'A.

A.1 Descripció de la posició

La posició de qualsevol punt del sòlid rígid es pot descriure per un vector de tres components p, que pot ser expressat respecte d'A, o bé respecte de B. Els subíndexs x, y, i z representen la projecció de p sobre tres eixos de coordenades d'A, i anàlogament pels subíndexs u, v, i w sobre els de B,

$$ {}^{A}p = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}, \qquad {}^{B}p = \begin{pmatrix} p_u \\ p_v \\ p_w \end{pmatrix}. $$

Tal com hem dit, assumirem que les posicions dels punts del sòlid respecte de B són conegudes i es mantenen constants, pel fet que aquest és rígid. Com veurem, per determinar aquestes posicions respecte d'A, serà necessari conèixer l'orientació del sòlid rígid.

A.2 Descripció de l'orientació

Hi ha diverses maneres de parametritzar l'orientació d'un sòlid rígid. Les més característiques són la parametrització mitjançant cosinus director, i la parametrització emprant angles d'Euler.

Representació amb cosinus directors. Sense pèrdua de generalitat, assumirem que l'origen d'A, de vectors unitaris u, v, w, és coincident amb el de B, de vectors unitaris i, j, k. El que pretén aquesta formulació és trobar una matriu de rotació (més genèricament de canvi de base) que permeti expressar un punt del sòlid rígid en A o B indistintament.

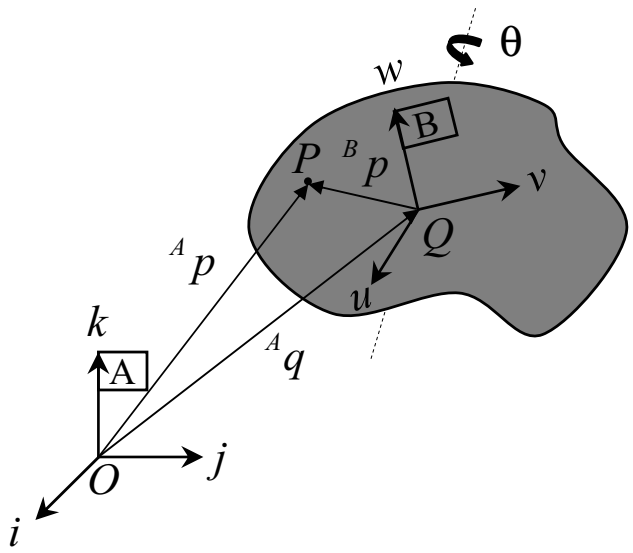


Figura A.1: Elements de la posició i orientació d'un sòlid rígid.

En primer lloc, cal expressar els tres vectors unitaris u, v, w en A,

$$\begin{aligned} {}^{A}u &= u_x i + u_y j + u_z k \\ {}^{A}v &= v_x i + v_y j + v_z k \\ {}^{A}w &= w_x i + w_y j + w_z k. \end{aligned} \tag{A.1}$$

Per altra banda, sabem que un punt p qualsevol del sòlid rígid es pot expressar en ambdues referències,

$${}^{A}p = p_x i + p_y j + p_z k \tag{A.2}$$

$${}^{B}p = p_u u + p_v v + p_w w. \tag{A.3}$$

Si substituïm cada equació de A.1 als vectors unitaris de A.3 trobem:

$${}^{A}p = (p_u u_x + p_v v_x + p_w w_x)i + (p_u u_y + p_v v_y + p_w w_y)j + (p_u u_z + p_v v_z + p_w w_z)k. \tag{A.4}$$

Notem que ara p estarà expressat en A ja que depèn dels vectors unitaris i, j, k. Comparant la darrera equació amb l'equació A.2 arribem al seguit d'igualtats:

$$\begin{aligned} p_x &= u_x p_u + v_x p_v + w_x p_w \\ p_y &= u_y p_u + v_y p_v + w_y p_w \\ p_z &= u_z p_u + v_z p_v + w_z p_w, \end{aligned} \tag{A.5}$$

que dónen lloc al sistema matricial ${}^{A}p = {}^{B}R_{A}\,{}^{B}p$, amb

$${}^{B}R_{A} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

${}^{B}R_{A}$ és la matriu de rotació de B respecte d'A. Amb el teorema *d'Euler* [35, pàg. 34] es pot demostrar que aquesta matriu indueix una orientació del sòlid rígid d'un angle θ respecte un cert eix fix a A (figura A.1). El valor d'aquest angle és $\theta = \arccos(\frac{a_{11}+a_{22}+a_{33}-1}{2})$, i el vector director del l'eix de rotació es pot trobar resolent el sistema $({}^{A}R_{B} - \lambda I){}^{A}p = 0$ per $\lambda = 1$.

Representació en angles d'Euler. Una altra manera d'identificar l'orientació final del sòlid rígid és a partir de tres matrius de rotació respecte els eixos de coordenades x, y, z d'angles α, ϕ, i θ,

$$R_{x,\alpha} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} R_{y,\phi} = \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} R_{z,\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ \cos\theta & 0 & 1 \end{pmatrix}.$$

Com que aquestes rotacions són independents, i l'orientació d'un sòlid rígid consta de tres graus de llibertat, podem emprar aquestes matrius per orientar-lo, fent el producte de les tres, amb l'ordre que es desitgi. Notem que el canvi en l'ordre de multiplicació, pel mateix valor dels angles, donarà lloc a diferents orientacions finals del sòlid rígid.

A.3 Transformacions homogènies

Tal com s'ha introduït anteriorment, la posició i orientació d'un sòlid rígid ve caracteritzada per la posició de l'origen Q de B respecte d'A, i l'orientació de B, també respecte d'A. Així doncs un punt P del sòlid es podrà expressar respecte d'A, com la suma del vector OQ i QP (figura A.1). Si expressem aquests dos vectors en A, tenim que OQ és de fet el vector ${}^{A}q$ que és la translació que hem aplicat al sòlid rígid, i QP és un vector que es pot convertir respecte a A aplicant-li la matriu de rotació ${}^{B}R_{A}$. Finalment obtenim l'expressió que determina la localització d'un sòlid rígid com

$${}^{A}p = {}^{A}q + {}^{B}R_{A}{}^{B}p. \tag{A.6}$$

Podem compactar aquesta expressió encara més, utilitzant una matriu que codifiqui la posició i orientació del sòlid conjuntament. Aquesta matriu s'anomena matriu de *transformació homogènia*, és de dimensió 4×4, i consta de quatre submatrius:

$${}^{B}T_{A} = \begin{pmatrix} {}^{A}R_{B} & \vdots & {}^{A}q \\ \dots & \vdots & \dots \\ \gamma & \vdots & \rho \end{pmatrix}. \tag{A.7}$$

γ és la submatriu 1×3 que representa una *transformació de perspectiva*, ρ és el *factor d'escala*, i les restants matrius ja s'han definit anteriorment. Per cinemàtica de mecanismes i robots manipuladors el factor d'escala és 1, i γ la matriu nul·la. Amb la matriu de transformació homogènia, les coordenades de dos sistemes de referència per a una posició i orientació del sòlid rígid és relacionen com:

$${}^{A}p = {}^{B}T_{A}{}^{B}p, \tag{A.8}$$

amb ${}^{A}p = [p_x, p_y, p_z, 1]^T$ i ${}^{B}p = [p_u, p_v, p_w, 1]^T$.

Finalment, mencionar que aquestes matrius poden ser multiplicades consecutivament per obtenir una matriu de transformació per composició. La demostració és intuïtiva. Sigui un sòlid rígid que va passant per diverses posicions i orientacions fins arribar a la posició i orientació enèsima (figura A.2). Les matrius de transformació de les localitzacions intermèdies donen lloc a les següents equacions:

$$\begin{aligned} {}^{A}p &= {}^{1}T_{A}\,{}^{1}p \\ {}^{1}p &= {}^{2}T_{1}\,{}^{2}p \\ &\dots \\ {}^{n-1}p &= {}^{n}T_{n-1}\,{}^{n}p. \end{aligned} \tag{A.9}$$

Substituint la segona equació de A.9 a la primera, i la tercera a aquesta nova, i així successivament, arribem a l'expressió

$${}^{A}p = {}^{1}T_{A}\,{}^{2}T_{1} \dots {}^{n}T_{n-1}\,{}^{n}p, \tag{A.10}$$

on clarament veiem que la matriu de transformació que relaciona la referència n amb la A és el producte de les transformacions intermèdies ${}^{n}T_{A} = {}^{1}T_{2}\,{}^{2}T_{1} \dots {}^{n}T_{n-1}$.

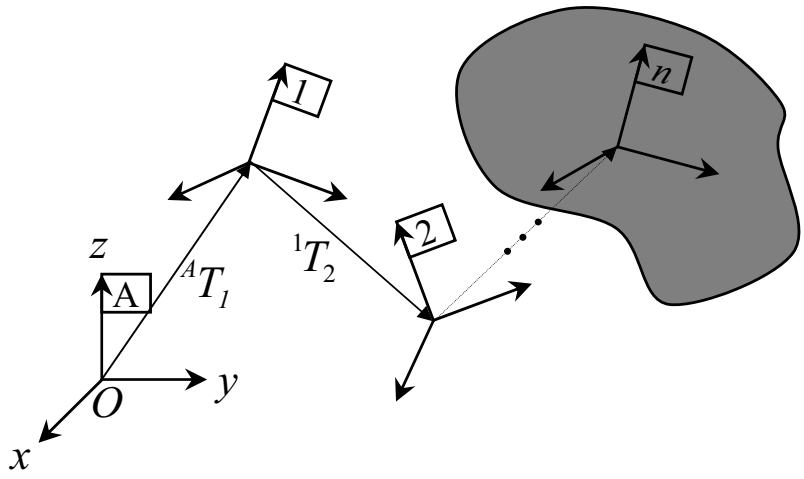


Figura A.2: Posicions i orientacions intermèdies d'un sòlid rígid.

Apèndix B

Anàlisi posicional d'un robot manipulador

B.1 Paràmetres geomètrics

Un robot manipulador de n graus de llibertat està format per una cadena de sòlids rígids connectats entre sí mitjançant n articulacions. Un dels extrems del manipulador és fix al món, i s'anomena *base* i l'altre extrem és lliure i s'anomena *pinça*. Els sòlids rígids es numeren seqüencialment de 0 a n i les articulacions de 1 a n (figura B.1). Per tant, excepte per l'element base i la pinça, cada sòlid rígid té dues articulacions, que poden ser de revolució o bé prismàtiques.

L'objectiu del robot manipulador és que realitzi una tasca específica amb la pinça. Per fer-ho serà necessari saber com canvia la posició i orientació de la pinça relativa a la base quan canviem les variables de les articulacions. Aquest problema a resoldre és l'anomenat problema *cinemàtic directe*. A vegades, però, necessitarem conèixer el problema invers. Per exemple, si coneixem la posició i orientació de l'extrem d'un objecte, i volem que la pinça el subjecti per aquest, haurem de trobar les variables de les articulacions que permeten aquesta subjecció. Aquest és l'anomenat problema *cinemàtic invers*.

B.2 Cinemàtica directa

Més formalment, el problema cinemàtic directe consisteix en trobar l'orientació i la posició de la pinça a partir de variables de les articulacions $q_1, \cdots, q_n$ (figura B.2). És comú en robòtica denotar per TCP el sistema de referència solidari a la pinça.

B.3 Cinemàtica inversa

El problema cinemàtic invers consisteix en trobar els valors de les variables de les articulacions que són necessàries per obtenir una posició i orientació del TCP desitjada (figura B.3). És important notar que mentre el problema de la cinemàtica directa dóna una única solució, és a dir, troba una única posició i orientació de la pinça, el problema de la cinemàtica inversa pot tenir vàries solucions.

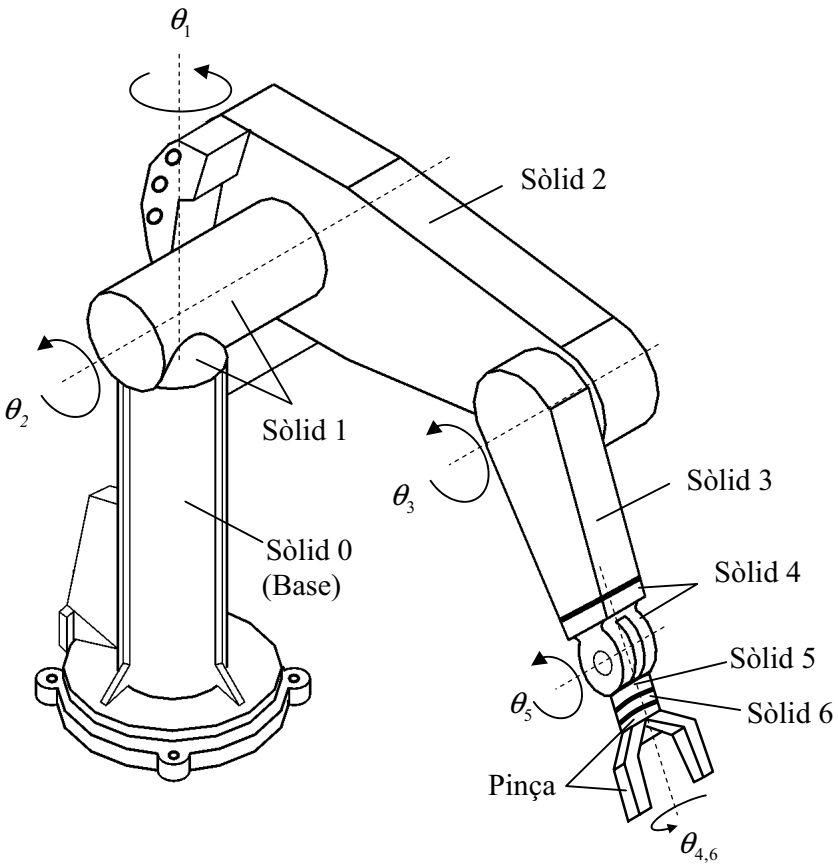


Figura B.1: Un exemple de robot manipulador, el robot puma.

B.4 Paràmetres de Denavit-Hartenberg

S'han proposat uns quants procediments per resoldre el problema cinemàtic directe i invers. Els més usats són possiblement: el mètode d'àlgebra de vectors [5, 23, 22], el mètode geomètric [8, 7], i el mètode de la matriu 4×4 [6]. Aquest últim, també anomenat representació de Denavit-Hartenbeg, és el que utilitzarem perquè permet representar de forma matricial la relació translacional i rotacional entre sòlids rígids adjacents d'una cadena cinemàtica. Primer defineix un sistema de coordenades lligat a cadascun dels sòlids rígids de manera que cada sistema de coordenades estigui referenciat respecte l'anterior.

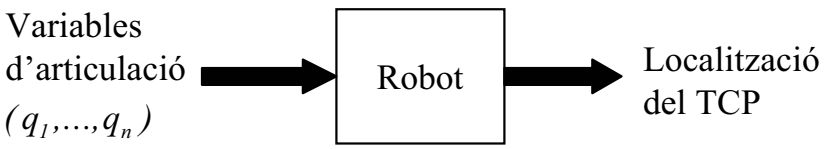


Figura B.2: Esquema de la cinemàtica directa.

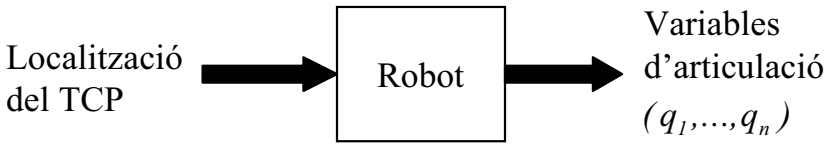


Figura B.3: Esquema de la cinemàtica inversa.

Després s'estableix una matriu de transformació entre sistemes de referència adjacents, amb la propietat que aquesta depèn de la variable d'articulació existent entre les dues referències.

Regles de posicionament dels sistemes de referència. En primer lloc, hem de dir que el sistema de referència i-èsim, és solidari al sòlid rígid i-èsim i les regles per posicionar-lo són:

- L'eix z_i es situa al llarg de l'articulació $(i+1)$.
- L'eix x_i és defineix al llarg de la normal comuna entre els eixos z_{i-1} i z_i. Si els dos eixos són paral·lels, l'eix x_i es tria a qualsevol direcció normal a ambdós eixos. I en cas que s'intersectin, la direcció de l'eix x_i es pot trobar fent el producte $z_{i-1} \times z_i$.
- L'eix y_i es determina fent que el sistema de referència i-èsim compleixi la regla de la mà dreta.

El sistema de referència zero es situa solidari a la base del robot, amb l'eix z_0 al llarg de l'articulació 1. Els altres eixos x_0, y_0 es situen en la direcció que desitgi l'usuari, però fent que la referència resultant satisfaci la regla de la mà dreta. En quan al TCP pot tenir una posició i orientació qualssevol, però amb l'eix x_n sent normal a la direcció de l'última articulació.

Matriu de transformació ${}^iT_{i-1}$. Un cop s'han definit els sistemes de referència podem obtenir de la figura B.4 els quatre paràmetres D-H: θ_i, α_i, a_i i d_i, que relacionen la transformació homogènia de la referència i respecte la $i-1$. Només cal saber que:

- θ_i és l'angle entre x_{i-1} i x_i mesurat respecte z_{i-1}.
- α_i és l'angle existent entre els eixos de les articulacions i i $i-1$, que coincideix amb l'angle entre z_{i-1} i z_i mesurat respecte x_i.
- d_i és la distància entre x_{i-1} a x_i mesurada al llarg de z_{i-1}. En realitat és una distància amb signe, que té valor positiu si el vector z_{i-1} té el sentit d'anar de x_{i-1} a x_i.
- a_i és la distància de z_{i-1} mesurada al llarg de x_i i com d_i el seu valor pot ser negatiu.

Per una articulació de revolució a_i, α_i, i d_i són constants, i θ_i és una variable que mesura el gir relatiu del sòlid rígid i respecte el $i-1$. A més podem definir θ_i com $\theta_i = \theta_i' + \theta_i^{ofset}$, on θ_i^{ofset} és l'angle existent entre x_{i-1} i x_i abans de que es produeixi un gir en l'articulació i. Per una articulació prismàtica a_i, α_i i θ_i són constants, mentre que d_i és variable i també es pot expressar com $d_i = d_i' + d_i^{ofset}$.

Coneguts els paràmetres de D-H, la matriu de transformació homogènia ${}^iT_{i-1}$ entre dos sistemes de coordenades consecutius, que denotarem com A_i, es pot expressar com el producte de les quatre matrius de transformació associades als paràmetres de D-H,

$$A_i = T_{z,d_i}T_{z,\theta_i}T_{x,a_i}T_{x,\alpha_i} = \begin{pmatrix} \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\theta_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad \text{(B.1)}$$

Mitjançant els paràmetres de D-H, la cinemàtica directa resulta força senzilla ja que la posició i orientació de la pinça respecte la base s'obté a través d'aplicar el producte de les diferents matrius de transformació de la cadena cinemàtica (veure equació A.9). Per a una cadena de n graus de llibertat la transformació del sòlid n respecte la base és:

$${}^nT_0 = A_1A_2\cdots A_6. \quad \text{(B.2)}$$

Si a més tenim en compte la matriu de transformació del TCP respecte la referència solidaria al sòlid rígid n, l'expressió queda:

$${}^{TCP}T_0 = A_1A_2\cdots A_6^{TCP}T_n. \quad \text{(B.3)}$$

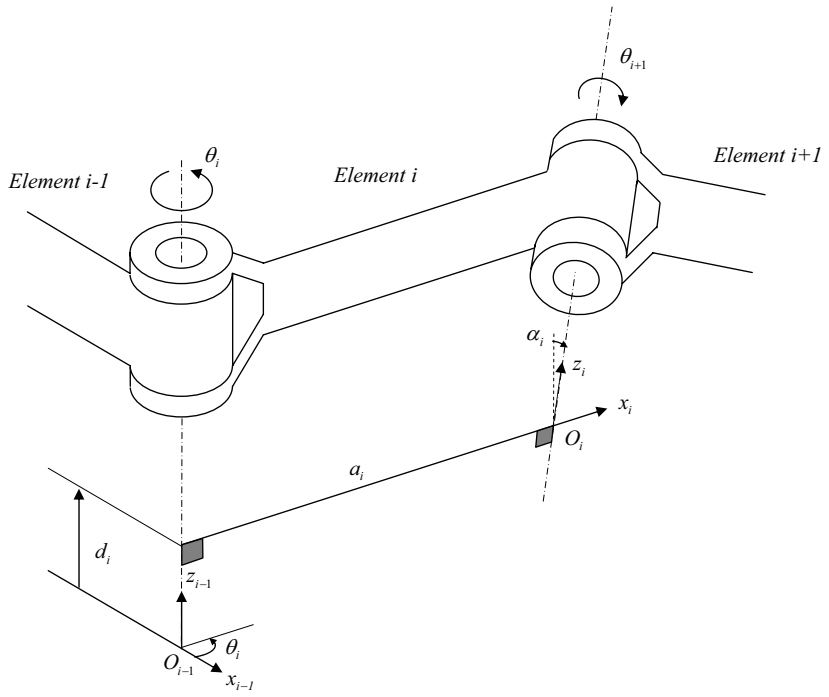


Figura B.4: Interpretació gràfica dels paràmetres de D-H.

Resoldre la cinemàtica inversa és més complex que la directa, i l'estratègia a seguir és intentar resoldre el sistema d'equacions matricials que s'obté d'igualar l'expressió ${}^{n}T_0$ de l'equació B.2 amb una expressió genèrica com la següent,

$$ {}^{n}T_0 = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.4} $$

on tots els valors de les columnes representen valor numèrics reals. Notem que resolem el sistema B.2 malgrat que hauríem de resoldre el sistema B.3 per ser coherents amb la definició de cinemàtica inversa. Això és així perquè els dos sistemes B.2 i B.3 són equivalents, ja que la matriu ${}^{TCP}T_n$ és sempre una dada i constant al llarg de tot problema (la pinça no aporta cap grau de llibertat que afecti en l'orientació i posicionament del TCP). En l'annex C.3, es resol la cinemàtica directa i la inversa per un robot Stäubli RX60.

Apèndix C

Anàlisi del robot manipulador Stäubli RX60

C.1 Característiques tècniques

El robot RX60 consta d'una cadena cinemàtica de set sòlids rígids units entre sí mitjançant sis articulacions de revolució. També es pot anomenar braç robòtic, ja que té disposades les articulacions d'una forma semblant a les d'un braç humà, amb la base fixa al terra. Així doncs els diferents sòlids rígids del braç són: la base (A), l'espatlla (B), el braç (C), el colze (D), l'avantbraç (E) i el canell (F) . D'origen el braç RX60 no porta incorporada cap pinça, permetent així que l'usuari inserti la més adequada a les necessitats de treball del robot.

A continuació es mostren les característiques geomètriques més rellevants pel desenvolupament del present projecte: una vista en tres dimensions per veure la seva arquitectura articular, el seu plànol de cotes que servirà per determinar els paràmetres de Denavit-Hartenberg (figura C.1), i finalment el volum de treball del robot, que a més indica el sentit positiu de cada una de les articulacions (figura C.2).

C.2 Cinemàtica directa d'un manipulador simple

Cinemàticament el braç robòtic pertany a la família de robots *manipuladors simples*. Aquests tenen les articulacions amb una arquitectura predeterminada: les tres primeres posicionen l'objecte en les coordenades x, y, z de l'espai de treball, i les tres últimes, amb eixos interseccionant en un punt, determinen la seva orientació. L'anàlisi cinemàtic d'aquests manipuladors s'ha estudiat profundament en la literatura, i s'ha demostrat que es pot obtenir una solució explícita de les seves equacions cinemàtiques, tal com fa Paul en el seu treball [28]. Paul empra la representació de Denavit-Hartenberg per fer aquesta anàlisi a un robot Puma, i això dóna lloc a una taula de paràmetres de D-H (taula C.1). Avaluant els paràmetres de cada articulació a l'expressió de la matriu B.1 trobem les matrius de transformació A_i entre sòlids consecutius. La solució de la cinemàtica directa

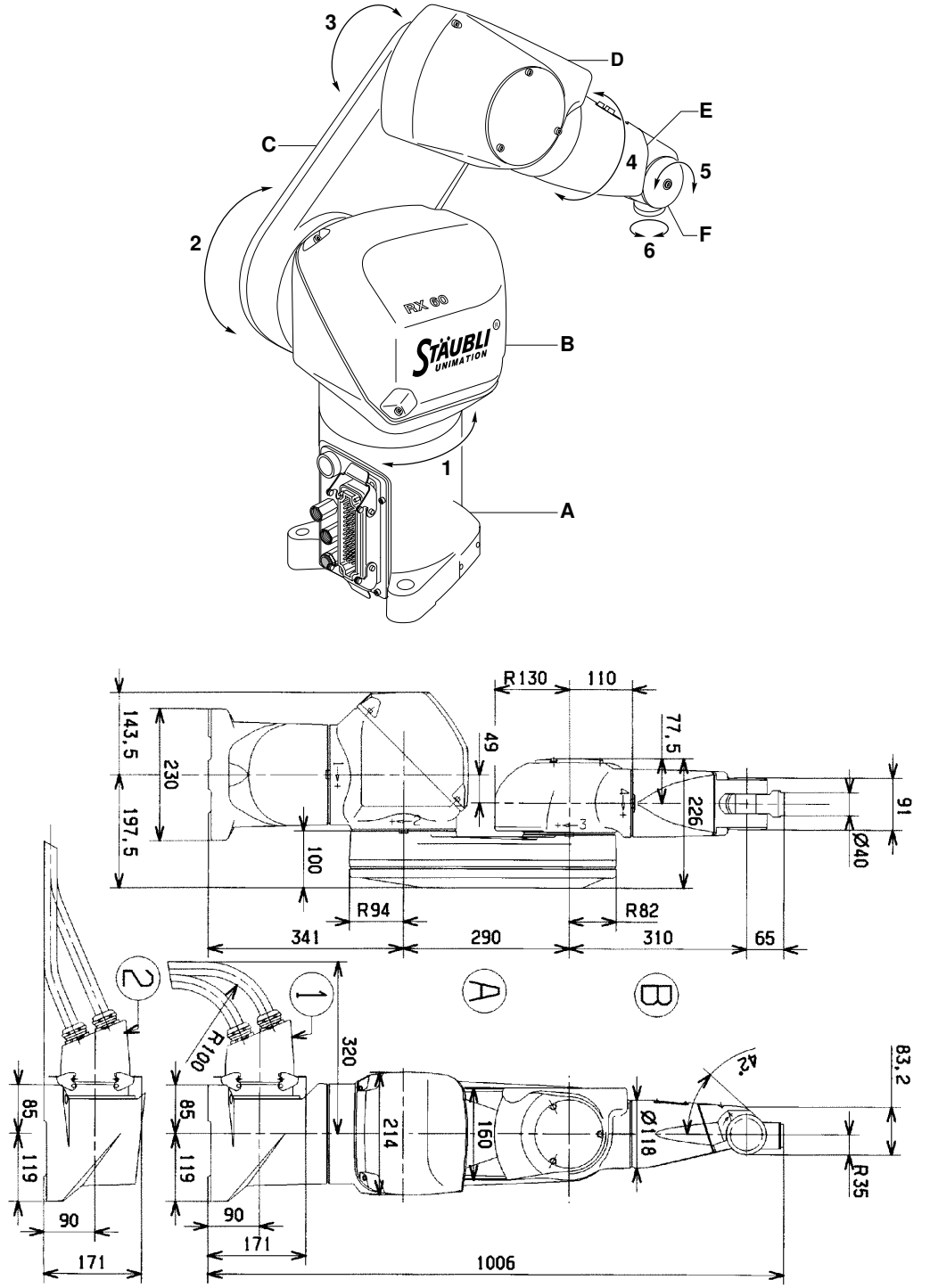


Figura C.1: Representació del braç robòtic RX60 i del seu plànol de cotes.

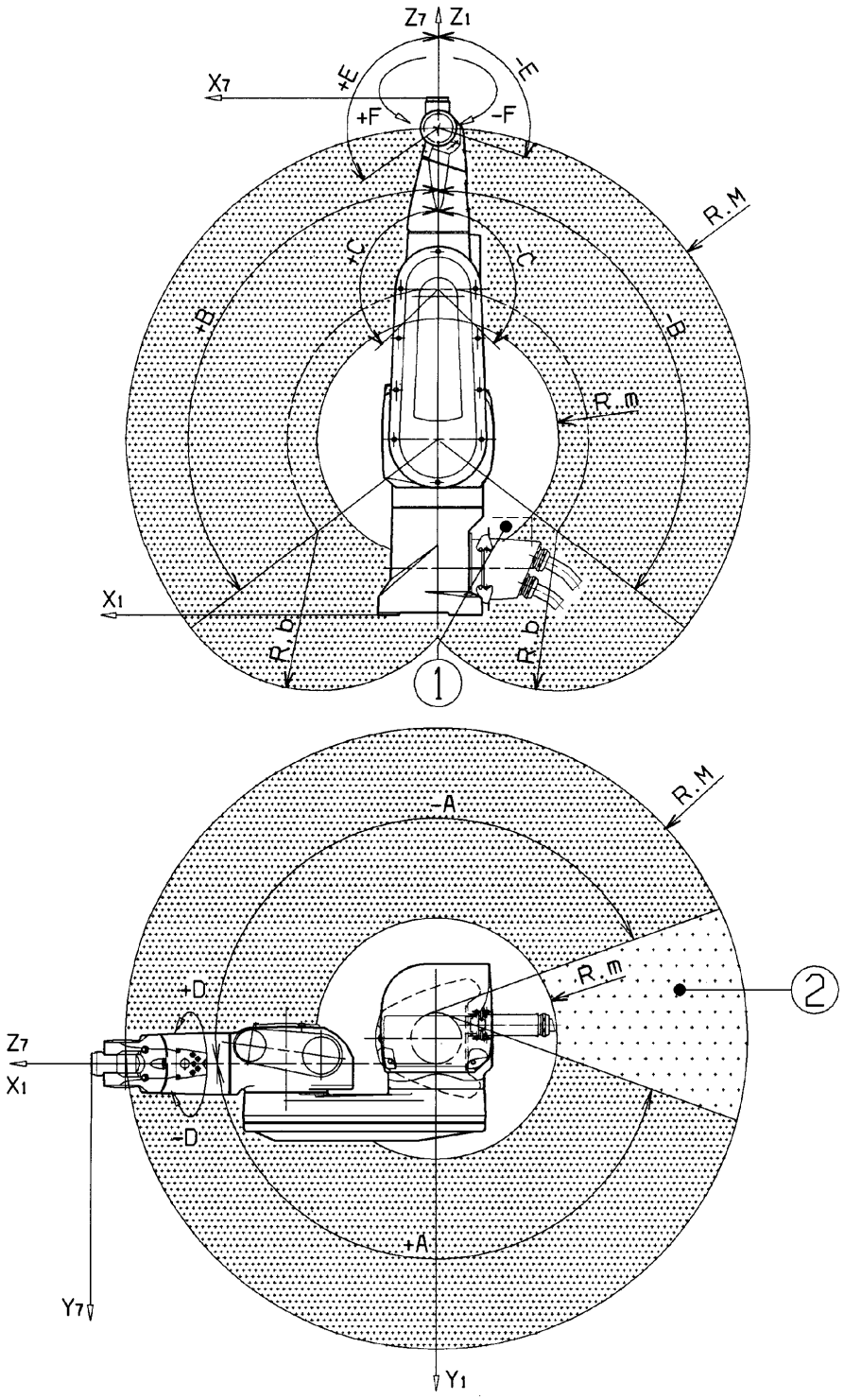


Figura C.2: Volum de treball del braç RX60.

Artic.	α_i	θ_i^{ofs}	d_i	a_i
1	-90º	180º	0	0
2	0	-90º	a_2	0
3	90º	90º	0	d_3
4	-90º	0	d_4	0
5	90º	0	0	0
6	0	0	0	0

$$\theta_i{=}\theta_i'{+}\theta_i^{ofs}.$$

Taula C.1: Taula dels paràmetres de D-H per un robot Puma.

és el producte successiu d'aquestes matrius, la matriu

$$ {}^6T_0 = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{C.1}$$

que lliga el sistema de referència del sòlid rígid 6 del robot respecte el de la base

$$n_x = C_1[C_{23}(C_4C_5C_6 - S_4S_6) - S_{23}S_5S_6] - S_1[S_4C_5C_6 + C_4S_6] \tag{C.2}$$
$$n_y = S_1[C_{23}(C_4C_5C_6 - S_4S_6) - S_{23}S_5C_6] + C_1[S_4C_5C_6 + C_4S_6] \tag{C.3}$$
$$n_z = -S_{23}(C_4C_5C_6 - S_4S_6) - C_{23}S_5C_6 \tag{C.4}$$
$$o_x = C_1[-C_{23}(C_4C_5C_6 + S_4C_6) + S_{23}S_5C_6] - S_1[-S_4C_5S_6 + C_4C_6] \tag{C.5}$$
$$o_y = S_1[-C_{23}(C_4C_5S_6 + S_4C_6) + S_{23}S_5S_6] + C_1[-S_4C_5S_6 + C_4C_6] \tag{C.6}$$
$$o_z = S_{23}(C_4C_5S_6 + S_4C_6) + C_{23}S_5S_6 \tag{C.7}$$
$$a_x = C_1(C_{23}C_4S_5 + S_{23}C_5) - S_1S_4S_5 \tag{C.8}$$
$$a_y = S_1(C_{23}C_4S_5 + S_{23}C_5) + C_1S_4S_5 \tag{C.9}$$
$$a_z = -S_{23}C_4S_5 + C_{23}C_5 \tag{C.10}$$
$$p_x = C_1(d_4S_{23} + a_3C_{23} + a_2C_2) - S_1d_3 \tag{C.11}$$
$$p_y = S_1(d_4S_{23} + a_3C_{23} + a_2C_2) + C_1d_3 \tag{C.12}$$
$$p_z = -(-d_4C_{23} + a_3S_{23} + a_2S_2), \tag{C.13}$$

amb S_i sent $\sin\theta_i$, C_i $\cos\theta_i$, S_{23} $\sin(\theta_2+\theta_3)$ i C_{23} $\cos(\theta_2+\theta_3)$.

C.3 Cinemàtica inversa d'un manipulador simple

Primer de tot definim U_i com:

$$U_i = A_i \cdots A_6. \tag{C.14}$$

Ara donada la transformació ${}^6T_0$, intentem resoldre el sistema matricial

$${}^6T_0 = U_2. \tag{C.15}$$

Del sistema C.15 no trobem cap equació que resolgui alguna de les variables angulars. Aleshores plantegem successivament els següents sistemes matricials (sistemes de C.16 a C.20) fins que determinem les sis solucions angulars,

$$A_1^{-1^6}T_0 = U_2 \tag{C.16}$$
$$A_2^{-1}A_1^{-1^6}T_0 = U_3 \tag{C.17}$$
$$A_3^{-1}A_2^{-1}A_1^{-1^6}T_0 = U_4 \tag{C.18}$$
$$A_4^{-1}A_3^{-1}A_2^{-1}A_1^{-1^6}T_0 = U_5 \tag{C.19}$$
$$A_5^{-1}A_4^{-1}A_3^{-1}A_2^{-1}A_1^{-1^6}T_0 = U_6. \tag{C.20}$$

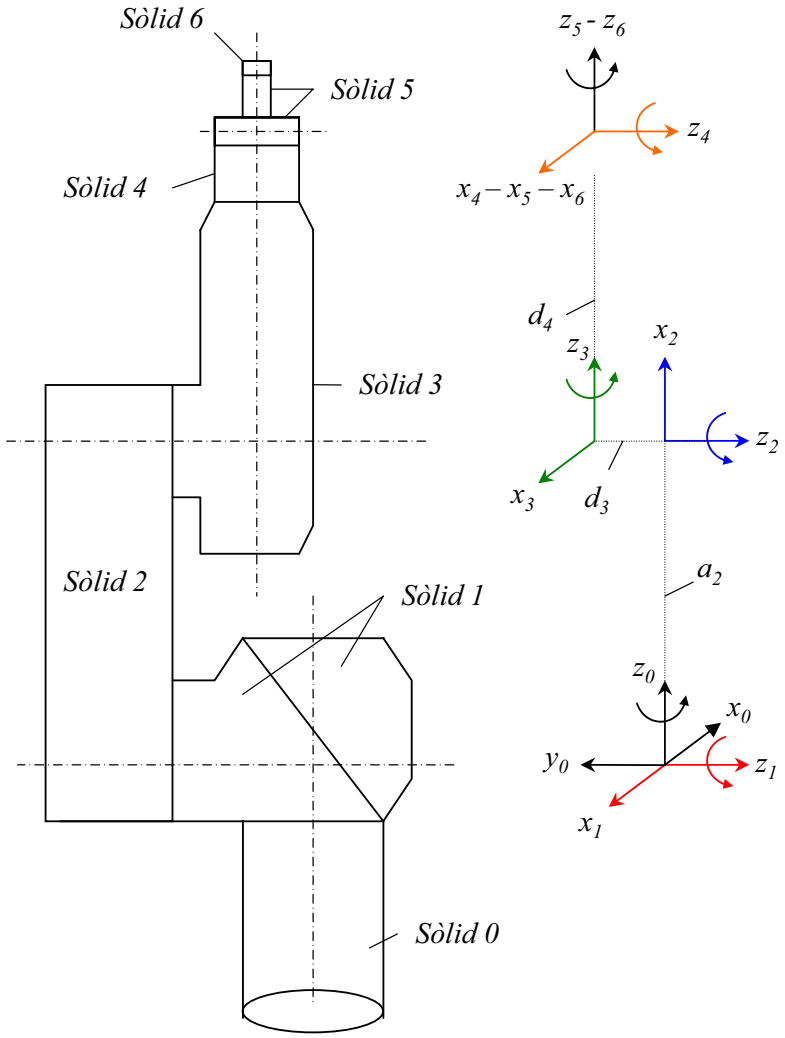


Figura C.3: Sistemes de referència deduïts de la taula C.1

Del sistema C.16 igualant els termes $(3,4)$ de les matrius esquerra i dreta de la igualtat obtenim:

$$-S_1 p_x + C_1 p_y = -d_3 \tag{C.21}$$

Aquesta equació només té una incògnita que és θ_1. Si fem el següent canvi de variables

$$p_x = r\cos\phi \tag{C.22}$$
$$p_y = r\sin\phi \tag{C.23}$$
$$r = +\sqrt{p_x^2 + p_y^2} \tag{C.24}$$
$$\phi = \arctan\left(\frac{p_y}{p_x}\right) \tag{C.25}$$

a l'equació C.21 i apliquem propietats trigonomètriques arribem a

$$\theta_1 = \arctan\left(\frac{p_y}{p_x}\right) + \arctan\left(\frac{d_3}{\pm\sqrt{r^2 + d_3^2}}\right), \quad \text{amb} \quad 0 < d3/r \leq 1. \tag{C.26}$$

Es pot comprovar que la matriu esquerra del sistema C.16 només depèn de θ_1. Ara igualant els termes $(1,4)$ i $(2,4)$ d'ambós membres del sistema C.16 obtenim dues equacions que només depenen de θ_2 i θ_3,

$$C_1 p_x + S_1 p_y = d_4 S_{23} + a_3 C_{23} + a_2 C_2 \tag{C.27}$$
$$-p_z = -d_4 C_{23} + a_3 S_{23} + a_2 S_2. \tag{C.28}$$

Si anomenem $f_{11_p} = C_1 p_x + S_1 p_y$ i $f_{12_p} = -p_z$ i sumem les equacions C.27 i C.28 arribem després de certes simplificacions a la relació explícita de θ_3, que veiem que és funció de l'angle θ_1,

$$\theta_3 = \arctan\left(\frac{a_3}{-d_4}\right) + \arctan\left(\frac{d}{\pm\sqrt{e - d}}\right), \tag{C.29}$$

a on

$$d(\theta_1) = f_{11_p}^2 + f_{12_p}^2 - d_4^2 - a_3^2 - a_2^2 \tag{C.30}$$
$$e = 4a_2^2 a_3^2 + 4a_2^2 d_4^2 \qquad (constant). \tag{C.31}$$

Igualant els termes del sistema C.17 no arribem a cap equació que puguem resoldre. En canvi, del sistema C.18, igualant els termes $(1,4)$ i $(3,4)$ obtenim dues equacions on les úniques variables són C_{23} i S_{23},

$$C_{23} f_{11_p} - S_{23} p_z = a_2 C_3 + a_3 \tag{C.32}$$
$$S_{23} f_{11_p} + C_{23} p_z = d_4 + a_2 S_3. \tag{C.33}$$

Un cop resoltes S_{23} i C_{23} fàcilment podem obtenir θ_{23} aplicant l'arctangent del quocient d'aquestes variables:

$$\theta_{23} = \arctan\frac{w_2 f_{11_p} - w_1 p_z}{w_1 f_{11_p} + w_2 p_z}, \tag{C.34}$$

a on

$$w_1 = a_2C_3 + a_3 \tag{C.35}$$
$$w_2 = d_4 + a_2S_3. \tag{C.36}$$

I per tant, ja podem trobar θ_2

$$\theta_2 = \theta_{23} - \theta_3. \tag{C.37}$$

Ara tots els termes del membre esquerra del sistema C.18 són coneguts. Els termes $(1,3)$ i $(2,3)$ donen equacions on intervenen els angles θ_4 i θ_5

$$C_4S_5 = C_{23}(C_1a_x + S_1a_y) - S_{23}a_z \tag{C.38}$$
$$S_4S_5 = -S_1a_x + C_1a_y. \tag{C.39}$$

Si $\sin\theta_5 \neq 0$ podem dividir les equacions C.39 i C.38 i aplicar l'arctangent per trobar

$$\theta_4 = \arctan\frac{-S_1a_x + C_1a_y}{C_{23}(C_1a_x + S_1a_y) - S_{23}a_z}. \tag{C.40}$$

Si $\theta_5 < 0$ aleshores $\theta_4' = \theta_4 + 180^o$. Quan $\sin\theta_5 = 0$, $(\theta_5 = 0)$ veiem que la igualtat entre C.38 i C.39 es compleix per qualsevol valor de θ_4. Diem que ens trobem en un cas degenerat, i això correspon a quan l'eix 4 i el 6 es troben alineats. En aquesta situació, també podem acceptar com a valor θ_4 l'obtingut per l'equació C.40.

Ara del sistema C.19 podem trobar dues equacions que ens relacionen θ_5 igualant els termes $(1,3)$ i $(2,3)$,

$$S_5 = C_4[C_{23}(C_1a_x + S_1a_y) - S_{23}a_z] + S_4[-S_1a_x + C_1a_y] \tag{C.41}$$
$$C_5 = S_{23}(C_1a_x + S_1a_y) + C_{23}a_z. \tag{C.42}$$

Fent la divisió de les equacions C.41 i C.42 i aplicant l'arctangent a ambdós membres arribem a

$$\theta_5 = \arctan\frac{C_4[C_{23}(C_1a_x + S_1a_y) - S_{23}a_z] + S_4[-S_1a_x + C_1a_y]}{S_{23}(C_1a_x + S_1a_y) + C_{23}a_z}. \tag{C.43}$$

I finalment, del sistema C.19 igualant els elements $(1,2)$ i $(2,2)$ d'ambdós membres trobem:

$$\begin{aligned} S_6 = &-C_5\{C_4[C_{23}(C_1o_x + S_1o_y) - S_{23}o_z] + S_4[-S_1o_x + C_1o_y]\} \\ &+ S_5\{S_{23}(C_1o_x + S_1o_y) + C_{23}o_z\} \end{aligned} \tag{C.44}$$

$$C_6 = -S_4[C_{23}(C_1o_x + S_1o_y) - S_{23}o_z] + C_4[-S_1o_x + C_1o_y], \tag{C.45}$$

sent l'equació de θ_6:

$$\theta_6 = \arctan\frac{-C_5\{C_4[C_{23}(C_1o_x + S_1o_y) - S_{23}o_z] + S_4[-S_1o_x + C_1o_y]\} + S_5\{S_{23}(C_1o_x + S_1o_y) + C_{23}o_z\}}{-S_4[C_{23}(C_1o_x + S_1o_y) - S_{23}o_z] + C_4[-S_1o_x + C_1o_y]} \tag{C.46}$$

Les solucions angulars anteriors tenen les següents dependències:

$$\theta_1 = \pm f(p_x, p_y) \tag{C.47}$$
$$\theta_3 = \pm f(p_x, p_y, p_z, \theta_1) \tag{C.48}$$
$$\theta_2 = f(p_x, p_y, p_z, \theta_1, \theta_3) \tag{C.49}$$
$$\theta_4 = f(a_x, a_y, a_z, p_x, p_y, p_z, \theta_1, \theta_3) \quad \text{o} \quad f(a_x, a_y, a_z, p_x, p_y, p_z, \theta_1, \theta_3) + 180^{\text{o}} \tag{C.50}$$
$$\theta_5 = f(a_x, a_y, a_z, p_x, p_y, p_z, \theta_1, \theta_3, \theta_4) \tag{C.51}$$
$$\theta_6 = f(a_x, a_y, a_z, o_x, o_y, o_z, p_x, p_y, p_z, \theta_1, \theta_3, \theta_4) \tag{C.52}$$

L'arbre que presentem a la figura C.4 mostra que com a molt, la cinemàtica inversa, donada una matriu ${}^6T_0$, té 8 conjunts de solucions angulars, ja que trobades θ_1, θ_3, θ_4, la resta de solucions són úniques. Quan l'articulació 1 presenta la solució θ_1^+ es diu que és una solució a esquerres("lefty"), i a dretes ("righty") per la solució θ_1^-. Per a θ_3^+ es diu que l'articulació 3 tres presenta una disposició de colze amunt ("up"), i colze avall ("down") per la solució θ_3^-. I finalment, per l'articulació 4 tenim que θ_4^- ("flip") és un gir de 180º respecte la solució θ_4^+ ("no-flip") (figura C.5).

C.4 Solucions del RX60

C.4.1 Paràmetres de Denavit-Hartenberg

Com que el braç RX60 pertany a la família de manipuladors simples, podem emprar els mateixos paràmetres que els de la taula C.1. Ara bé, els valors a_i i d_i dependran de la

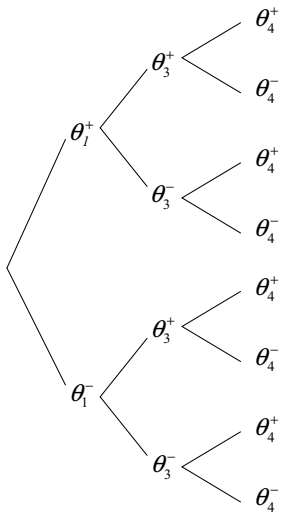


Figura C.4: Arbre de solucions de la cinemàtica inversa per un manipulador simple.

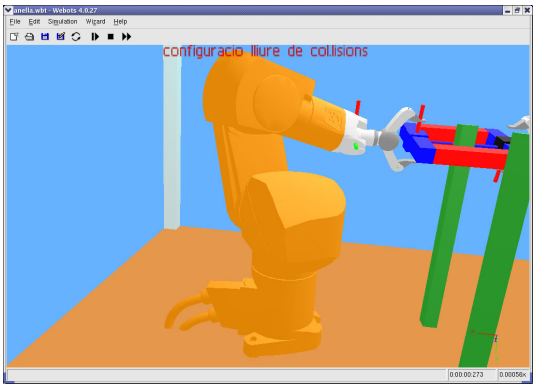
geometria del robot. Analitzant el plànol de cotes de la figura C.1 podem determinar-los

$$\begin{aligned} a_2 &= 0.290 \\ d_3 &= 0.049 \\ d_4 &= 0.31, \end{aligned} \tag{C.53}$$

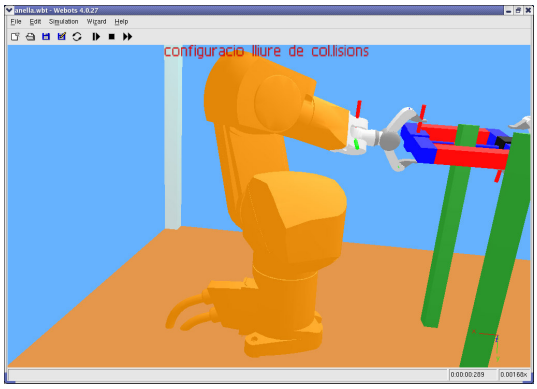
on tots els valors estan donats en metres.

C.4.2 Rang de treball dels angles

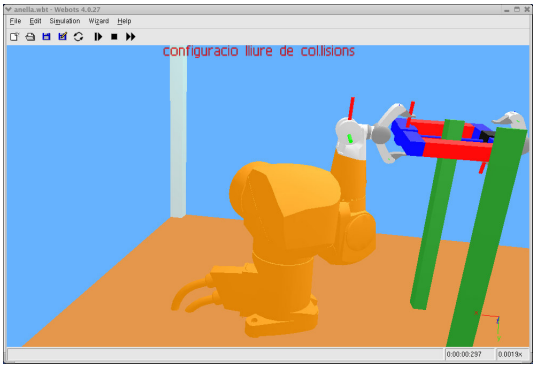
Els braços RX60 a més tenen establerts uns rangs de treball per les seves articulacions. La taula C.2 mostra aquests valors:



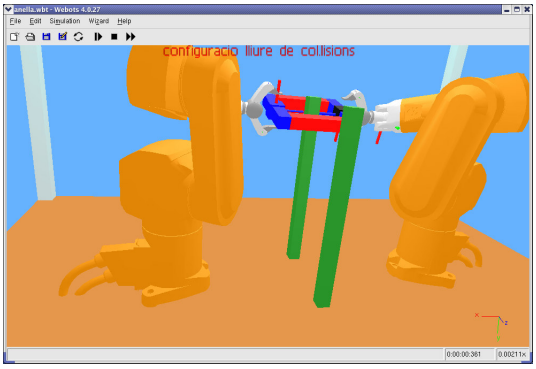
(a) Solució run



(b) Solució ruf



(c) Solució rdn



(d) Solució luf

Figura C.5: .

Artic.	1	2	3	4	5	6
θ_{max}	160º	127.5º	134.5º	270º	120.5º	270º
θ_{min}	-160º	-127.5º	-134.5º	-270º	-109.5º	-270º

Taula C.2: Valors màxims i mínims dels angles de les articulacions del braç RX60.

C.4.3 Solucions angulars

Convé notar, que totes les solucions trobades de l'anàlisi de Paul són vàlides per als seus sistemes de referència, i per tant seria desitjable que utilitzéssim els mateixos per al nostre robot RX60, per tal que les solucions fossin aplicables directament. De fet bé ho podríem fer, car el robot RX60 és també un manipulador simple. El problema està en que el RX60 té tres articulacions que tenen un sentit de gir contrari al del treball de Paul. Així doncs caldrà fer un canvi de signe per passar els angles de les referències de Paul a les del RX60 i viceversa. També cal afegir, que pel robot RX60 tindrem en compte des d'un principi els "ofsets" θ_i^{ofs} de cada articulació. (taula C.1). Introduïts aquests girs, el robot es trobarà a la configuració de "home"(quan els eixos de les articulacions 1, 5, i 6 estan alineats) i en aquesta configuració prendrem com a nuls tots els valors angulars. Així doncs, donat un angle solució de la cinemàtica inversa, l'angle que introduirem al robot serà $\theta_i^{RX60} = \theta_i - \theta_i^{ofs}$, ja que θ_i^{ofs} s'ha tingut en compte inicialment. Les conversions angulars entre les referències de Paul i les del braç RX60 tenint en compte els ofsets i els diferents sentits de gir són:

$$\theta_1^{RX60} = \theta_1 - \theta_1^{ofs} \tag{C.54}$$
$$\theta_2^{RX60} = -(\theta_2 - \theta_2^{ofs}) \tag{C.55}$$
$$\theta_3^{RX60} = -(\theta_3 - \theta_3^{ofs}) \tag{C.56}$$
$$\theta_4^{RX60} = \theta_4 \tag{C.57}$$
$$\theta_5^{RX60} = -\theta_5 \tag{C.58}$$
$$\theta_6^{RX60} = \theta_6 \tag{C.59}$$

Finalment, dir que sempre que treballem amb les rutines de cinemàtica directa/inversa, els angles hauran d'estar en la referència del treball de Paul, i quan haguem d'aplicar uns angles al braç robòtic, els valors angulars hauran d'estar referits a la referència del braç RX60.

Apèndix D

Anàlisi posicional de l'objecte a transportar

D.1 Sistemes de referència

Per tal que les pinces dels robots RX60 subjectin l'objecte per dos extrems desitjats cal definir les matrius que anomenem de *prensió esquerra* i *dreta*. La matriu de prensió esquerra és la matriu ${}^{TCP_a}T_{obj}$ que defineix com té que posicionar-se i orientar-se el TCP del robot actiu respecte la referència central de l'objecte. La matriu de prensió esquerra denotada com ${}^{TCP_p}T_{obj}$ té el mateix significat però pel TCP del robot passiu. D'aquesta manera, si coneixem les característiques geomètriques de l'objecte podrem subjectar-lo pels extrems que desitgem. Notem que per qualsevol de les dues matrius, l'eix z del TCP associat s'ha de trobar perpendicular i cap a dins de la superfície de l'objecte. D'altra forma les pinces del robot mai la podrien subjectar per aquell extrem.

D.2 Cinemàtica directa

A continuació s'indica la solució de la cinemàtica directa per l'objecte, que consisteix en trobar la matriu de transformació entre la referència central d'aquest (obj) i l'absoluta del mon (R). Per calcular-la necessitarem disposar d'un vector de translació de la referència de l'objecte (p_x, p_y, p_z), i tres angles d'Euler d'eixos de rotació consecutius x, y i z (θ, α, γ), que per simplicitat en l'anàlisi els denotarem com (x, y, z). Amb aquesta parametrització la matriu desitjada és pot expressar com:

$$ {}^{obj}T_R = T_{transl} T_{x,\theta} T_{y,\alpha}, T_{z,\gamma}, $$

i finalment el producte dóna

$$ {}^{obj}T_R = \begin{pmatrix} c(y)c(z) & -c(y)s(z) & s(y) & p_x \\ s(x)s(y)c(z) + c(x)s(z) & -s(x)s(y)s(z) + c(x)c(z) & -s(x)c(y) & p_u \\ -c(x)s(y)c(z) + s(x)s(z) & c(x)s(y)s(z) + s(x)c(z) & c(x)c(y) & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \text{(D.1)} $$

amb $c(a)$ com $\cos a$ i $s(a)$ com $\sin a$.

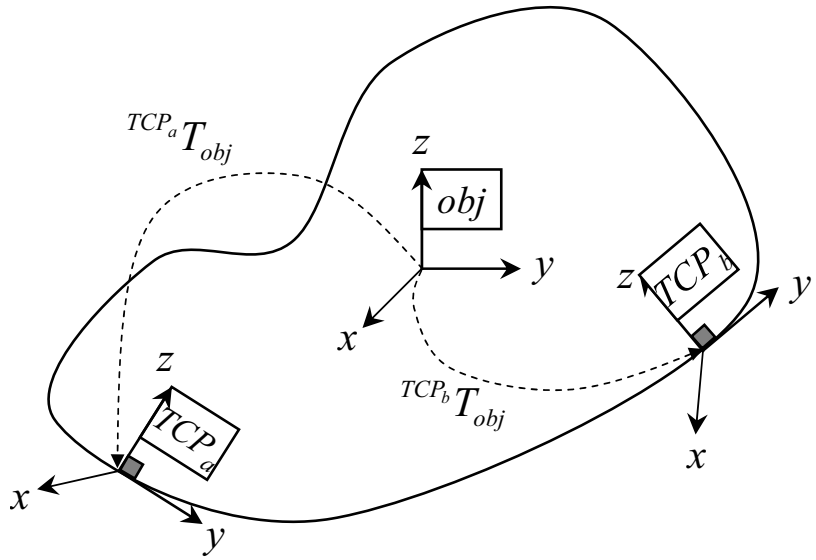


Figura D.1: Sistemes de referència de l'objecte a transportar.

D.3 Cinemàtica inversa

Ara el problema consisteix en donada una matriu de transformació ${}^{obj}T_R$ de l' objecte trobar els paràmetres de translació (p_x, p_y, p_z) i rotació (x, y, z) que el descriuen. Suposem que la matriu ${}^{obj}T_R$ donada té l'expressió següent

$$ {}^{n}T_0 = \begin{pmatrix} n_x & o_x & a_x & d_x \\ n_y & o_y & a_y & d_y \\ n_z & o_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{D.2} $$

on tots els valors de les columnes representen valors numèrics reals.

El vector de translació (p_x, p_y, p_z) es troba directament igualant la tercera columna de la matriu D.1 i la matriu D.2. Així doncs trobem:

$$ p_x = d_x \tag{D.3} $$
$$ p_y = d_y \tag{D.4} $$
$$ p_z = d_z. \tag{D.5} $$

a) Angle z

Si ara fem la divisió de les equacions que sorgeixen d'igualar les components $(1,1)$ i $(1,2)$ de les matrius D.2 i D.1 arribem a una equació que només depèn de z.

$$ -\tan z = (\frac{\sigma_x}{n_x}) \quad \text{amb} \quad c(y) \neq 0. \tag{D.6} $$

Per poder aplicar l'arctangent sense haver de suposar a quin quadrant pertany la solució

podem treure el signe sabent que $-\tan(z) = \tan(180 - z)$,

$$z = 180 - \arctan 2(\frac{\sigma_x}{n_x}) \quad \text{amb} \quad c(y) \neq 0, \tag{D.7}$$

on *arctan2* és la funció arctangent però que retorna l'angle en el quadrant adient definit pel signe del numerador i del denominador del seu argument.

a) Angle x

Podem trobar l'angle x fent la divisió de les equacions que s'obtenen d'igualar les components $(2,3)$ i $(3,3)$ de les matrius D.2 i D.1.

$$-\tan x = (\frac{a_y}{a_z}) \quad \text{amb} \quad c(y) \neq 0. \tag{D.8}$$

Com el cas anterior podem resoldre l'equació aplicant propietats trigonomètriques i arribem:

$$z = 180 - \arctan 2(\frac{a_y}{a_z}) \quad \text{amb} \quad c(y) \neq 0. \tag{D.9}$$

a) Angle y

Si igualem les components $(2,3)$ i $(3,3)$ de les dues matrius D.2 i D.1 arribem a dues equacions, que les podem elevar al quadrat i després sumar-les. A continuació igualem les components $(1,3)$, les elevem al quadrat i fem la divisió amb la darrera equació. Si aquesta equació li fem un tractament trigonomètric veiem que només depèn de l'angle y i la solució és la següent

$$y = \arctan 2(\frac{a_x}{\pm\sqrt{a_y^2 + a_z^2}}). \tag{D.10}$$

Com veiem en l'equació D.10 hem posat dos signes al denominador. El signe serà positiu quan $c(y) > 0$, i això passarà quan n_x i c_z tinguin el mateix signe, d'altra banda el signe serà negatiu.

Totes les solucions dels angles anteriors les hem trobat considerant que $c(y) \neq 0$. Així doncs hem d'estudiar les solucions quan $c(y) = 0$.

a) Solucions pel cas $c(y) = 0$.

Per identificar quan ens trobem en aquest cas serà quan y valgui 90° o bé 270. Això es tradueix quan la component a_x de la matriu de transformació sigui 1 en valor absolut. Substituïnt els valors de y a la matriu D.1 i igualant-la amb la D.2 es pot deduir que per aquest cas la solució és:

$$x = -\text{signe}(a_x) * z + \arctan 2(\frac{n_y * \text{signe}(a_x)}{\sigma_y}). \tag{D.11}$$

Veiem que la solució depèn del signe de a_x i a més existeixen infinites solucions ja que és una equació que depèn del paràmetre z.

Apèndix E

Manual d'usuari

E.1 Com utilitzar el manual

Aquest és un manual d'ús pràctic, per tal que usuaris amb escassos coneixements de planificació de moviments siguin capaços de crear i resoldre problemes d'aquest tipus amb el planificador desenvolupat en el present projecte. Per usuaris que desconeguin la funcionalitat de l'aplicació és recomanable que facin una lectura a la secció E.3, per la resta poden passar directament a les seccions E.2 i E.5.

La funcionalitat del planificador s'ha organitzat en quatre finestres, anomenades "Graus de llibertat", "Paràmetres", "Tasques", i "Anàlisi del graf". Descrivim el seu contingut tot seguit.

E.2 Inici

Per iniciar el planificador cal accedir primer a la carpeta *∼/prc* i cridar el planificador amb el nom del fitxer que descriu els robots i el seu entorn d'obstacles:

cd ∼/prc
iniprc nom

E.3 Descripció de les finestres

L'entorn de finestres s'ha implementat en *c* mitjançant la lliberia GTK+ [12]. Per tal d'inicialitzar i visualitzar aquest entorn cal arrencar Webots com s'ha dit anteriorment i prémer el botó "play".

E.3.1 Graus de llibertat

Aquesta finestra permet canviar els graus de llibertat associats següents elements modelitzats en Webots: les articulacions dels robots, la posició i orientació de l'objecte, la posició i orientació de les seves referències de prensió esquerra i dreta (secció D.1) i l'obertura de la pinça (figura E.1). Sense aquesta finestra només podríem moure els elements

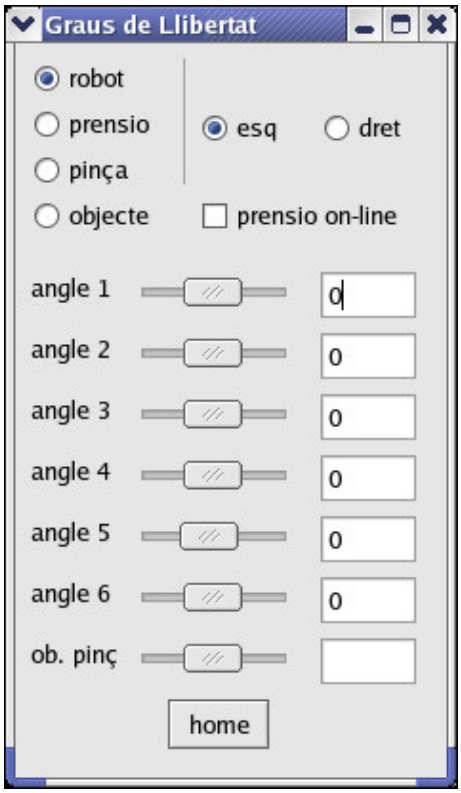


Figura E.1: Finestra de graus de llibertat.

mitjançant l'arbre d'escena de Webots[1] que resulta poc pràctica. S'ha pensat en un disseny compacte i simple a fi que puguem interectuar amb els elements, sense que aquesta finestra ens privi de veure els que estem movent. Gràcies a que tots els elements tenen sis graus de llibertat (exceptuant la pinça), només és necessari crear sis barres de desplaçament i una addicional per la pinça.

Per moure un element prèviament cal seleccionar-lo amb un dels següents botons: *robot*, *prensió*, *pinça* o *objecte*. Les opcions *esquerra* i *dreta* només afecten quan movem els robots i les referències de prensió. Un cop sel·leccionat l'element, l'usuari pot canviar-ne els valors dels seus graus de llibertat bé emprant les barres de desplaçament , o bé introduint valors en els camps d'entrada (quadres en blanc). Una de les opcions més interessants d'aquesta finestra és l'etiquetada com *prensio on-line*, que permet desplaçar l'objecte arrossegant-lo amb el ratolí, tot visualitzant com els robots l'agafen en cada moment. El planificador, a més, avisa si la configuració actual de l'objecte no és atansable pels robots. Per a que la seva selecció tingui efecte, a més d'estar seleccionat el botó *objecte* cal que s'hagi premut el botó *inicialitzar* de la barra de tasques.

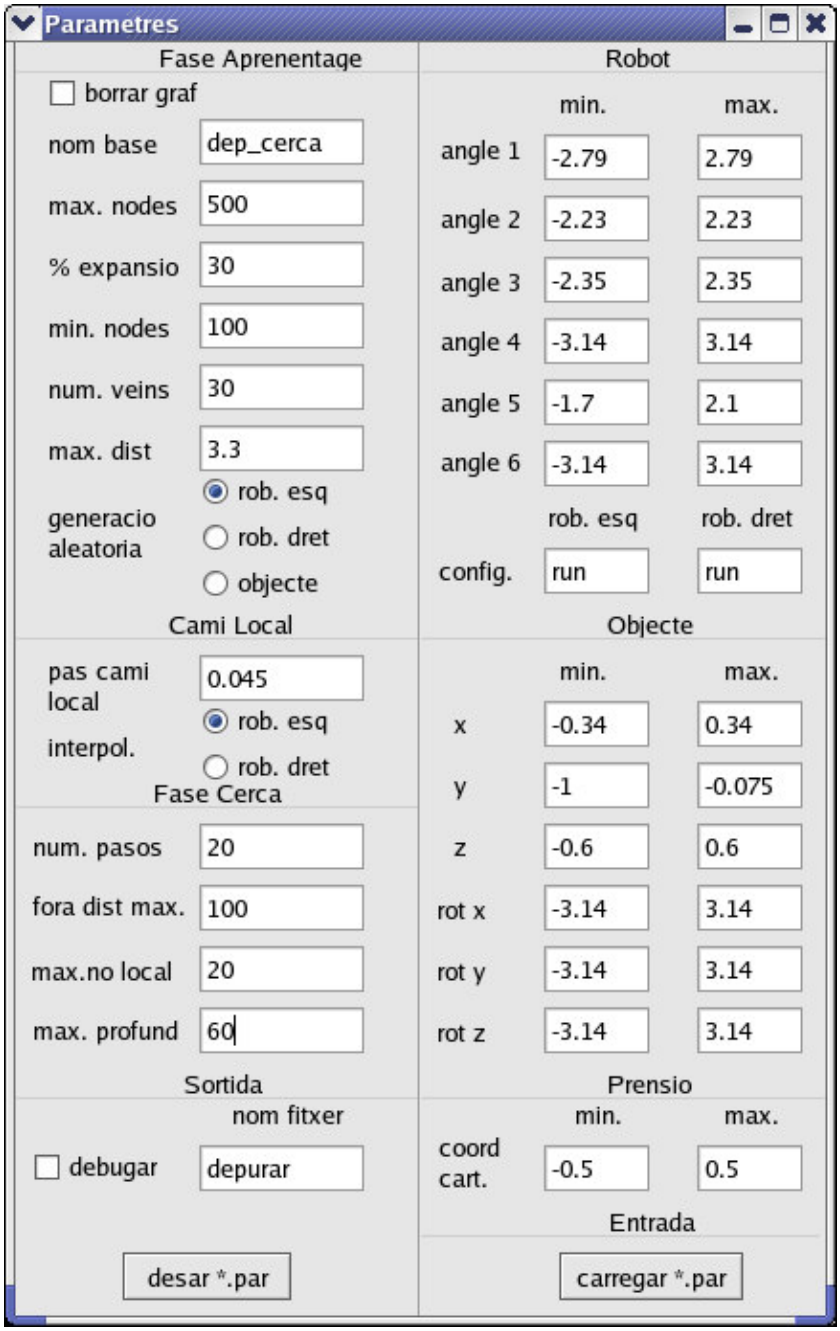


Figura E.2: Finestra de paràmetres.

E.3.2 Paràmetres

Aquesta finestra conté els paràmetres que controlen el comportament de l'algorisme de planificació, que són constants per un problema donat (figura E.2). Es descriuen a continuació, agrupats segons els set blocs principals de la finestra.

[1] Anomenat "scene tree"a la documentació de Webots [referència a documentació webots]

Fase d'aprenentatge:

- *nom base*: Nom base comú sense l'extensió que tindran tots els fitxers solució de la resolució d'un problema.
- *max. nodes*: Nombre de nodes que generem en la construcció d'un graf.
- *% d'expansió*: Percentatge de nodes que expansionarem respecte el nombre introduït en el camp d'entrada *max. nodes*.
- *min. nodes*: És el llindar mínim per defecte de l'opció *filtrar* de la barra d'anàlisi del graf, si no hem introduït cap valor en el camp *min*.
- *num. veïns*: Nombre màxim de nodes veïns amb els que s'intentarà conncectar cada nou node generat durant la fase de construcció.
- *generacio aleatòria*: Indica quin és l'element que mostregem aleatòriament per obtenir configuracions lliures.

Camí local:

- *pas camí local*: Pas d'avanç del planificador local.
- *interpol.*: Indica quin dels dos robots és l'actiu.

Fase cerca:

- *num.pasos:* Màxim nombre d'iteracions en generar una configuració aleatòria amb distància inferior a *max. dist* respecte la de partida.
- *fora max.dist*: Màxim nombre de configuracions aleatòries per iteració que poden caure fora de la bola de radi *max. dist* centrada en la configuració de partida.
- *max. no local*: Màxim nombre de fracassos del planificador local entre la configuració de partida i la generada aleatòriament a l'interior de la bola de radi *max. dist*.
- *max. profund*: Màxim nombre de nodes d'un camí per connectar un node de partida a un del graf.

Robot:

- El quadre etiquetat com Robot conté els rangs vàlids de treball de les articulacions dels braços RX60, i la configuració desitjada per la seva cinemàtica inversa, *config. rob. esq* i *config. rob. dret*.

Objecte:

- Pel quadre etiquetat com *Objecte* els rangs que es defineixen són únicament aplicables si l'objecte és l'element que generem aleatòriament.
- El quadre etiquetat com *Prensió* estableix el llindar màxim de les coordenades x, y i z de les referències de prensió respecte la central de l'objecte.

Sortida:

- *depurar*: Si marquem aquesta opció, en la construcció del graf es generarà un fitxer amb informació exhaustiva de la planificació efectuada.
- *desar *.par*: Desa tots els paràmetres en un fitxer amb el nom establert pel camp *nom_base*.

Entrada:

- *carregar *.par*: Carrega tots els paràmetres a partir d'un fitxer amb el nom establert pel camp *nom_base*.

E.3.3 Tasques

Aquesta finestra E.3 conté els botons d'accés a totes les tasques del planificador agrupades linealment en columnes d'acord amb les fases de que consta. A la part inferior de cada una tenim els botons per carregar i desar en fitxer les solucions obtingudes dels botons de la primera línea. Pels fixers **.map* i **.par* el seu nom ve definit pel camp *nom base*. Pels fitxers **.infi* el seu nom ve definit pel *nom base* més l'extensió d'aquest nom entrada en el camp *fitxer infi*. Per l'ultim tipus de fitxers **.trj* el seu nom és el nom base, més l'extensió del *fitxer infi* més l'extensió entrada en el camp *fitxer trj*. A continuació expliquem la funcionalitat dels diferents botons, agrupats per columnes.

Columna 1: Inicialitzacions

- *inicialitzar*: S'ha de prémer al principi de cada problema i quan desplacem les bases dels robots o bé les referències de prensió de l'objecte (secció D.1).

Columna 2: Fase d'aprenentatge

- *construir*: Construeix un mapa de rutes que conté configuracions lliures de la cadena tancada i trajectòries lliures entre parelles de configuracions (secció 4.1.1).
- *extendre*: Realitza l'expansió de $max.\ nodes \cdot \%expansio$ nodes del graf, amb l'objectiu de disminuir el nombre de components connexos obtinguts a l'etapa de construcció (secció 4.1.2).
- *carregar *.map*: Carrega el fitxer d'un mapa de rutes.

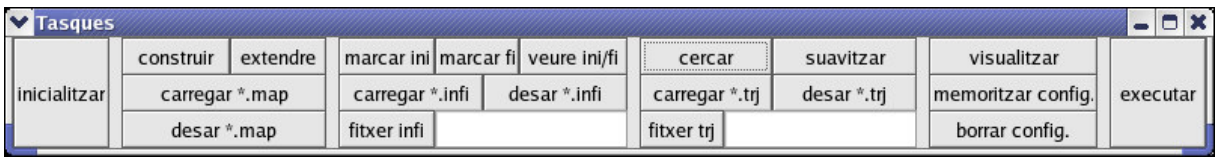


Figura E.3: Finestra de tasques.

- *desar *.map*: Desa en un fitxer el mapa obtingut de l'etapa de construcció o d'expansió.

Columna 3: Tria de les configuracions d'inici i final

- *marcar ini*: Emmagatzema la configuració actual com la inicial del problema a resoldre, sempre que sigui lliure.
- *marca fi*: Emmagatzema la configuració actual com la final del problema a resoldre, sempre que sigui lliure.
- *veure ini/fi*: Permet veure alternativament les configuracions inicial i final que s'han emmagatzemat.
- *carregar *.infi*: Carrega des d'un fitxer les configuracions inicial i final.
- *desar *.infi*: Desa en un fitxer les configuracions marcades com inicial i final.

Columna 4: Fase d'interrogació

- *cercar*: Intenta connectar les configuracions inicial i final al mapa, i en cas d'èxit busca el camí que les uneix (secció 4.2).
- *suavitzar*: Suavitza el camí trobat en l'etapa de cercar.
- *carregar *.trj*: Carrega d'un fitxer un camí solució.
- *desar *.trj*: Desa en un fitxer un camí solució obtingut en les etapes de cerca o de suavització.

Columna 5: Visualització de la solució.

- *visualitzar*: Genera la trajectòria a partir d'un camí solució que tenim en memòria.
- *memoritzar config.*: Permet memoritzar manualment una configuració lliure com un node d'un camí.
- *esborrar config.*: Borra l'últim node del camí que tenim en memòria.

Columna 5 : Solució real

- *executar*: Envia el fitxer que conté un camí solució als controladors dels braços RX60 per tal que realitzin la trajectòria real.

E.3.4 Anàlisi del graf

Aquesta finestra analitza els resultats del graf obtingut en la fase d'aprenentatge (figura E.4). Bàsicament els botons s'agrupen en dues categories, els que fan referència a un anàlisi purament visual (Visualització) i els que permeten analitzar la connectivitat del graf.

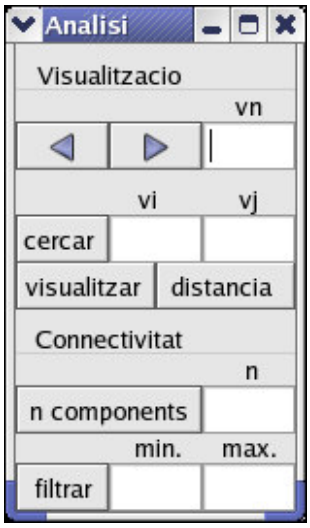


Figura E.4: Finestra d'anàlisi del graf.

Visualització :

- *Avançar*: Aquest botó visualitza les configuracions dels nodes del graf per índex creixent.
- *Retrocedir*: Visualitzem les configuració dels nodes del graf per índex decreixent.
- *Vn*: Visualitzem les configuracions del node enèssim del graf.
- *Cercar*: Si entrem dos índexs de nodes del graf *vi* i *vj* aquest botó permet trobar si existeix un camí entre ells.
- *Visualitzar*: Funcionament anàleg al seu homònim.
- *Distancia*: Troba la distància fina entre els dos nodes entrats en *vi* i *vj*

Connectivitat:

- *N components*: Mostra una taula dels n components connexos més grans, ordenats decreixentment, on el valor de n ha estat introduït en el camp d'entrada del costat.
- *Filtrar*: Elimina els components connexos amb un nombre de nodes inferior o igual al valor del camp *min.* i més grans o iguals que el camp *max.* Si no s'entra cap valor en *min.*, aquest pren el valor definit com *min. nodes* de la finestra de paràmetres. Per eliminar un determinat component connex s'ha d'entrar el seu nombre de nodes en *min.* i en *max.*

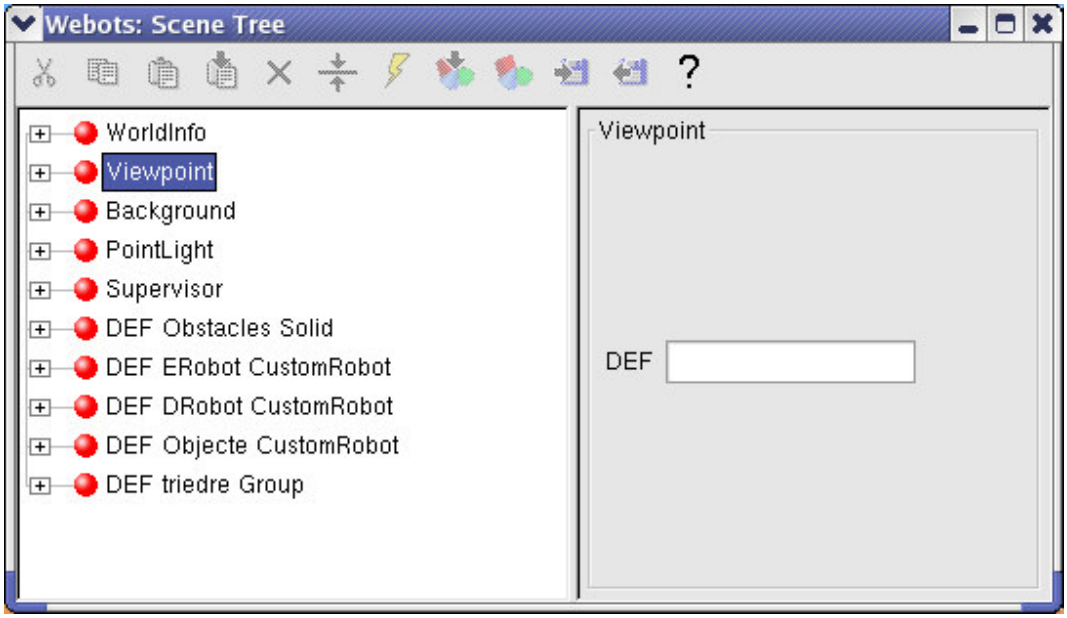


Figura E.5: Arbre d'escena de Webots per al problema base.

E.4 Guia per l'especificació d'un problema

Per especificar un problema cal representar en Webots els robots, l'objecte a transportar i els obstacles. En principi els primers apareixeran en tots els problemes que resoldrem, ja que tractem amb planificació de cadenes tancades. És per aquest motiu que treballarem amb un problema base, que presenta un arbre d'escena que sistematitza el procés d'inserció del obstacles i l'objecte, que són els únics elements que canviaran pels diferents problemes (figura E.5).

L'usuari ha de saber que els robots i l'objecte estan definits com a nodes CustomRobot perquè són els elements que movem de l'espai de treball i, en canvi, els obstacles com a nodes *Solid* perquè romanen estàtics en tota la simulació. Si no ho féssim així el detector de col·lisions no funcionaria adequadament.

E.4.1 Inserir els obstacles

En primer lloc obrim el problema base i el guardem amb un altre nom. Per inserir un obstacle primer seleccionem amb el cursor l'últim element de la figura E.5 i després triem l'opció "import node" que fa aparèixer una finestra que permet triar l'element que volem inserir com obstacle (d'entre els del directori Objects de webots). Un cop triat, aquest s'ha de tallar amb "cut" i enganxar-lo "paste" al camp children del node anomenat *obstacles interiors*. (figura E.6).

Per fer un bon ús del detector de col·lisions és important que cada node emmagatzemat en la carpeta *Objects* sigui del tipus *Solid* i que el seu camp boundingObject sigui un USE del *Shape* definit en el seu camp *children*. També podem tenir el cas que un node *Solid* contingui altres sòlids en els seu camp *children*. En aquest cas són aquests últims els que

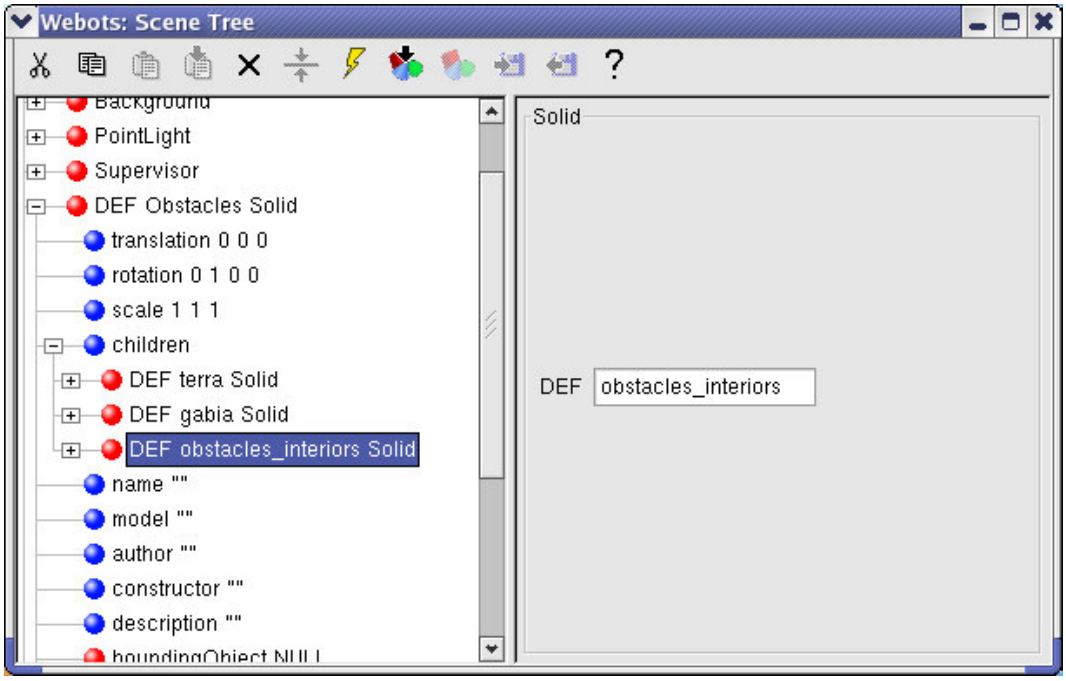


Figura E.6: Nivell de l'arbre d'escena on s'han d'inserir els obstacles. Els sòlids terra i gàbia són sòlids que com els robots estaran en tots els problemes i per tant mai els modificarem.

hauran de tenir el seu camp boundingObject omplert.

E.4.2 Inserir l'objecte

Per inserir l'objecte es realitza el mateix procés que per l'obstacle però ara el situem en el nivell que indica l'arbre d'escena de la figura E.7.

E.4.3 Actualitzar el fitxer de col·lisions

El detector de col·lisions només detecta nodes *Solid* que tinguin al seu camp *children* un únic *Shape* (mallat triangularment), i on el camp *boundingObject* utilitzi aquest Shape mitjançant un USE. Afegim també que aquests nodes *Solid* han de tenir un nom, ja que aquest s'utilitzarà en les crides del detector de col·lisions.

El fitxer de col·lisions consta de dues parts: en la primera es fa una declaració de totes les etiquetes dels nodes *Solid* susceptibles a col·lisions, i en la segona les parelles de sòlids pels que no volem verificar la seva col·lisió. Per exemple, quan tinguem un obstacle que està en contacte amb el terra, no voldrem detectar la col·lisió entre aquests dos.

Per crear un fitxer col·lisió el que farem serà obrir el l'anomenat *parelles_no_colisio.col* i guardar-lo amb el mateix nom triat de la secció E.4.1 però amb l'extensió col. Ara només caldrà obrir aquest fitxer i modificar aquelles parts que s'indiquen en els comentaris.

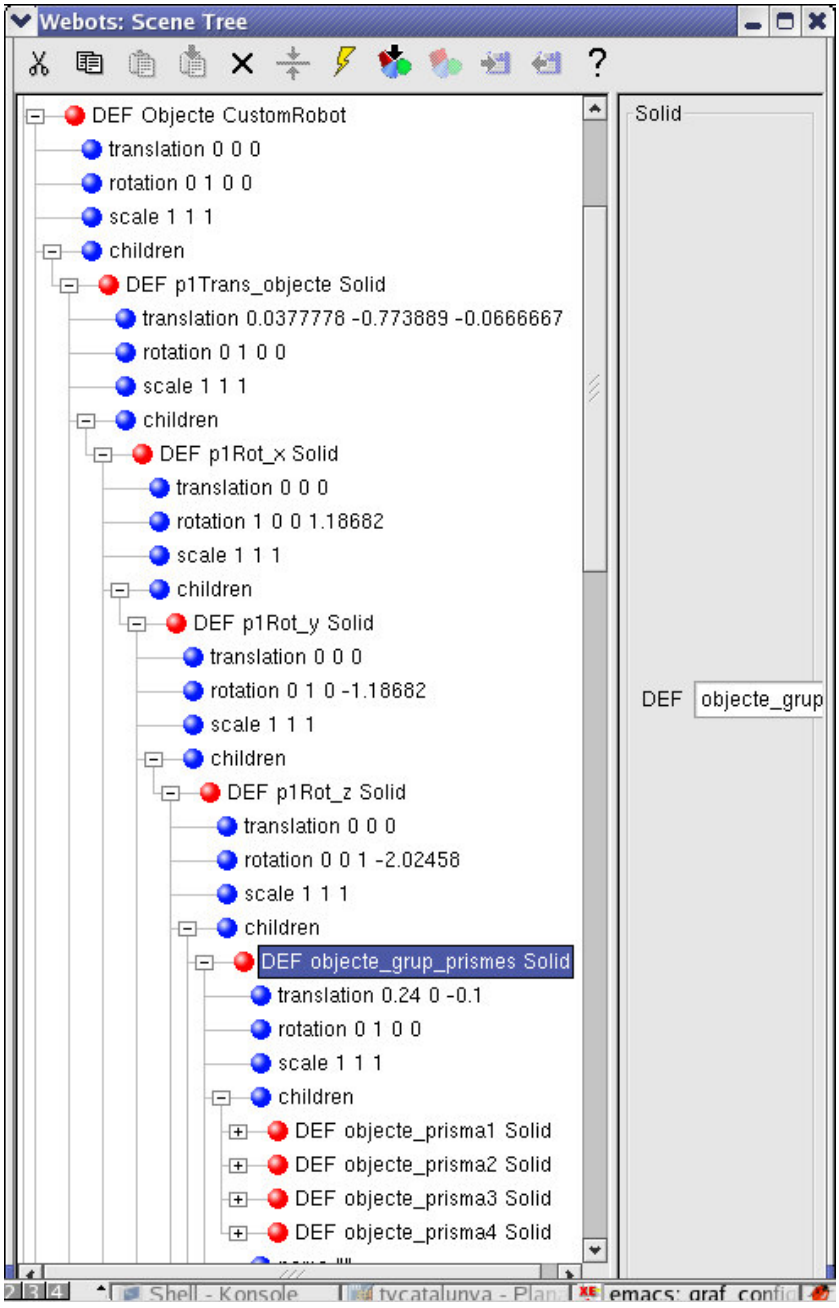


Figura E.7: El node ombrejat correspon a l'objecte que s'ha inserit, el qual esta format per uns quants sòlids en el seu camp *children*.

E.4.4 Fixar la posició i orientació dels robots

La posició i orientació dels robots les fixem a partir de l'arbre d'escena de Webots. Pel robot esquerre s'haurien de modificar els camps *translation* i *rotation* del node *ETransformacio_1* (figura E.8), per l'altre robot es faria anàlogament però pel node *DTransformacio_1*.

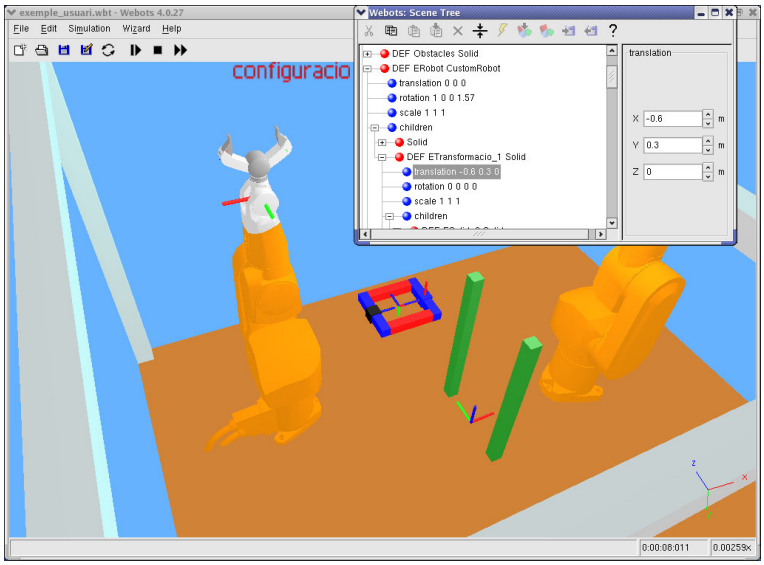


Figura E.8: Camps de posició i orientació de la base esquerra del robot respecte el món.

E.4.5 Fixar la prensió de l'objecte

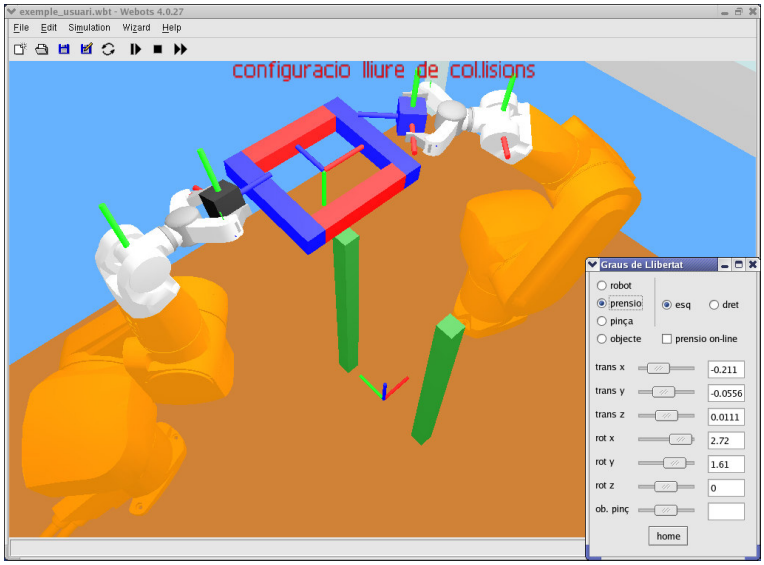


Figura E.9: L'exemple mostra que les pinces dels robots es situen en les referències de prensió esquerre i dreta de l'objecte amb independència de la forma que tingui aquest.

Mitjançant la finestra de graus de llibertat, seleccionant el botó *prensió* i marcant *prensió on-line*, podem definir la posició i orientació dels sistemes de referència esquerra i dret de l'objecte respecte el seu sistema de referència central (secció D.1). Aquesta posició i orientació és defineix a partir de les barres de desplaçament, o bé a partir dels camps d'entrada (figura E.9).

E.5 Guia per resoldre un problema

Un cop tenim l'arxiu wbt amb els robots, els obstacles i l'objecte incorporats, i el fitxer de col·lisions actualitzat, estem en condicions de resoldre un problema. Entrarem al programa Webots amb la comanda: *iniprc nom_problema*, on *nom_problema* fa referència als arxius *nom_problema.wbt* i *nom_problema.col*.

Cal esmentar que totes les utilitats del planificador estan en l'executable prc. Aquest ha d'estar en el camp *controllers* del node *Supervisor* de l'escena de treball. Per defecte ho estarà ja que hem fet modificacions del problema base que el contenia.

E.5.1 Definir els paràmetres

En primer lloc triem els valors dels paràmetres que volem (figura E.2). En els següents apartats, quan no s'indiqui de quina finestra és el botó esmentat, es suposarà que és la situada a la finestra de "Tasques" (figura E.3).

E.5.2 Executar la fase d'aprenentatge

Amb els paràmetres definits, després d'inicialitzar el problema, és el moment de prémer el botó *construir* (etapa de construcció). Quan finalitzi aquesta, mitjançant l'anàlisi del graf podem veure els components connexos que s'han format. Si n'hi ha molts i creiem que la solució n'hauria de tenir menys procedim a prémer *extendre* (etapa d'expansió) per intentar disminuir el seu nombre. Finalitzada aquesta etapa, podem aplicar el filtre de la finestra d'anàlisi (figura E.4) per eliminar aquells components que considerem inútils.

E.5.3 Triar les configuracions inicial i final

Les configuracions d'inici i final es trien a partir de l'opció *prensió on-line* de la finestra de graus de llibertat (secció E.3.1). Quan està seleccionada, i el problema està inicialitzat, movent l'objecte podem trobar-nos amb una de les tres possibilitats, indicades a les figures E.10, E.11 i E.12. Quan la configuració sigui lliure, aleshores la podrem emmagatzemar com a inicial o final prement el botó *marcar_ini* o *marcar_fi*. Notem que si la configuració no és lliure en prémer aquests botons ens sortirà un missatge indicant-ho i per tant no pot ser memoritzada.

E.5.4 Executar la interrogació

Un cop tenim una configuració d'inici i final emmagatzemades, podem buscar un camí lliure que uneixi aquestes dues a partir del mapa, prement el botó *cercar*. Si se'n troba un, aquest pot ser suavitzat prement el botó *suavitzar*.

E.5.5 Tipus de fitxers emprats

Tal com s'ha introduït a la secció E.3.3 tenim botons associats a la càrrega i desament de fitxers de les diferents solucions parcials del problema. Els tipus de fitxers que tractem són els següents:

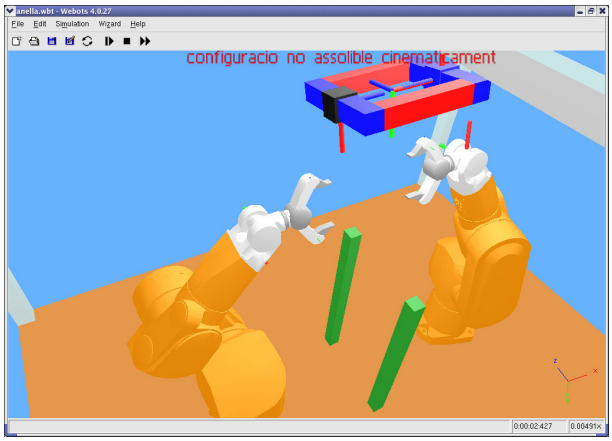


Figura E.10: Configuració no assolible cinemàticament.

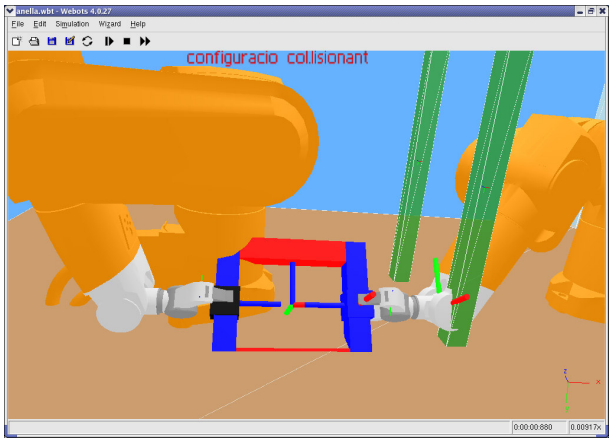


Figura E.11: Configuració que està col·lisionant.

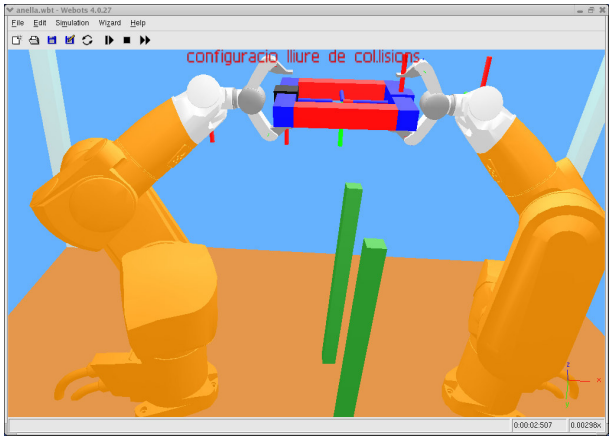


Figura E.12: Configuració lliure de col·lisions.

- **.par*: Aquest fitxer conté tots els paràmetres associats a un problema.
- **.map*: Aquest fitxer conté la informació del graf generat en la fase d'aprenentatge. Al principi del fitxer es presenta la informació de cada una de les configuracions lliures dels nodes (els angles del robot esquerre i dret, i la posició i orientació de l'objecte) i al final d'aquest les connexions entre els mateixos.
- **.infi*: Conté la informació de les configuracions d'inici i final.
- **.trj*: Conté la seqüència de configuracions lliures que determinen un camí que uneix la configuració inicial amb la final.
- **.dep*: Conté els resultats parcials de l'execució de l'etapa de construcció.

Apèndix F

Anàlisi descendent de l'algorisme

F.1 Introducció

Hem implementat l'algorisme probabilístic descrit als capítols 4 i 5, tot seguint un mètode jeràrquic, és a dir, per successius nivells de detall. A més ho hem fet de forma que el planificador sigui adaptable fàcilment al cas en que més de dos robots del tipus considerat estiguin treballant cooperativament, subjectant simultàniament l'objecte al llarg del seu recorregut.

A grans trets podem classificar el codi desenvolupat en tres grups de mòduls: un primer on s'implementa el mètode probabilístic, un segon que encapsula totes les rutines que interactuen Webots (capítol 6.1.1), fent que el planificador sigui en gran mesura independent del software emprat, i un tercer que constitueix la interfície gràfica del planificador desenvolupat.

En el pseudocodi que desenvoluparem s'han omès aspectes secundaris com l'anàlisi d'errors, la creació i alliberació de memòria, i les referències a punters, entre d'altres. A més, aquesta anàlisi no serà detallada, i es centrarà en parts rellevants del codi, que facilitin la comprensió del perquè hem escollit unes determinades entitats lògiques. El format d'implementació del pseudocodi ha seguit els criteris fixats pel departament de Llenguatges i Sistemes Informàtics de la UPC. En quan al llenguatge utilitzat per implementar el codi s'ha emprat el llenguatge *c* (secció H).

F.2 Pseudocodi del programa principal

Un primer nivell d'abstracció del mètode probabilístic el donen les fases de que consta:

1. **accio construccio_graf_config**
   (
   ent dades_config: DADES_CONFIG,
   ent/sort G: GRAF
)

2. **accio expansio_graf_config**
   (
   ent dades_config: DADES_CONFIG,
   ent/sort G: GRAF
)

3. **accio interrogacio_graf_config**
   (
   ent G: GRAF,
   ent config_ini: CONFIG,
   ent config_fi: CONFIG,
   ent dades_config: DADES_CONFIG,
   ent/sort trajectoria: GRAF
)

4. **accio suavitzacio_trajectoria_graf_config**
   (
   ent dades_config: DADES_CONFIG,
   ent trajectoria: GRAF
)

Totes aquestes accions actuen sobre el tipus GRAF, els nodes del qual contenen configuracions lliures del robot. Hom podria pensar que la darrera acció, que treballa sobre una trajectòria, no es pot encabir com una operació de GRAFS. De fet si ho podem fer ja que aquesta està formada per una seqüència de nodes que poden interpretar-se com una cadena de nodes unida per arestes.

Realitzarem l'anàlisi descendent per l'acció que representa l'etapa de construcció, per les restants s'ha seguit la mateixa metodologia (annex H). En primer lloc procedim a l'especificació de les estructures de dades de que consta. Per al tipus GRAF s'ha emprat una llista doblement enllaçada, que es caracteritza per la independència de les seves operacions respecte a la informació que dipositem en els seus nodes [10]. El lector interessat en l'especificació detallada d'aquesta estructura pot consultar els mòduls llr.h i glr.h. Per altra banda, DADES_CONFIG conté la informació que és constant al llarg d'una simulació i que és necessària per obtenir les configuracions lliures del robot. El motiu que no s'hagi inclòs com part de la informació d'una configuració serà discutit quan analitzem les operacions que treballen sobre aquest tipus. En resum, el primer nivell opera sobre les estructures de tipus GRAF i sobre configuracions del tipus CONFIG que emmagatzegarem en aquest.

La primera descripció de *construccio_graf_config()* no significa baixar de nivell jeràrquic, ja que les accions de que consta treballen sobre la mateixa estructura de GRAF. Aquesta serà una pràctica habitual del nostre disseny, subdividir un problema de certa envergadura en d'altres més petits que són resolts posteriorment. Pel que fa a aquesta acció, aquest procés de subdivisió finalitzarà quan les tasques més senzilles depenguin d'operacions que treballen sobre el tipus CONFIG, i/o d'operacions bàsiques del tipus GRAF.

```
accio construccio_graf_config (  ent dades_config: DADES_CONFIG,  )
                                  ent/sort G: GRAF
 1  vertex_c : enter
 2  si graf_buit(G)
 3     llavors
 4              vertex_c := 1   Inicialment el graf és buit.
 5     sino     vertex_c := ultim_vertex(G) + 1   Ampliació d'un graf existent.
 6
 7  mentre (vertex_c ≤ MAX_VERTEXS)
 8      fer
 9          crear_vertex_c_graf_config(dades_config, vertex_c, G)
10          connectar_vertex_c_graf_config(dades_config, vertex_c, G)
11          vertex_c := vertex_c + 1
12
13  calcular_pesos_graf_config(G)
```

Les accions de 2 i 5 són operacions bàsiques sobre el tipus GRAF i les de 9, 10 i 13 s'han de subdividir novament:

```
accio crear_vertex_c_lliure_graf_config (  ent dades_config: DADES_CONFIG,  )
                                           ent vertex_c: enter,
                                           ent/sort G:GRAF
1  config : CONFIG
2  canviar_a_configuracio_lliure(CERT, dades_config, config)
3  afegir_vertex(G, vertex_c, config)
```

En aquesta acció l'operació a la línia 2 actua sobre el tipus CONFIG i l'acció de la línia 3 és una operació primitiva del tipus GRAF. La implementació de *connectar_vertex_c_a_graf_config()* és encara una acció força complexa, que pot descomposar-se en moltes de més simples. Per simplificar la seva anàlisi mencionarem les accions de que depèn, i d'aquestes farem el mateix fins arribar a operacions que tractin amb el tipus CONFIG, que són les que interessen pel següent nivell d'anàlisi. El lector interessat en la implementació de les accions intermèdies de *connectar_vertex_c_a_graf_config()* pot consultar el codi font a l'annex H.

```
accio connectar_vertex_c_a_graf_config (  ent dades_config: DADES_CONFIG ,  )
                                          ent vertex_c: enter ,
                                          ent/sort G: GRAF
1  vertexs_veins : TAULA_VERTEXS_VEINS
2  pvertex_c : VERTEX
3  hi_ha_veins, colisionen : boolea
4  i : enter
5
6  pvertex_c := buscar_vertex_en_graf(G, vertex_c)
7  hi_ha_veins := cercar_veins_vertex_c_graf_config(dades_config,
```

```
                                                    pvertex_c,
                                                    G,
                                                    vertex_veins)
index_vei := 0
si hi_ha_veins
   llavors
        index_vei := cercar_primera_connexio_graf_config(dades_config,
                                                         pvertex_c,
                                                         vertex_veins,
                                                         G)
        si no_darrer_vei(vertex_veins)
           llavors
                afegir_aresta(G, vertex_c, vertex_veins[i])
                marcar_connexes_a_vertex_graf_config(vertex_veins[i])
                i := i + 1
        mentre (i < nombre_veins(vertex_veins))
            fer si (¬(connexio_vertex_c_vertex_n(pvertex_c,
                                                 vertex_veins[i]))∧
                (planificador_local_graf_config(CERT,
                                                 dades_config,
                                                 pvertex_c,
                                                 vertex_veins[i])))
                llavors
                    colisionen := obtenir_colisio()
                    incrementar_crides_i_fallades_entre_vertexs(colisionen,
                                                                pvertex_c,
                                                                vertex_veins[i])
                    marcar_connexes_a_vertex_graf_config(vertex_veins[i])
                    afegir_aresta(G, vertex_c, vertex_veins[i])

                i := i + 1
```

En el codi anterior TAULA_VERTEX_VEINS és l'estructura que conté els vèrtexs que són veïns a un de donat (secció 4.1.1). La seva definició i operacions associades pot trobar-se en el mòdul *vei.h* de l'annex H. Les accions de que depèn *connectar_vertex_c_graf_config()* són:

1. cercar_veins_vertex_c_graf_config(): Es subdivideix novament.

2. cercar_primera_connexio_graf_config(): Es subdivideix novament.

3. marcar_connexes_a_vertex_graf_config(): Depèn d'operacions del tipus GRAF.

4. connexio_vertex_c_vertex_n() : Depèn d'operacions del tipus GRAF.

5. planificador_local_graf_config() : Depèn d'operacions del tipus CONFIG.

6. incrementar_crides_i_fallades_entre_vertexs(): Depèn d'operacions del tipus CONFIG.

Si tornem a subdividir 1 i 2 tenim:

1. cercar_veins_vertex_c_graf_config()

- comparar_solucions_vertex_graf_config(): Depèn d'operacions del tipus GRAF.
- distancia_entre_vertexs_graf_config(): Depèn d'operacions del tipus CONFIG.
- distancia_fina_entre_vertexs(): Depèn d'operacions del tipus CONFIG.

2. cercar_primera_connexio_graf_config()

- planificador_local_graf_config(): Depèn d'operacions del tipus CONFIG.

Així doncs les úniques accions de *connectar_vertex_c_a_graf_config()* on la implementació de les quals depenen del tipus CONFIG són les següents:

boolea planificador_local_graf_config (ent detectar_colisio: boolea , ent dades_config: DADES_CONFIG , ent pvertex_c: GRAF , ent pvertex_n: GRAF)

```
1  config_c : CONFIG
2  config_n : CONFIG
3  planificador_local : boolea
4
5  config_c := accedir_configuracio_vertex(pvertex_c)
6  config_n := accedir_configuracio_vertex(pvertex_n)
7  planificador_local := planificador_local_configi_configf
8  (detectar_colisio, dades_config, config_c, config_n)
9  retorna planificador_local
```

boolea distancia_entre_vertexs_graf_config (ent pvertex_c: GRAF , ent pvertex_n: GRAF)

```
1  config_c : CONFIG
2  config_n : CONFIG
3  distancia : real
4
5  config_c := accedir_configuracio_vertex(pvertex_c)
6  config_n := accedir_configuracio_vertex(pvertex_n)
7  distancia := distancia_entre_configuracions(config_c, config_n)
8  retorna distancia
```

boolea distancia_fina_entre_vertexs_graf_config (ent pvertex_c: GRAF ,)
ent pvertex_n: GRAF

1 $config\_c : CONFIG$
2 $config\_n : CONFIG$
3 $distancia : real$
4
5 $config\_c := accedir\_configuracio\_vertex(pvertex\_c)$
6 $config\_n := accedir\_configuracio\_vertex(pvertex\_n)$
7 $distancia := distancia\_fina\_entre\_configuracions(config\_c, config\_n)$
8 **retorna** *distancia*

boolea comparar_solucions_vertexs_graf_config (ent pvertex_c: GRAF ,)
ent pvertex_n: GRAF

1 $config\_c : CONFIG$
2 $config\_n : CONFIG$
3 $igual\_solucio : boolea$
4
5 $config\_c := accedir\_configuracio\_vertex(pvertex\_c)$
6 $config\_n := accedir\_configuracio\_vertex(pvertex\_n)$
7 $igual\_solucio := comparar\_solucions\_configuracions(config\_c, config\_n)$
8 **retorna** *igual_solucio*

De les operacions del tipus CONFIG, només descriurem *canviar_a_configuracio_lliure()* i *planificador_local_graf_config()* perquè són les més rellevants per seguir amb l'anàlisi descendent que estem realitzant. En el present projecte una configuració quan els braços robòtics junt amb l'objecte formen una cadena cinemàtica tancada. Per tant, hem definit CONFIG com la següent tupla:

CONFIG=**tupla**
robote: CONF_ROBOT
robotd:CONF_ROBOT
obj: CONF_OBJECTE
f: enter
n: enter
w: real
ftupla,

on CONF_ROBOT conté els angles de la configuració d'un robot i el tipus de solució cinemàtica, i CONF_OBJECTE la posició i orientació de l'objecte a transportar. Els enters f i n són el nombre de crides fallades i totals del planificador local associades a la configuració. El pes w d'aquesta es calcularà a partir de f i n finalitzada la fase de construcció (seccio 4.1.2) mitjançant la rutina *calcular_pesos_graf_config()*. Tal com s'ha definit el tipus CONFIG, hom s'adona que l'hem creat com agrupació d'entitats més simples, el tipus CONF_ROBOT i el CONF_OBJECTE, que fan que les implementacions

de les operacions del tipus CONFIG siguin independents de com s'hagin definit aquestes dues estructures.

És el moment d'especificar el tipus DADES_CONFIG tal com havíem anunciat al principi d'aquesta secció. El definim com una tupla amb els següents camps:

tau_carac:taula[MAX_CADENA] de caracter
DADES_CONFIG=**tupla**
elem_aleat: tau_car
elem_interp: tau_car
pdh: PDH
jr: JNTS_RANGE
T: TRANSF_CTNT
conf_rob_esq: tau_car
conf_rob_dret: tau_car
ftupla

Molt breument, elem_aleat indica quin és l'element que mostregem aleatòriament i elem_interp quin és el robot actiu quan fem la crida al planificador local (secció 5.3.2). PDH és una estructura que conté els paràmetres de Denavit-Hartenberg del braç robòtic i jr els rangs de treball de les seves articulacions, així com els *ofssets* associats (secció B.4). Aquestes dues últimes estructures de dades estan definides en *Gtypes.h*, on el lector pot veure la seva definició. El tipus TRANSF_CTNT és una estructura que conté les següents matrius de transformació constants: les que posicionen i orienten cada una de les bases dels braços robòtics respecte el món (secció 5.4.2.1), i les que relacionen les referències dels extrems de l'objecte amb la seva central (secció D.1). Per últim, conf_rob_esq i conf_rob_dret contenen el tipus de solució que triem per la cinemàtica inversa pel robot que treballi com a passiu (seccio 5.3.2 i solucions de C.52). La informació d'aquesta estructura és constant per un problema donat, i no l'hem agrupat com un camp de CONFIG perquè aleshores l'hauríem d'haver recalculat per a cada configuració nova que creéssim, i això hauria alentit l'etapa de construcció. L'altre opció era definir DADES_CONFIG com una variable global, però creiem que conté massa informació com per que no aparegui en les capçaleres de les accions que l'utilitzen. Per tant la inicialitzem al principi del problema i després la passem com a paràmetre de les accions que la fan servir. Conegudes les especificacions dels tipus CONFIG i DADES_CONFIG podem fer la implementació de les operacions més rellevants que treballen sobre el tipus CONFIG:

boolea canviar_a_configuracio_lliure (ent detectar_colisions: boolea ,
ent dades_config: DADES_CONFIG ,
ent/sort config: CONFIG)

```
1 robot : CONF_ROBOT
2 objecte : CONF_OBJECTE
3 conf_valida : boolea
4 distancia : real
5 si es_objecte(dades_config.elem_aleat)
6    llavors
```

```
7              fer
8                  posicio_objecte_aleatori(objecte)
9                  conf_valida := canviar_a_configuracio_donat_objecte(detectar_colisions,
10                                                                     dades_config,
11                                                                     objecte,
12                                                                     config)
13             fins ¬ conf_valida
14     sino    fer
15                 angles_robot_aleatori(robot)
16                 conf_valida := canviar_a_configuracio_donat_robot(detectar_colisions,
17                                                                   dades_config.elem_aleat,
18                                                                   dades_config,
19                                                                   robot,
20                                                                   config)
21             fins ¬ conf_valida
```

Hom pot veure que una configuració lliure es pot obtenir o bé mostrejant els graus de llibertat de l'objecte o bé els d'un robot. Les accions *canviar_a_configuracio_donat_objecte()* i *canviar_a_configuracio_donat_robot()* són operacions del tipus configuració, en canvi *posicio_objecte_aleatori(objecte)* i *angles_robot_aleatori(robot)* treballen amb el tipus objecte i robot respectivament. A continuació, per simplificar, suposarem que mostregem el robot esquerre, i per tant implentarem l'acció *canviar_a_configuracio_donat_robot()*.

boolea canviar_a_configuracio_donat_robot (ent detectar_colisions: boolea ,
ent robot_actiu: tau_car ,
ent dades_config: DADES_CONFIG ,
ent robot: CONF_ROBOT ,
ent/sort config: CONFIG)

```
1  conf_valida, conf_valida_cine : boolea
2  colisionen : boolea
3
4  Trobe, Trobd, Tobj, P1 : TRANSF_HOMOG
5  tipus_conf : tau_car
6
7  si es_robot_esquerra(robot_actiu)
8     llavors
9              copiar_robot_a_nou_robot(robot, acces_robot_esq(config))
10             conf_valida_cine := angles_robot_ref_RX60_en_rang
11             (robot, dades_config.jr)
12
13             si conf_valida_cine
14                llavors
15                       angles_robot_ref_RX60_ref_paper(config)
16                       cinematica_directa_RX60(dades_config.pdh,
```

```
17                                          acces_robot_esq(config),
18                                          Trobe,
19                                          tipus_conf)
20                          producte_transf_homogenies(dades_config.T.ba_T_b0e,
21                                          Trobe,
22                                          P1)
23                          producte_transf_homogenies(P1,
24                                          dades_config.T.graspe_inv,
25                                          Tobj)
26                          producte_transf_homogenies(dades_config.T.b0d_T_ba,
27                                          Tobj,
28                                          P1)
29                          producte_transf_homogenies(P1,
30                                          dades_config.T.graspd,
31                                          Trobd)
32                          conf_valida_cind := cinematica_inversa_RX60(dades_config.pdh,
33                                                      dades_config.jr,
34                                                      Trobd,
35                                          dades_config.conf_rob_dret,
36                                          acces_robot_dret(config))
37                          si conf_valida_cind
38                            llavors cinematica_inversa_objecte(Tobj, acces_objecte(config))
39
40                                  angles_robot_ref_paper_ref_RX60
41                                  (acces_robot_esq(config))
42                                  angles_robot_ref_paper_ref_RX60
43                                  (acces_robot_dret(config))
44
45                                  si detectar_colisions
46                                    llavors
47                                          inicialitzar_colisio()
48
49                                          canviar_a_configuracio_predeterminada
50                                          (config)
51                                          avancar_simulacio()
52                                          colisionen := obtenir_colisio()
53
54  retorna ((conf_valida_cine ∧ conf_valida_cind) ∧ (¬ colisionen))
```

La implementació d'aquesta acció depèn bàsicament dels tipus CONF_OBJECTE i CONF_ROBOT que seran els següents nivells d'anàlisi. També s'ha de destacar les operacions relacionades amb transformacions homogènies de les línies 19 a 28, implementades en el mòdul *transf_homogenies.h*. De 48 veiem que tenim una acció que depèn del tipus CONFIG que la implementem a continuació.

accio canviar_a_configuracio_predeterminada (ent config: CONFIG)
1 $canviar\_angles\_robot(ROBOT\_ESQUERRA, acces\_robot\_esq(config))$
2 $canviar\_angles\_robot(ROBOT\_DRET, acces\_robot\_dret(config))$
3 $canviar\_posicio\_objecte(acces\_objecte(config))$

Veiem que depèn d'operacions que treballen sobre el tipus CONF_ROBOT i CONF_OBJECTE. Abans de procedir amb l'especificació del tipus CONF_OBJECTE i CONF_ROBOT queda implementar l'acció que realitza la planificació local, i que era una operació sobre el tipus CONFIG:

boolea planificador_local_configi_configf (ent detectar_colisions: boolea ,)
ent dades_config: DADES_CONFIG ,
ent configi: CONFIG ,
ent configf: CONFIG ,
ent robot: CONF_ROBOT ,

1 $vector\_ini : VECTOR$
2 $vector : VECTOR$
3 $vector\_resta : VECTOR$
4 $vector\_landa : VECTOR$
5 $n, f : enter$
6 $conf\_valida : boolea$
7 $robot\_actiu : tau\_car$
8 $planificador\_local : boolea$
9 $config\_act : CONFIG$
10 $robot : CONF\_ROBOT$
11
12 **si** $robot\_actiu\_es\_esq(dades\_config.elem\_interp)$
13 **llavors** $convertir\_ROBOT\_VECTOR(acces\_robot\_esq(configi), vector\_ini)$
14 $convertir\_ROBOT\_VECTOR(acces\_robot\_esq(configf), vector)$
15
16 **sino** $convertir\_ROBOT\_VECTOR(acces\_robot\_dret(configi), vector\_ini)$
17 $convertir\_ROBOT\_VECTOR(acces\_robot\_dret(configf), vector)$
18
19 $vector\_resta := restar\_vectors(vector\_ini, vector)$
20 $vector\_landa := vector\_long\_landa(vector\_resta, PAS\_AVANÇ)$
21 $nf := ultima\_iteracio(vector\_ini, vector, PAS\_AVANÇ)$
22
23 $n := 1$
24 $conf\_valida := CERT$
25 **mentre** $((n \leq nf) \wedge conf\_valida)$
26 **fer**
27 $vector := vector\_enessim(n, vector\_ini, vector\_landa)$
28 $convertir\_VECTOR\_ROBOT(vector, robot)$
29 $conf\_valida := canviar\_a\_configuracio\_donat\_robot$
30 $(detectar\_colisio, dades\_config.elem\_interp, dades\_config, robot, config\_act)$

31 $n := n + 1$
32
33 **retorna** *conf_valida*

La darrera acció depèn bàsicament d'operacions que treballen amb el tipus VECTOR. Ho hem fet així, perquè en l'estratègia de resolució del planificador local tenia un caràcter vectorial (secció 5.4.2.2). Les línies 12 a 17 traspassen la informació del robot actiu de les configuracions d'inici i final a un vector d'inici i un de final. Després tot el tractament el realitzem amb operacions VECTOR, i només quan necessitem calcular les configuracions traspassem la informació del vector actual a una variable tipus CONF_ROBOT (línia 28). Cal destacar també, que la creació de *canviar_a_configuració_donat_robot()* s'ha fet per tal de simplifcar el codi, ja que veiem que és emprada tant en l'acció *canviar_a_configuracio_lliure()* com en aquesta darrera. Ja només ens queda definir com hem definit els tipus CONF_OBJECTE, CONF_ROBOT i VECTOR:

tau_artic:taula[1···6] de real
tau_cars:taula[1···4]de caracter
CONF_ROBOT=**tupla**
ARTIC: tau_artic
sol: tau_cars
ftupla

tau_comp:taula[1···3] de real
CONF_OBJECTE=**tupla**
transl: tau_comp
rotacio: tau_comp
ftupla

tau_vector:taula[1···MAX_LONG] de real
VECTOR=**tupla**
vector: tau_vector
long: enter
ftupla

Apèndix G

Documentació dels mòduls

G.1 Introducció

En aquest annex donem una documentació detallada pels mòduls que considerem principals i pels secundaris fem un resum del seu contingut. Cal senyalar que en la documentació detallada només descrivim les capçaleres de les rutines públiques (les incloses als fitxers *.h) i els seus paràmetres associats. El lector interessat en la implementació de les rutines públiques i privades pot consultar al codi font que s'ha realitzat amb llenguatge *c*.

G.2 Mòduls principals

G.2.1 graf_config.h

Conté les rutines que treballen amb un graf de configuracions.

```
#include "vei.h"
#include "vertexs_exp.h"
#include "component.h"
#include "globals.h"
#include "configuracio.h"
#include "traject.h"
#include "graf.h"
#include "wgraph.h"
#include "tau_list.h"
```

Definicions

- #define **CONFIG_VERTEX**(V) ((CONFIG∗)(((vertex_data∗)DATA(V)) → specific_data))

Funcions

- void **construccio_graf_config** (DADES_CONFIG ∗dades_config, graph ∗p_G)

  Realitza l'etapa de construcció.

- void **expansio_graf_config** (DADES_CONFIG *dades_config, graph *p_G)

  Realiza la fase d'expansió.

- void **eliminar_components_graf_config** (graph *p_G, int filtrar_min, int filtrar_max)

  Filtra aquells components residuals.

- void **interrogacio_graf_config** (graph *p_G, CONFIG *config_ini, CONFIG *config_fi, DADES_CONFIG *dades_config, list *ptraject)

  Realitza la fase d'interrogació.

- void **suavitzacio_trajectoria** (DADES_CONFIG *dades_config, list *p_traject)

  Realitza la suavització d'una trajectoria.

- void **visualitzar_trajectoria** (DADES_CONFIG *dades_config, list traject)

  Visualitzem una trajectòria emmagatzemada.

- void **cercar_cami_minim_lliure** (DADES_CONFIG *dades_config, list *ptraject)

  Cerca el camí mínim d'una trajectòria.

- int **buscar_components_connexos_del_graf_config** (graph G, COMPONENT *taula_components)

  Busca el nombre de components connexos del graf.

Documentació de les Funcions

Autor:

Gorka Bonals

Versió:

1.0

Data:

Agost 2005

Descripició

En aquest mòdul creem un graf de configuracions amb el mètode que anomenem mapa probabilístic. Les principals funcions públiques constitueixen les diferents fases de que consta el mètode.

- **int buscar_components_connexos_del_graf_config (graph *G*, COMPONENT * *taula_components*)**

  Busca el nombre de components connexos del graf.

Paràmetres:

G Par. ent. Graf de configuracions.

taula_components Par. ent/sort. Taula que té la informació que volem escriure: nombre de nodes per component i un node representatiu de cada un d'ells.

Retorna:

Un enter que és el nombre de components connexos.

- **void cercar_cami_minim_lliure (DADES_CONFIG * *dades_config*, list * *ptraject*)**

  Crea un graf de la trajectòria, i d'aquest busca el camí mínim entre el node inicial i final de la trajectòria. La trajectòria mínima resultant haurà de ser lliure de col.lisions.

  Paràmetres:

  dades_config Par. ent. Dades constants per la generació i connexió de les configuracions.

  ptraject Par. ent/sort. Conté la trajectòria com una seqüència de configuracions.

- **void construccio_graf_config (DADES_CONFIG * *dades_config*, graph * *p_G*)**

  Aquesta rutina realitza l'etapa de construcció del mapa probabilístic. Consisteix en la generació de configuracions lliures que s'emmagatzemen com nodes d'un graf. Les arestes entre aquests nodes indiquen que hi ha una trajectòria lliure entre elles.

  Paràmetres:

  dades_config Par. ent. Dades constants per la generació i connexió de les configuracions.

  p_G Par. ent/sort. Llista doblement enllaçada que representa el graf que creem.

- **void eliminar_components_graf_config (graph * *p_G*, int *filtrar_min*, int *filtrar_max*)**

  Filtra del graf aquells components amb un número de nodes que estan per sota i per sobre d'un llindar. Quan els llindars són idèntics elimina els/el comonent que té/tenen aquell número de nodes.

  Paràmetres:

  p_G Par. ent/sort. Graf de configuracions.

  filtrar_min Par. ent. Nombre enter pel llindar mínim.

  filtrar_max Par. ent. Nombre enter pel llindar màxim.

- **void expansio_graf_config (DADES_CONFIG * *dades_config*, graph * *p_G*)**

  Realitza l'etapa d'expansió del mètode probabilístic, que expandeix aquells nodes que estan en regions de gran dificultat.

Paràmetres:

dades_config Par. ent. Dades constants per la generació i connexió de les configuracions.

p_G Par. ent/sort. Graf de configuracions que creem.

- **void interrogacio_graf_config (graph * *p_G*, CONFIG * *config_ini*, CONFIG * *config_fi*, DADES_CONFIG * *dades_config*, list * *ptraject*)**

  Aquí es realitza l'etapa d'interrogació, la qual connecta dues configuracions donades al graf. Si existeix una seqüència de nodes que uneix les configuracions donades, aquesta s'emmagatzema en una llista (list).

  Paràmetres:

  p_G Par. ent. Graf de configuracions

  config_ini Par. ent. La configuració inicial.

  config_fi Par. ent. La configuració final.

  dades_config Par. ent. Dades constants per la generació i connexió de les configuracions.

  ptraject Par. ent/sort. Conté la trajectòria com una seqüència de configuracions.

- **void suavitzacio_trajectoria (DADES_CONFIG * *dades_config*, list * *p_traject*)**

  Suavització de la trajectòria pel mètode de l'optimitzador de la poligonal.

  Paràmetres:

  dades_config Par. ent. Dades constants per la generació i connexió de les configuracions.

  p_traject Par. ent/sort. Conté la trajectòria com una seqüència de configuracions.

- **void visualitzar_trajectoria (DADES_CONFIG * *dades_config*, list *traject*)**

  Visualitza una trajectòria com la concatenació de les trajectòries entre nodes adjacents. Aquestes últimes es calculen mitjançant el planificador local.

  Paràmetres:

  dades_config Par. ent. Dades associades a les configuracions.

  traject Par. ent. Conté la trajectòria com una seqüència de configuracions.

G.2.2 configuracio.h

Les rutines que treballen amb configuracions.

```
#include <string.h>
#include "globals.h"
#include "transf_ctnt.h"
#include "vector.h"
#include "objecte.h"
#include "robots.h"
#include "colisio.h"
```

Estructures de Dades

struct struct_planif_local

- bool **solucio**
- int **n**
- **CONFIG** ∗ **config_ant**

struct configu

- CONF_ROBOT ∗ **robote**
- CONF_ROBOT ∗ **robotd**
- CONF_OBJECTE ∗ **obj**
- int **f**
- int **n**
- real **w**

struct dades_config

- char **elem_aleat** [MAX_CADENA]
- char **elem_interp** [MAX_CADENA]
- PDH **pdh**
- JNTS_RANGE **jr**
- TRANSF_CTNT **T**
- char **conf_rob_esq** [4]
- char **conf_rob_dret** [4]

Definicions

- #define **ROBOTE**(config) ((config) → robote)
- #define **ROBOTD**(config) ((config) → robotd)
- #define **OBJECTE**(config) ((config) → obj)
- #define **N_PLAN_LOC**(config) ((config) → n)
- #define **F_PLAN_LOC**(config) ((config) → f)
- #define **PES**(config) ((config) → w)

- #define **ARTICE**(config) (ARTIC(ROBOTE(config)))
- #define **ARTICD**(config) (ARTIC(ROBOTD(config)))
- #define **ARTICE_i**(config, i) (ARTIC_i((ROBOTE(config)),i))
- #define **ARTICD_i**(config, i) (ARTIC_i((ROBOTD(config)),i))
- #define **OBJ_TRANSL_i**(config, i) TRANSL_i((OBJECTE(config)),i)
- #define **OBJ_ROT_i**(config, i) ROT_i((OBJECTE(config)),i)
- #define **SOLUCE**(config) (SOLUC((ROBOTE(config))))
- #define **SOLUCD**(config) (SOLUC((ROBOTD(config))))

Definicions de Tipus

- typedef **configu CONFIG**
- typedef **dades_config DADES_CONFIG**
- typedef **struct_planif_local INFO_PLANIF_LOCAL**

Funcions

- **CONFIG** ∗ **crear_configuracio** ()

  Creació d'una configuració.

- void **alliberar_configuracio** (**CONFIG** ∗∗config)

  Alliberació d'una configuració.

- void **canviar_a_configuracio_predeterminada** (**CONFIG** ∗config)

  Visualitzar una configuració.

- void **obtenir_configuracio_actual** ()

  Guarda la configuració actual.

- void **copiar_obtenir_config_a_configuracio** (**CONFIG** ∗config)

  Copia la configuració a una altre variable.

- bool **canviar_a_configuracio_on_line** (**DADES_CONFIG** ∗**dades_config**, **CONFIG** ∗config)

  Intenta tancar la cadena movent l'objecte.

- void **canviar_a_configuracio_lliure** (bool detectar_colisions, **DADES_CONFIG** ∗**dades_config**, **CONFIG** ∗config)

  Troba aleatòriament una configuració lliure.

- void **copiar_config_a_nova_config** (**CONFIG** ∗config, **CONFIG** ∗nconfig)

  Traspàs d'una configuració a una altra.

- **CONFIG** ∗ **generar_configuracio_previa_colisio** (**DADES_CONFIG** ∗**dades_config**, **CONFIG** ∗config_act)

  Troba una configuració propera a un obstacle.

- **CONFIG** ∗ **generar_configuracio_robot_aleatori_en_bola_DIST_MAX** (**DADES_CONFIG** ∗**dades_config**, **CONFIG** ∗config_act)

  Genera una configuració lliure dintre d'una bola de distància.

- void **escriure_configuracio** (**CONFIG** ∗config)

  Escriu per pantalla el contingut d'una configuració.

- void **desar_configuracio** (FILE ∗fa, **CONFIG** ∗config)

  Desa una configuració en un fitxer de text.

- void **carregar_configuracio** (FILE ∗fa, **CONFIG** ∗config)

  Carrega en memòria una configuració des d'un fitxer de text.

- **INFO_PLANIF_LOCAL** ∗ **crear_info_planificador_local** ()

  Crea un punter al que retorna el planificador local.

- void **alliberar_info_planificador_local** (**INFO_PLANIF_LOCAL** ∗∗planificador_local)

  Allibera una variable del tipus INFO_PLANIF_LOCAL.

- **INFO_PLANIF_LOCAL** ∗ **planificador_local_configi_configf** (bool detectar_colisio, **DADES_CONFIG** ∗**dades_config**, **CONFIG** ∗configi, **CONFIG** ∗configf)

  Crida del planificador local entre dues configuracions.

- real **distancia_entre_configuracions** (**CONFIG** ∗config1, **CONFIG** ∗config2)

  Distància euclídia angular entre dues configuracions.

- real **distancia_fina_entre_configuracions** (**DADES_CONFIG** ∗**dades_config**, **CONFIG** ∗configi, **CONFIG** ∗configf)

  Distància, com a longitud recorreguda pel planificador local.

- bool **comparar_tipus_solucions_configuracions** (**CONFIG** ∗config1, **CONFIG** ∗config2)

  Compara les solucions del robot passiu.

Documentació de les Funcions

Autor:
Gorka Bonals Sastre

Versió:
1.0

Data:
Agost 2005

Descripició
Implementem les rutines que treballen amb el tipus CONFIG, que constitueixen les configuracions del sistema a planificar. Pel present projecte, una configuració és la cadena cinemàtica tancada que formen els dos robots RX60 quan transporten un objecte subjectat amb les seves pinces.

- **void alliberar_configuracio (CONFIG ** *config*)**

  Allibera memòria per una configuració, que consta de sis angles pel robot esquerra, sis més pel dret i el vector d'orientació i de translació de l'objecte.

  Paràmetres:
  config Par. ent/sort. Una configuració.

- **void alliberar_info_planificador_local (INFO_PLANIF_LOCAL ** *planificador_local*)**

  Allibera la memòria del punter de tipus INFO_PLANIF_LOCAL que retorna el planificador local.

  Paràmetres:
  planificador_local Par. ent/sort. Allibera la variable planificador_local.

- **void canviar_a_configuracio_lliure (bool *detectar_colisions*, DADES_CONFIG * *dades_config*, CONFIG * *config*)**

  Mostreja aleatòriament els angles del robot actiu fins que el robot passiu tanqui la cadena cinemàtica. La configuració obtinguda serà lliure o no depenen del valor de la variable detectar_colisions.

  Paràmetres:
  detectar_colisions Par. ent. Si és certa, s'inspecciona si les configuracions obtingudes són lliures de col.lisions, altrament només si es produeix tancament.
  dades_config Par. ent. Dades constants associades a l'obtenció de la cadena cinemàtica tancada.
  config Par. ent/sort. Configuració lliure que obtindrem.

- **bool canviar_a_configuracio_on_line (DADES_CONFIG * *dades_config*, CONFIG * *config*)**

  Movent l'objecte de Webots amb el mouse es recalcula la cinemàtica inversa per cada robot i es visualitza en cas d'existir solució. També s'extreu per pantalla si hi ha solució en forma de cadena tancada, i en cas de que n'hi hagi s'informa de si és lliure de col.lisions.

  Paràmetres:

  dades_config Par. ent. Punter a la informació constant associada a l'obtenció d'una cadena cinemàtica tancada.

  config Par. ent/sort. Configuracio atual on-line.

  Retorna:

  Un boolea que indica si a trobat una configuració lliure.

- **void canviar_a_configuracio_predeterminada (CONFIG * *config*)**

  Visualitza en Webots una configuració donada la cadena tancada a tractar.

  Paràmetres:

  config Par. ent. Configuracio que volem visualitzar.

- **void carregar_configuracio (FILE * *fa*, CONFIG * *config*)**

  Carrega una configuració en memòria des d'un fitxer de text. El que carrega és el següent: sis angles pel robot esquerra, sis més pel dret i el vector d'orientació i de translació de l'objecte.

  Paràmetres:

  fa Par. ent. Fitxer d'on carreguem la configuració.

  config Par. ent/sort. Configuració que carreguem de fitxer.

- **bool comparar_tipus_solucions_configuracions (CONFIG * *config1*, CONFIG * *config2*)**

  Compara les solucions cinemàtiques inverses del robot passiu per a dues configuracions.

  Paràmetres:

  config1 Par. ent. Primera configuració.

  config2 Par. ent. Segona configuració.

  Retorna:

  Si les dues solucions cinemàtiques són iguals.

- **void copiar_config_a_nova_config (CONFIG * *config*, CONFIG * *nconfig*)**

  Copia la informació emmagatzemada en una configuració a una altra variable.

  Paràmetres:

  config Par. ent. Configuració d'entrada.

nconfig Par. ent/sort. Configuració copiada.

- **void copiar_obtenir_config_a_configuracio (CONFIG * *config*)**

  Copia la configuració obtinguda en les variables de Webots, en una nova variable tipus CONFIG.

  Paràmetres:

  config Par. ent/sort. Configuració a on copiem la configuració obtinguda.

- **CONFIG* crear_configuracio ()**

  Reserva memòria per una configuració, que consta de sis angles pel robot esquerra, sis més pel dret i el vector d'orientació i de translació de l'objecte.

  Retorna:

  Una configuració.

- **INFO_PLANIF_LOCAL* crear_info_planificador_local ()**

  Crea un punter a INFO_PLANIF_LOCAL, que és l'estructura que guarda el que ens interessa d'una crida del planificador local: la configuració previa a un fracàs per col.lisió, la iteració on ha passat, i el resultat de la crida.

  Retorna:

  Un punter a l'estructura INFO_PLANIF_LOCAL.

- **void desar_configuracio (FILE * *fa*, CONFIG * *config*)**

  Desa una configuració en un fitxer de text. El que desa és el següent: sis angles pel robot esquerra, sis més pel dret i el vector d'orientació i de translació de l'objecte.

  Paràmetres:

  fa Par. ent/sort. Fitxer on desem la configuració.

  config Par. ent. Configuració donada.

- **real distancia_entre_configuracions (CONFIG * *config1*, CONFIG * *config2*)**

  Calcula la distància entre dues configuracions, com la distància euclídia entre els vectors angulars de la configuració config1 i la configuració config2.

  Paràmetres:

  config1 Par. ent. Configuració inicial.

  config2 Par. ent. Configuració final.

  Retorna:

  Un real que és la distància euclídia entre les dues configuracions.

- **real distancia_fina_entre_configuracions (DADES_CONFIG * *dades_config*, CONFIG * *configi*, CONFIG * *configf*)**

  Calcula la distància entre dues configuracions, com la longitud de la trajectòria realitzada en la crida del planificador entre aquestes dues.

Paràmetres:

dades_config Par. ent. Dades constants associades a l'obtenció de la cadena cinemàtica tancada.

configi Par. ent. Configuració inicial.

configf Par. ent. Configuració final.

Retorna:

Un real que és la distància euclídia entre les dues configuracions.

- **void escriure_configuracio (CONFIG $*$ *config*)**

Escriu pel canal estàndar de sortida el contingut d'una configuració: sis angles pel robot esquerra, sis més pel dret i el vector d'orientació i de translació de l'objecte.

Paràmetres:

config Par. ent Configuració que escrivim.

- **CONFIG$*$ generar_configuracio_previa_colisio (DADES_CONFIG $*$ *dades_config*, CONFIG $*$ *config_act*)**

Genera direccions aleatòries per una configuració config_act i troba la última lliure de col.lisions per aquella direcció. Per seguir una direcció s'empra el planificador local.

Paràmetres:

dades_config Par. ent. Dades constants associades a l'obtenció de la cadena cinemàtica tancada.

config_act Par. ent/sort. Configuració des d'on volem generar una direcció aleatòria.

Retorna:

La última configuració lliure de col.lisions per una direcció aleatòria.

- **CONFIG$*$ generar_configuracio_robot_aleatori_en_bola_DIST_MAX (DADES_CONFIG $*$ *dades_config*, CONFIG $*$ *config_act*)**

Genera una configuració aleatòria interna en una bola de de distància, centrada en la configuració config_act. A més, la configuració generada és lliure de col.lisions, així com la crida del planificador local entre la de partida i aquesta.

Paràmetres:

dades_config Par. ent. Dades constants associades a l'obtenció de la cadena cinemàtica tancada.

config_act Par. ent. Configuració des d'on d'on volem generar una nova configuració.

Retorna:

Una configuració aleatòria interna en una bola de de distància.

- **void obtenir_configuracio_actual ()**

  Guarda la configuració actual en les taules angle_rob_esq, angle_rob_dret i transl_- objecte de treball de webots_interface.h. Es fa així perquè Webots només permet emmagatzemar la informació dels elements de l'escena de treball en la mateixa zona de memòria per tota una simulació.

- **INFO_PLANIF_LOCAL∗ planificador_local_configi_configf (bool *detectar_colisio*, DADES_CONFIG ∗ *dades_config*, CONFIG ∗ *configi*, CONFIG ∗ *configf*)**

  Mira si existeix una trajectòria entre dues configuracions. Depenen del valor de detectar_colisions es mira si és lliure o no de col.lisions.

  Paràmetres:

  detectar_colisio Par. ent. Si és certa, s'inspecciona si les configuracions obtingudes en la trajectòria són lliures de col.lisions, altrament només si es produeix tancament.

  dades_config Par. ent. Dades constants associades a l'obtenció de la cadena cinemàtica tancada.

  configi Par. ent. Configuració inicial.

  configf Par. ent. Configuració final.

  Retorna:

  Una estructura que conté tota la informació que necessitem.

G.2.3 robots.h

Les rutines que treballen amb els robots.

```
#include "funcions_generals.h"
#include "transf_homogenies.h"
#include "macros_generals.h"
#include "globals.h"
#include "webots_interface.h"
```

Estructures de Dades

struct conf_robot

- **ARTIC** ∗ **artic**
- char ∗ **sol**

Definicions

- #define **ARTIC**(robot) ((robot) → artic)
- #define **ARTIC_i**(robot, i) (((robot) → artic)[i])
- #define **SOLUC**(robot) ((robot) → sol)

Definicions de Tipus

- typedef real **ARTIC**
- typedef **conf_robot CONF_ROBOT**

Funcions

- **CONF_ROBOT** ∗ **crear_robot** ()

  Creació d'una configuració pel robot.

- void **alliberar_robot** (**CONF_ROBOT** ∗∗robot)

  Allibera la informació d'un robot.

- void **assignar_angles_a_robot** (ARTIC ∗artic, **CONF_ROBOT** ∗robot)

  Assignar els angles a un robot.

- void **canviar_angles_robot** (char ∗nom_robot, **CONF_ROBOT** ∗robot)

  Visualitzar una configuració del robot.

- void **angles_robot_ref_paper_ref_RX60** (**CONF_ROBOT** ∗robot)

  Canvi de referència dels angles.

- void **angles_robot_ref_RX60_ref_paper** (**CONF_ROBOT** ∗robot)

Canvi de referència dels angles.

- void **angles_robot_aleatoris** (**CONF_ROBOT** *robot)

  Obté una configuració aleatòria del robot.

- void **obtenir_angles_robot** (char *nom_robot)

  Guarda els angles actuals d'un robot.

- void **copiar_obtencio_angles_a_robot** (char *nom_robot, **CONF_ROBOT** *robot)

  Copia els angles actuals del robot a una altre variable.

- void **copiar_robot_a_nou_robot** (**CONF_ROBOT** *robot, **CONF_ROBOT** *nrobot)

  Traspàs d'una configuració d'un robot a una altra.

- void **cinematica_directa_RX60** (PDH pdh, **CONF_ROBOT** *robot, TRANSF_HOMOG Mr, char *tipus_conf)

  Resol el problema cinemàtic directe pel robot.

- bool **cinematica_inversa_RX60** (PDH pdh, JNTS_RANGE *jr, TRANSF_HOMOG Mr, char *tipus_conf, **CONF_ROBOT** *robot)

  Resol el problema cinemàtic invers pel robot.

- bool **angles_robot_ref_RX60_en_rang** (**CONF_ROBOT** *robot, JNTS_RANGE *jr)

  Comprova si els angles del robot estan en rang.

- void **escriure_angles_robot** (char *nom_robot, const **CONF_ROBOT** *robot)

  Escriu els angles del robot per pantalla.

- void **desar_angles_robot** (FILE *fa, **CONF_ROBOT** *robot)

  Desa els angles del robot en un fitxer.

- void **carregar_angles_robot** (FILE *fa, **CONF_ROBOT** *robot)

  Carrega els angles del robot des d' un fitxer.

- void **desar_angles_robots** (FILE *ftraj, **CONF_ROBOT** *robote, **CONF_ROBOT** *robotd)

  Desa les configuracions de dos robots.

Documentació de les Funcions

Autor:
Gorka Bonals Sastre

Versió:
1.0

Data:
Agost 2005

Descripició
Implementa les rutines que treballen amb el tipus CONF_ROBOT, que conté una configuració d'un robot especificada com els sis angles que el descriuen (camp artic) i el seu tipus de solució cinemàtica (camp sol).

- **void alliberar_robot (CONF_ROBOT ** *robot*)**

  Allibera la memòria de la variable robot, que consta de sis valors angulars pel robot esquerra, sis més pel dret i el vector d'orientació i de translació de l'objecte.

  Paràmetres:
  robot Par. ent/sort. Configuració del robot que volem alliberar.

- **void angles_robot_aleatoris (CONF_ROBOT * *robot*)**

  Troba un vector de sis angles aleatoris que emmagatzema al camp artic del la variable robot.

  Paràmetres:
  robot Par. ent/sort. Conté la configuració aleatòria d'un robot.

- **void angles_robot_ref_paper_ref_RX60 (CONF_ROBOT * *robot*)**

  Transforma els angles de la referència de l'article desenvolupat per Paul a angles aplicables al RX60.

  Paràmetres:
  robot Par. ent/sort. Configuració d'un robot.

- **bool angles_robot_ref_RX60_en_rang (CONF_ROBOT * *robot*, JNTS_RANGE * *jr*)**

  Comprova si els angles del robot estan dintre el rang de les articulacions.

  Paràmetres:
  robot Par. ent. Conté la configuració d'un robot.
  jr Par. ent. Conté els rangs de les articulacions.

  Retorna:
  Si els angles del robot estan en rang.

- **void angles_robot_ref_RX60_ref_paper (CONF_ROBOT * *robot*)**

  Transforma els angles de la referència del RX60 als de l'article d'en Paul.

Paràmetres:

robot Par. ent/sort. Configuració d'un robot.

- **void assignar_angles_a_robot (ARTIC * *artic*, CONF_ROBOT * *robot*)**

  Afegeix uns angles donats a la variable que conté una configuració d'un robot.

  Paràmetres:

  artic Par. ent. Conté un vector de sis angles.

  robot Par. ent/sort. Variable del robot que li assignen els angles donats.

- **void canviar_angles_robot (char * *nom_robot*, CONF_ROBOT * *robot*)**

  Visualitza en Webots pel robot que desitgem, esquerra o dret, els angles continguts en la variable robot.

  Paràmetres:

  nom_robot Par. ent. Nom del robot que volem moure.

  robot Par. ent. Variable que conté una configuració d'un robot.

- **void carregar_angles_robot (FILE * *fa*, CONF_ROBOT * *robot*)**

  Carrega els angles del robot des d'un fitxer.

  Paràmetres:

  fa Par. ent. Fitxer des d'on carreguem la configuració del robot.

  robot Par. ent/sort. Conté els angles del fitxer.

- **void cinematica_directa_RX60 (PDH *pdh*, CONF_ROBOT * *robot*, TRANSF_HOMOG *Mr*, char * *tipus_conf*)**

  Resol el problema directe del robot RX60.

  Paràmetres:

  pdh Par. ent. Conté els paràmetres de Denavit-Hartenberg.

  robot Par.ent. Conté els angles dada del robot.

  Mr Par. ent/sort. És la matriu de transformació directe a trobar.

  tipus_conf Par. ent/sort. És el tipus de configuració cinemàtica que trobem dels angles donats.

- **bool cinematica_inversa_RX60 (PDH *pdh*, JNTS_RANGE * *jr*, TRANSF_HOMOG *Mr*, char * *tipus_conf*, CONF_ROBOT * *robot*)**

  Resol el problema invers del robot RX60.

  Paràmetres:

  pdh Par. ent. Conté els paràmetres de Denavit-Hartenberg.

  jr Par. ent. Conté els rangs de les articulacions.

  Mr Par. ent. Matriu de transformació donada.

  tipus_conf Par. en. És el tipus de configuració cinemàtica que volem determinar.

robot Par. ent/sort Contindrà els angles que volem trobar.

- **void copiar_obtencio_angles_a_robot (char * *nom_robot*, CONF_ROBOT * *robot*)**

  Copia la configuració obtinguda en les variables de Webots, en una nova variable tipus CONFIG_ROBOT.

  Paràmetres:

  nom_robot Par. ent. Nom del robot que volem recuperar els angles.

  robot Par. ent/sort. Configuració copiada.

- **void copiar_robot_a_nou_robot (CONF_ROBOT * *robot*, CONF_ROBOT * *nrobot*)**

  Copia la informació emmagatzemada en una configuració d'un robot a una altra variable del tipus C0NF_ROBOT.

  Paràmetres:

  robot Par. ent. Configuració donada del robot.

  nrobot Par. ent/sort. Configuració on fem el traspàs.

- **CONF_ROBOT* crear_robot ()**

  Reserva memòria per una robot, que consta de sis valors angular i del tipus de solució cinemàtica.

  Retorna:

  Una variable tipus ROBOT.

- **void desar_angles_robot (FILE * *fa*, CONF_ROBOT * *robot*)**

  Desa els angles del robot en un fitxer.

  Paràmetres:

  fa Par. ent. Nom del fitxer a on desem la configuració.

  robot Par. ent. Conté els angles que volem desar.

- **void desar_angles_robots (FILE * *ftraj*, CONF_ROBOT * *robote*, CONF_ROBOT * *robotd*)**

  Desa les configuracions de dos robots.

  Paràmetres:

  ftraj Fitxer a on desem les configuracions dels dos robots.

  robote Conté la configuració d'un robot.

  robotd Conté la configuració d'un robot.

- **void escriure_angles_robot (char * *nom_robot*, const CONF_ROBOT * *robot*)**

  Escriu els angles del robot per pantalla.

Paràmetres:

nom_robot Par. ent. Nom del robot.

robot Par. ent. Conté els angles que escriurem.

- **void obtenir_angles_robot (char * *nom_robot*)**

  Guarda els angles actuals del robot que considerem en la taula angle_rob_esq o angle_rob_dret de webots_interface.h. Es fa així perquè Webots només permet emmagatzemar la informació dels elements de l'escena de treball en la mateixa zona de memòria per tota una simulació.

  Paràmetres:

  nom_robot Par. ent. Nom del robot que volem moure aleatòriament.

G.2.4 objecte.h

Les rutines que treballen amb l'objecte.

```
#include "globals.h"
#include "funcions_generals.h"
#include "transf_homogenies.h"
#include "macros_generals.h"
#include "webots_interface.h"
```

Estructures de Dades

struct conf_obj

- **TRANSL** ∗ **transl**
- **ROT** ∗ **rot**

Definicions

- #define **TRANSL**(objecte) ((objecte) → transl)
- #define **TRANSL_i**(objecte, i) (((objecte) → transl)[i])
- #define **ROT**(objecte) ((objecte) → rot)
- #define **ROT_i**(objecte, i) (((objecte) → rot)[i])

Definicions de Tipus

- typedef real **TRANSL**
- typedef real **ROT**
- typedef **conf_obj CONF_OBJECTE**

Funcions

- **CONF_OBJECTE** ∗ **crear_posicio_objecte** ()

  Creació d'una configuració de l'objecte.

- void **alliberar_posicio_objecte** (**CONF_OBJECTE** ∗∗objecte)

  Alliberació d'una configuració de l'objecte.

- void **copiar_objecte_a_nou_objecte** (**CONF_OBJECTE** ∗objecte, **CONF_OBJECTE** ∗nobjecte)

  Copia la informació d'un objecte a un altre.

- void **canviar_posicio_objecte** (**CONF_OBJECTE** ∗objecte)

  Visualitzar una configuració de l'objecte.

- void **posicio_objecte_aleatoria** (**CONF_OBJECTE** ∗objecte)

Calcula una configuració aleatòria per l'objecte.

- void **obtenir_posicio_objecte** ()

  Obté la posició i orientació de l'objecte.

- void **copiar_obtencio_posicio_a_objecte** (**CONF_OBJECTE** *objecte)

  Copia la configuració actual de l'objecte a una altra variable.

- void **cinematica_directa_objecte** (**CONF_OBJECTE** *objecte, TRANSF_HOMOG Mo)

  Resol el problema cinemàtic directe per l'objecte.

- void **cinematica_inversa_objecte** (TRANSF_HOMOG Mo, **CONF_OBJECTE** *objecte)

  Resol el problema cinemàtic invers per l'objecte.

- void **escriure_posicio_objecte** (const **CONF_OBJECTE** *objecte)

  Escriure per pantalla la posició i orientació de l'objecte.

- void **desar_posicio_objecte** (FILE *fa, **CONF_OBJECTE** *objecte)

  Desa en un fitxer la posició i orientació de l'objecte.

- void **carregar_posicio_objecte** (FILE *fa, **CONF_OBJECTE** *objecte)

  Carrega d'un fitxer la posició i orientació de l'objecte.

Documentació de les Funcions

Autor:
Gorka Bonals Sastre

Versió:
1.0

Data:
Agost 2005

Descripició
Implementem les rutines que treballen amb el tipus CONF_OBJECTE, que conté una configuració d'un objecte especificada com els sis angles que el descriuen (camp artic) i el seu tipus de solució cinemàtica (camp sol).

- **void alliberar_posicio_objecte (CONF_OBJECTE ** *objecte*)**

  Paràmetres:
  objecte Par. ent/sort. Configuració de l'objecte que volem alliberar.

- **void canviar_posicio_objecte (CONF_OBJECTE * *objecte*)**

  Visualitza en Webots la posició i orientació de l'objecte a partir de la variable objecte.

  Paràmetres:

  objecte Par. ent. Configuració de l'objecte.

- **void carregar_posicio_objecte (FILE * *fa*, CONF_OBJECTE * *objecte*)**

  Carrega en la variable objecte la configuració d'un objecte que està en el fitxer fa.

  Paràmetres:

  fa Par. ent. Fitxer que llegim per lectura.

  objecte Par. ent/sort. Punter a una configuració de l'objecte.

- **void cinematica_directa_objecte (CONF_OBJECTE * *objecte*, TRANSF_HOMOG *Mo*)**

  Resol el problema cinemàtic directe per l'objecte. És a dir, a partir de la posició i orientació de l'objecte troba la matriu de transformació que el descriu.

  Paràmetres:

  objecte Par. ent. Punter a una configuració de l'objecte.

  Mo Par. ent/sort. Transformació homogènia que volem trobar.

- **void cinematica_inversa_objecte (TRANSF_HOMOG *Mo*, CONF_OBJECTE * *objecte*)**

  Resol el problema cinemàtic invers per l'objecte. És a dir, a partir de la matriu de transformació de l'objecte troba la posició i orientació del mateix.

  Paràmetres:

  Mo Par. ent. Transformació homogènia donada.

  objecte Par.ent/sort. Configuració de l'objecte.

- **void copiar_objecte_a_nou_objecte (CONF_OBJECTE * *objecte*, CONF_OBJECTE * *nobjecte*)**

  Copia la informació d'un objecte a un altre.

  Paràmetres:

  objecte Par. ent. Configuració de l'objecte.

  nobjecte Par. ent/sort. Configuració de l'objecte.

- **void copiar_obtencio_posicio_a_objecte (CONF_OBJECTE * *objecte*)**

  Copia la configuració obtinguda en les variables de Webots de l'objecte, en una nova variable tipus CONFIG_ROBOT.

  Paràmetres:

  objecte Par. ent/sort. Configuració de l'objecte copia.

- **CONF_OBJECTE* crear_posicio_objecte ()**

  Reserva memòria per una configuració del objecte, que consta de un vector de translació i un vector de rotació.

  Retorna:
  Un punter al tipus CONF_OBJECTE.

- **void desar_posicio_objecte (FILE * *fa*, CONF_OBJECTE * *objecte*)**

  Desa en un fitxer el vector de posició i orientació de la variable objecte.

  Paràmetres:
  fa Par. sort. Fitxer que llegim per escritura.
  objecte Par. ent. Configuració de l'objecte que desarem en el fitxer.

- **void escriure_posicio_objecte (const CONF_OBJECTE * *objecte*)**

  Escriu pel canal estàndard de sortida el vector de posició i orientacicó de l'objecte.

  Paràmetres:
  objecte Par. ent. Configuració de l'objecte.

- **void obtenir_posicio_objecte ()**

  Guarda el vector de translació i orientació actuals de l'objecte en les taules transl_objecte i rot_objecte. Es fa així perquè Webots només permet emmagatzemar la informació dels elements de l'escena de treball en la mateixa zona de memòria per tota una simulació.

- **void posicio_objecte_aleatoria (CONF_OBJECTE * *objecte*)**

  Mostreja aletòriament el vector de translació i orientació de l'objecte obtenint així una configuració aleatòria del mateix.

  Paràmetres:
  objecte Par. ent/sort. Configuració aleatòria de l'objecte.

G.2.5 vector.h

Conté les rutines que treballen amb un vector.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "macros_generals.h"
#include "general.h"
```

Estructures de Dades

struct vect

- int **n**
- real * **v**

Definicions

- #define **DIM**(vector) ((vector) → n)
- #define **ELEM**(vector) ((vector) → v)
- #define **ELEM_i**(vector, i) ((vector) → v[i])

Definicions de Tipus

- typedef **vect VECTOR**

Funcions

- **VECTOR** * **crear_vector** (int n)

  Crea un vector d'n components.

- void **alliberar_vector** (**VECTOR** **vector)

  Allibera un vector d'n components.

- **VECTOR** * **escalar_per_vector** (**VECTOR** *vector, real k)

  Producte d'un escalar per un vector.

- **VECTOR** * **restar_vectors** (**VECTOR** *vector1, **VECTOR** *vector2)

  S'obté la resta entre dos vectors.

- **VECTOR** * **sumar_vectors** (**VECTOR** *vector1, **VECTOR** *vector2)

  S'obté la suma entre dos vectors.

- real **modul_vector** (**VECTOR** *vector)

S'obté el mòdul d'un vector.

- **VECTOR** ∗ **vector_long_landa** (**VECTOR** ∗vector, real landa)

  S'obté un vector de longitud lambda.

- void **escriure_vector** (**VECTOR** ∗vector)

  S'escriu un vector per pantalla.

- **VECTOR** ∗ **vector_enessim** (int n, **VECTOR** ∗vector_ini, **VECTOR** ∗vector_landa)

  Calcula l'equació vectorial de la recta discretitzada.

- int **ultima_iteracio** (**VECTOR** ∗vector_ini, **VECTOR** ∗vector_fi, real landa)

  Calcula el nombre d'iteracions.

Documentació de les Funcions

Autor:
Gorka Bonals Sastre

Versió:
1.0

Data:
Agost 2005

Descripició
Implementa totes les rutines relacionades amb operacions bàsiques amb vectors.

- **void alliberar_vector (VECTOR ∗∗ *vector*)**

  Primer allibera memòria dels n components del camp v del vector i després allibera la memòria d'aquest darrer.

  Paràmetres:
  vector Par. ent/sort. És el vector que volem alliberar.

- **VECTOR∗ crear_vector (int *n*)**

  Reserva memòria per una varialbe del tipus VECTOR i pel camp v reserva memòria a n reals.

  Paràmetres:
  n Par. ent. Nombre de components.

  Retorna:

- **VECTOR∗ escalar_per_vector (VECTOR ∗ *vector*, real *k*)**

  Realitza el producte d'un escalar per un vector.

  Paràmetres:

  vector Par. ent/sort. Vector de n components.

  k Par. ent. Valor escalars.

- **void escriure_vector (VECTOR ∗ *vector*)**

  S'escriu les components d'un vector pel canal estandar de sortida.

  Paràmetres:

  vector Par. ent. Vector d' n components.

- **real modul_vector (VECTOR ∗ *vector*)**

  S'obté el mòdul d'un vector.

  Paràmetres:

  vector Par. ent. Vector d' n components.

  Retorna:

  El mòdul que és un real.

- **VECTOR∗ restar_vectors (VECTOR ∗ *vector1*, VECTOR ∗ *vector2*)**

  S'obté la resta entre dos vectors.

  Paràmetres:

  vector1 Par. ent. Vector d' n components.

  vector2 Par. ent. Vector d' n components.

  Retorna:

  El vector resta.

- **VECTOR∗ sumar_vectors (VECTOR ∗ *vector1*, VECTOR ∗ *vector2*)**

  S'obté la suma entre dos vectors.

  Paràmetres:

  vector1 Par. ent. Vector d' n components.

  vector2 Par. ent. Vector d' n components.

  Retorna:

  El vector suma.

- **int ultima_iteracio (VECTOR ∗ *vector_ini*, VECTOR ∗ *vector_fi*, real *landa*)**

  Calcula el nombre de passos de la recta discretitzada

  Paràmetres:

  vector_ini Par. ent. Vector o punt d'inci de de la recta.

vector_fi Par. ent. Vector o punt de final de la recta.
landa Par. ent. Longitud del pas d'avanç.

Retorna:
El nombre de passos que hem discretitzat la recta.

- **VECTOR∗ vector_enessim (int *n*, VECTOR ∗ *vector_ini*, VECTOR ∗ *vector_landa*)**

  Calcula l'equació vectorial de la recta discretitzada del la secció de la mèmoria 5.4.2.2.

  Paràmetres:
  n Par. ent. Volem anar al pas enessim.
  vector_ini Par. ent. Vector o punt d'inci de de la recta.
  vector_landa Vector de longitud lambda.

  Retorna:
  El vector de la recta en la iteració enèssima.

- **VECTOR∗ vector_long_landa (VECTOR ∗ *vector*, real *landa*)**

  S'obté el mòdul d'un vector de longitud lambda.

  Paràmetres:
  vector Par. ent. Vector d' n components.
  landa Par. ent. Longitud del pas d'avanç.

  Retorna:
  El vector però amb longitud lambda.

G.3 Mòduls secundaris

Associats a la interfície gràfica

interface.h

Conté les rutines i les estructures de dades per crear cada una de les finestres (secció E.3).

config.h

Conté les constants que intervenen en la creació de la interfície gràfica.

support.h

Conté les macros que intervenen en la creació de la interfície gràfica.

callbacks.h

Conté les rutines associades a cada un dels botons de la interfície gràfica. Dintre d'aquestes inserim altres rutines del nostre programa.

Associats al detector de col·lisions de Webots

lib_colisions.c

Conté les rutines que utilitza el detector de col·lisions de Webots. Tot i que no és un fitxer d'extensió *h* l'hem inclòs per la seva importància.

tau_colisions.h

Conté les rutines per llegir el fitxer d'extensió *col* (secció E.4.3) per tal de crear una matriu que contingui les parelles de sòlids que no volem verificar la seva col·lisió.

colisio.h

Conté les rutines per inicialitzar i llegir el fitxer *colisions.txt*. Aquest conté un 1 si la última detecció ha estat de col·lisió, altrament conté un 0.

Associats a l'algorisme de planificació

globals.h

Els paràmetres de l'algorisme són constants per un problema donat, però es poden canviar per altres problemes mitjançant la finestra de paràmetres. És per això que els hem definit com variables globals.

Gtypes.h

Conté bàsicament els tipus i les rutines per calcular la cinemàtica inversa i directa de manipuladors simples.

fitxer.h

Agrupa les rutines que llegeixen o desen la informació dels fitxers *.map, *.par, *.infi i *.trj (secció E.5.5). També conté les rutines que generen els fitxers *.map.wbt i *.trj.wbt que permeten representar en Webots un mapa i una trajectòria.

vei.h

Conté les rutines que treballen sobre una taula de nodes candidats a veïns. L'estructura de dades que utilitzem és una tupla del tipus TAULA_VERTEX_VEINS amb dos camps: un d'ells és un punter a una altre tupla del tipus INFO_VEI i l'altre camp correspon al nombre d'elements que voldrem reservar memòria del tipus INFO_VEI. INFO_VEI és una tupla que conté els camps: (1) distància del node c al veí,(2) punter del node candidat a veí, (3) índex del node candidat a veí.

vertex_exp.h

Conté les rutines que treballen sobre una taula de nodes a expansionar i del seu pes associat (secció 4.1.2). L'estructura de dades per representar aquesta taula és com en el cas del vei.h però en comptes del camp INFO_VEI tenim el camp VERTEX_EXP.

traject.h

Conté les rutines que treballen amb una taula de reduccions de distància associades a cadascun dels nodes d'una trajectòria solució (secció 5.4.5.1). L'estructura de dades per representar aquesta taula és com en el cas del vei.h però en comptes del camp INFO_VEI tenim el camp NODE_TRAJECT_AUX. NODE_TRAJECT_AUX conté forces camps on el de més interès és la distància de reducció.

component.h

Conté les rutines que treballen sobre una taula de nombre de nodes per componen connex i d'un node representatiu de cadascun d'ells.

trasnf_homogenies.h

Conté totes les operacions bàsiques que podem realitzar amb transformacions homogènies, com per exemples les associades a rotacions d'angles respecte eixos x, y i z entre d'altres.

transf_homog_ctnt.h

Calcula les transformacions homogènies que són constants per un problema donat: les que posicionen i orienten la base dels robots respecte la referència absoluta 5.4.2.1 i les que estableixen la prensió esquerra i dreta de l'objecte (secció D.1).

transf_homog_ctnt.h

Agrupa les transformacions homogènies en una tupla de tipus TRANSF_CTNT.

wgraph.h

Representa un graf com una matriu on les files corresponen als nodes i les columnes diferents arestes que surten de cada vèrtex. Aquesta estructura de graf és la que s'utilitza per implementar la rutina que calcula l'algorisme d'obtenció del camí mínim (secció 5.4.5.2).

Que interactuen amb rutines de Webots

noms_gll.h

Etiqueta amb constants estàndard tots els noms dels nodes de l'escena de Webots que representen articulacions. Així si canviéssim els noms de l'escena en el programa només suposaria actualitzar l'etiqueta corresponent d'aquest mòdul.

webots_interface.h

Encapsula totes les rutines que utilitzem de Webots (secció F.1).

Associats a l'estructura de dades per emmagatzemar el graf

llr.h

Conté les operacions que treballen amb una llista enllaçada.

llrtools.h

Conté utilitats que es basen en operacions bàsiques del mòdul llr.h

glr.h

Conté les rutines per crear una graf a partir d'una llista doblement enllaçada.

glrtools.h

Conté utilitats que es basen en operacions bàsiques del mòdul glr.h

Altres

generals.h

Conté definició de tipus força bàsics, com serien variables lògiques, valors reals, etc.

ctnt_generals.h

macros_generals.h

Conté tots aquells paràmetres que són constants en un problema. Molts d'ells tenen les respectives variables globals associades (globals.h), que per defecte s'inicialitzen amb aquestes constants.

funcions_generals.h

Són funcions bastant bàsiques que treballen amb valors reals.

Apèndix H

Codi font

H.1 main.c

```
/*
 * Initial main.c file generated by Glade. Edit as required.
 * Glade will not overwrite this file.
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

/////////////////////////////////////////////////////
/// @file main.c
/// @brief Construccio de la interficie grafica.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par Descripicio
///     Crida les rutines de construccio
///     de les diferents finestres de la interficie grafica.
///     A l'ultima linia s'espera a que l'usuari realitzi
///     algun event en alguna de les finestres (com premer
///     un boto).
/////////////////////////////////////////////////////

#define _MAIN_

#include <gtk/gtk.h>
#include "interface.h"
#include "support.h"
#include "globals.h"
#include "webots_interface.h"

int
main (int argc, char *argv[])
{
  GtkWidget *graus_llibertat_window;
  GtkWidget *window_parametres;
  GtkWidget *tasques_window;
  GtkWidget *analisi_graf_window;
  time_t t;

  webots_init();
  inicialitzar_globals();
  srand(time(&t));

#ifdef ENABLE_NLS
  bindtextdomain (GETTEXT_PACKAGE, PACKAGE_LOCALE_DIR);
  bind_textdomain_codeset (GETTEXT_PACKAGE, "UTF-8");
  textdomain (GETTEXT_PACKAGE);
#endif

  gtk_set_locale();
  gtk_init (&argc, &argv);

  /*
   * The following code was added by Glade to create one of each component
   * (except popup menus), just so that you see something after building
   * the project. Delete any components that you don't want shown initially.
   */
  graus_llibertat_window = create_graus_llibertat_window ();
  gtk_widget_show (graus_llibertat_window);
  window_parametres= create_window_parametres();
  gtk_widget_show (window_parametres);
  tasques_window = create_tasques_window ();
```

```
  gtk_widget_show (tasques_window);
  analisi_graf_window = create_analisi_graf_window ();
  gtk_widget_show (analisi_graf_window);

  gtk_main ();

  return 0;
}
```

H.2 globals.h

```
#ifndef GLOBALS_H
#define GLOBALS_H

/////////////////////////////////////////////////////
/// @file globals.h
/// @brief Variables globals per tots els moduls.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par Descripicio
///      Son els parametres del problema que per ser
///      canviats a la finestra de parametres han de
///      ser variables globals.
/////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ctnt_generals.h"
#include "Gtypes.h"
#include "transf_ctnt.h"
#include "general.h"

#ifdef _MAIN_
#   define GLOBAL
#else
#   define GLOBAL extern
#endif

void inicialitzar_globals();

GLOBAL int g_MAX_NODES;                 // Max. nombre nodes del mapa de rutes
GLOBAL real g_PERC_EXP;                 // Perc. del nombre àmxim de nodes.
GLOBAL int g_MIN_NODES;                 // Min. nombre nodes de cada component connex
GLOBAL int g_NVEINS;                    // Max. nombre de veins en l'aprenentatge
GLOBAL real g_MAX_DIST;                 // Max. distancia dels veins a connectar
GLOBAL int g_CONT_SUP_DIST_MAX;         // Max  nombre de nodes generats per sobre de dist max.
GLOBAL int g_CONT_NO_CAMI;              // Max. nombre de fracassos del cami local.
GLOBAL int g_MAX_NODES_CAMI;

GLOBAL real g_XMAX;                     //Llindar superior per la coordenada x.
GLOBAL real g_XMIN;                     //Llindar superior per la coordenada y.
GLOBAL real g_YMAX;                     //Llindar superior per la coordenada z.
GLOBAL real g_YMIN;                     //Llindar inferior per la coordenada x.
GLOBAL real g_ZMAX;                     //Llindar inferior per la coordenada y.
GLOBAL real g_ZMIN;                     //Llindar inferior per la coordenada z.


GLOBAL real g_ROT_XMAX;                 //Llindar superior per la rotacio respecte l'eix x.
GLOBAL real g_ROT_YMAX;                 //Llindar superior per la rotacio respecte l'eix y.
GLOBAL real g_ROT_ZMAX;                 //Llindar superior per la rotacio respecte l'eix z.
GLOBAL real g_ROT_XMIN;                 //Llindar inferior per la rotacio respecte l'eix x.
GLOBAL real g_ROT_YMIN;                 //Llindar inferior per la rotacio respecte l'eix y.
GLOBAL real g_ROT_ZMIN;                 //Llindar inferior per la rotacio respecte l'eix z.

GLOBAL real g_JMIN1;                    //Llindar inferior de l'articulacio 1 del robot.
GLOBAL real g_JMAX1;                    //Llindar superior de l'articulacio 1 del robot.
GLOBAL real g_JMIN2;                    //Llindar inferior de l'articulacio 2 del robot.
GLOBAL real g_JMAX2;                    //Llindar superior de l'articulacio 2 del robot.
GLOBAL real g_JMIN3;                    //Llindar inferior de l'articulacio 3 del robot.
GLOBAL real g_JMAX3;                    //Llindar superior de l'articulacio 3 del robot.
GLOBAL real g_JMIN4;                    //Llindar inferior de l'articulacio 4 del robot.
GLOBAL real g_JMAX4;                    //Llindar superior de l'articulacio 4 del robot.
GLOBAL real g_JMIN5;                    //Llindar inferior de l'articulacio 5 del robot.
GLOBAL real g_JMAX5;                    //Llindar superior de l'articulacio 5 del robot.
GLOBAL real g_JMIN6;                    //Llindar inferior de l'articulacio 6 del robot.
GLOBAL real g_JMAX6;                    //Llindar superior de l'articulacio 6 del robot.

GLOBAL real g_pas_cami_local;           //Longitud de pas d'çavan del planificador local.
GLOBAL FILE* g_fd;                      //Fitxer on guardem la informacio d'execucio
GLOBAL FILE* g_ftraj;                   //Fitxer on guardem els punts d'una trajectoria.

GLOBAL bool g_crear_ftraj;              //Boolea per guardar una trajectoria en fitxer.
GLOBAL int g_cont_comp_connexes;        //Contador de components connexos.
GLOBAL int g_N_PASOS;                   //Max. nombre de passos.
GLOBAL int g_N_FORA_DIST_MAX;           //Max. configuracions fora la bola de distancia.
GLOBAL int g_MAX_NO_CAMI_LOCAL;         //Max. fracassos del planificador local.
GLOBAL int g_max_intents;               //Max. intents per intentar la connexio.
GLOBAL int g_max_profund;               //Max. nombre de nodes d'una òtrajectria aleatoria.
GLOBAL int g_NUM_COMPONENTS;            //Contador de nombre de components.
```

```
#endif
```

H.3 globals.c

```
#include "globals.h"

void inicialitzar_globals()
{
  g_MAX_NODES=MAX_NODES;
  g_PERC_EXP=PERC_EXP;
  g_MIN_NODES=MIN_NODES;
  g_NVEINS=NVEINS;
  g_MAX_DIST=MAX_DIST;
  g_CONT_NO_CAMI=CONT_NO_CAMI;
  g_CONT_SUP_DIST_MAX=CONT_SUP_DIST_MAX;
  g_MAX_NODES_CAMI=MAX_NODES_CAMI;

  g_XMAX=XMAX;
  g_XMIN=XMIN;
  g_YMAX=YMAX;
  g_YMIN=YMIN;
  g_ZMAX=ZMAX;
  g_ZMIN=ZMIN;

  g_ROT_XMAX=ROT_XMAX;
  g_ROT_YMAX=ROT_YMAX;
  g_ROT_ZMAX=ROT_ZMAX;
  g_ROT_XMIN=ROT_XMIN;
  g_ROT_YMIN=ROT_YMIN;
  g_ROT_ZMIN=ROT_ZMIN;


  g_JMIN1=JMIN1;
  g_JMAX1=JMAX1;
  g_JMIN2=JMIN2;
  g_JMAX2=JMAX2;
  g_JMIN3=JMIN3;
  g_JMAX3=JMAX3;
  g_JMIN4=JMIN4;
  g_JMAX4=JMAX4;
  g_JMIN5=JMIN5;
  g_JMAX5=JMAX5;
  g_JMIN6=JMIN6;
  g_JMAX6=JMAX6;

  g_pas_cami_local=PAS_CAMI_LOCAL;

  g_N_PASOS=N_PASOS;
  g_N_FORA_DIST_MAX=N_FORA_DIST_MAX;
  g_MAX_NO_CAMI_LOCAL=MAX_NO_CAMI_LOCAL;
  g_max_intents=MAX_INTENTS;
  g_max_profund=MAX_PROFUND;
  g_NUM_COMPONENTS=NUM_COMPONENTS;
  g_crear_ftraj=FALSE;
}
```

H.4 interface.h

```
/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

/////////////////////////////////////////////////////////
/// @file interface.h
/// @brief Rutines de les finestres grafiques.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
/// é      Cont les estructures de dades que treballen
///       amb les rutines que construeixen les
///       diferents finestres de l'interficie.
/////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////
//Agrupa els botons robot, prensio i çpina.
/////////////////////////////////////////////////////////
typedef struct tres_toggle
{
  GtkWidget *robot_togglebutton;
  GtkWidget *prensio_togglebutton;
  GtkWidget *pinca_togglebutton;
}TRES_TOGGLE;

/////////////////////////////////////////////////////////
//Agrupa les 6 barres de desplacament de G. llibertat.
/////////////////////////////////////////////////////////
typedef struct sis_widgets
```

```
{
  GtkWidget *g_widget_1;
  GtkWidget *g_widget_2;
  GtkWidget *g_widget_3;
  GtkWidget *g_widget_4;
  GtkWidget *g_widget_5;
  GtkWidget *g_widget_6;
}SIS_WIDGETS;

/////////////////////////////////////////////////////
//Agrupa totes les entrades de Parametres.
/////////////////////////////////////////////////////
typedef struct widgets_parametres
{
  GtkWidget *borrar_graf_checkbutton;
  GtkWidget *nom_base_entry;
  GtkWidget *max_nodes_entry;
  GtkWidget *min_nodes_entry;
  GtkWidget *percen_expansio_entry;
  GtkWidget *num_veins_entry;
  GtkWidget *max_dist_entry;
  GtkWidget *rob_esq_aleat_radiobutton;
  GtkWidget *objecte_aleat_radiobutton;
  GtkWidget *rob_dret_aleat_radiobutton;
  GtkWidget *pas_cami_local_entry;
  GtkWidget *rob_esq_interpol_radiobutton;
  GtkWidget *rob_dret_interpol_radiobutton;
  GtkWidget *num_pasos_entry;
  GtkWidget *fora_dist_max_entry;
  GtkWidget *max_no_local_entry;
  GtkWidget *max_profunditat_entry;
  GtkWidget *fitxer_debug_checkbutton;
  GtkWidget *nom_fitxer_debug_entry;
  GtkWidget *angle_1_min_entry;
  GtkWidget *angle_1_max_entry;
  GtkWidget *angle_2_min_entry;
  GtkWidget *angle_2_max_entry;
  GtkWidget *angle_3_min_entry;
  GtkWidget *angle_4_max_entry;
  GtkWidget *angle_4_min_entry;
  GtkWidget *angle_3_max_entry;
  GtkWidget *angle_6_max_entry;
  GtkWidget *angle_6_min_entry;
  GtkWidget *angle_5_max_entry;
  GtkWidget *angle_5_min_entry;
  GtkWidget *config_rob_esq_entry;
  GtkWidget *config_rob_dret_entry;
  GtkWidget *x_max_entry;
  GtkWidget *x_min_entry;
  GtkWidget *y_max_entry;
  GtkWidget *y_min_entry;
  GtkWidget *z_max_entry;
  GtkWidget *z_min_entry;
  GtkWidget *rot_x_max_entry;
  GtkWidget *rot_x_min_entry;
  GtkWidget *rot_y_max_entry;
  GtkWidget *rot_y_min_entry;
  GtkWidget *rot_z_max_entry;
  GtkWidget *rot_z_min_entry;
  GtkWidget *coord_cart_min_prensio_entry;
  GtkWidget *coord_cart_max_prensio_entry;
}WIDGETS_PARAMETRES;

/////////////////////////////////////////////////////
//Les 4 finestres que creem.
/////////////////////////////////////////////////////
GtkWidget* create_graus_llibertat_window (void);
GtkWidget* create_window_parametres (void);
GtkWidget* create_tasques_window (void);
GtkWidget* create_analisi_graf_window (void);
```

H.5 interface.c

```
/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>

#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

#include "callbacks.h"
```

```
#include "interface.h"
#include "support.h"

WIDGETS_PARAMETRES* widgets_parametres;

#define GLADE_HOOKUP_OBJECT(component,widget,name) \
  g_object_set_data_full (G_OBJECT (component), name, \
    gtk_widget_ref (widget), (GDestroyNotify) gtk_widget_unref)

#define GLADE_HOOKUP_OBJECT_NO_REF(component,widget,name) \
  g_object_set_data (G_OBJECT (component), name, widget)

GtkWidget*
create_graus_llibertat_window (void)
{
  GtkWidget *graus_llibertat_window;
  GtkWidget *g_llibertat_layout;
  GtkWidget *g_llibertat_robot_radiobutton;
  GtkWidget *g_llibertat_prensio_radiobutton;
  GtkWidget *g_llibertat_pinca_radiobutton;
  GSList *g_llibertat_robot_radiobutton_group = NULL;
  GtkWidget *g_llibertat_objecte_radiobutton;
  GtkWidget *g_llibertat_1_label;
  GtkWidget *g_llibertat_2_label;
  GtkWidget *g_llibertat_3_label;
  GtkWidget *g_llibertat_4_label;
  GtkWidget *g_llibertat_5_label;
  GtkWidget *g_llibertat_6_label;
  GtkWidget *g_llibertat_1_hscale;
  GtkWidget *g_llibertat_2_hscale;
  GtkWidget *g_llibertat_3_hscale;
  GtkWidget *g_llibertat_4_hscale;
  GtkWidget *g_llibertat_5_hscale;
  GtkWidget *g_llibertat_6_hscale;
  GtkWidget *vseparator1;
  GtkWidget *esq_radiobutton;
  GSList *esq_radiobutton_group = NULL;
  GtkWidget *dret_radiobutton;
  GtkWidget *g_llibertat_1_entry;
  GtkWidget *g_llibertat_2_entry;
  GtkWidget *g_llibertat_3_entry;
  GtkWidget *g_llibertat_4_entry;
  GtkWidget *g_llibertat_5_entry;
  GtkWidget *g_llibertat_6_entry;
  GtkWidget *home_button;
  GtkWidget *obertura_pinca_label;
  GtkWidget *obertura_pinca_hscale;
  GtkWidget *obertura_pinca_entry;
  SIS_WIDGETS* g_sis_hscale;
  SIS_WIDGETS* g_sis_label;
  GtkWidget *prensio_on_line_CheckButton;
  //DOS_TOGGLE* g_dos_toggle;
  TRES_TOGGLE* g_tres_toggle;

  graus_llibertat_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_widget_set_size_request (graus_llibertat_window, 226, 385);
  gtk_window_set_title (GTK_WINDOW (graus_llibertat_window), _("Graus de Llibertat"));

  g_llibertat_layout = gtk_layout_new (NULL, NULL);
  gtk_widget_show (g_llibertat_layout);
  gtk_container_add (GTK_CONTAINER (graus_llibertat_window), g_llibertat_layout);
  gtk_layout_set_size (GTK_LAYOUT (g_llibertat_layout), 400, 400);
  GTK_ADJUSTMENT (GTK_LAYOUT (g_llibertat_layout)->hadjustment)->step_increment = 0;
  GTK_ADJUSTMENT (GTK_LAYOUT (g_llibertat_layout)->vadjustment)->step_increment = 0;

  g_llibertat_robot_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("robot"));
  gtk_widget_show (g_llibertat_robot_radiobutton);
  gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_robot_radiobutton, 8, 8);
  gtk_widget_set_size_request (g_llibertat_robot_radiobutton, 81, 20);
  gtk_radio_button_set_group (GTK_RADIO_BUTTON (g_llibertat_robot_radiobutton),
                              g_llibertat_robot_radiobutton_group);
  g_llibertat_robot_radiobutton_group = gtk_radio_button_get_group
    (GTK_RADIO_BUTTON (g_llibertat_robot_radiobutton));

  g_llibertat_objecte_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("objecte"));
  gtk_widget_show (g_llibertat_objecte_radiobutton);
  gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_objecte_radiobutton, 8, 80);
  gtk_widget_set_size_request (g_llibertat_objecte_radiobutton, 75, 20);
  gtk_radio_button_set_group (GTK_RADIO_BUTTON (g_llibertat_objecte_radiobutton),
                              g_llibertat_robot_radiobutton_group);
  g_llibertat_robot_radiobutton_group = gtk_radio_button_get_group
    (GTK_RADIO_BUTTON (g_llibertat_objecte_radiobutton));

  g_llibertat_prensio_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("prensio"));
  gtk_widget_show (g_llibertat_prensio_radiobutton);
  gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_prensio_radiobutton, 8, 32);
  gtk_widget_set_size_request (g_llibertat_prensio_radiobutton, 71, 20);
  gtk_radio_button_set_group
    (GTK_RADIO_BUTTON (g_llibertat_prensio_radiobutton), g_llibertat_robot_radiobutton_group);
  g_llibertat_robot_radiobutton_group = gtk_radio_button_get_group
    (GTK_RADIO_BUTTON (g_llibertat_prensio_radiobutton));
```

```
g_llibertat_pinca_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("Ã§pina"));
gtk_widget_show (g_llibertat_pinca_radiobutton);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_pinca_radiobutton, 8, 56);
gtk_widget_set_size_request (g_llibertat_pinca_radiobutton, 97, 20);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (g_llibertat_pinca_radiobutton),
                            g_llibertat_robot_radiobutton_group);
g_llibertat_robot_radiobutton_group = gtk_radio_button_get_group
  (GTK_RADIO_BUTTON (g_llibertat_pinca_radiobutton));



g_llibertat_1_label = gtk_label_new (_("angle 1"));
gtk_widget_show (g_llibertat_1_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_1_label, 8, 120);
gtk_widget_set_size_request (g_llibertat_1_label, 50, 16);
gtk_label_set_justify (GTK_LABEL (g_llibertat_1_label), GTK_JUSTIFY_LEFT);
gtk_label_set_line_wrap (GTK_LABEL (g_llibertat_1_label), TRUE);


g_llibertat_2_label = gtk_label_new (_("angle 2"));
gtk_widget_show (g_llibertat_2_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_2_label, 8, 152);
gtk_widget_set_size_request (g_llibertat_2_label, 50, 16);
gtk_label_set_justify (GTK_LABEL (g_llibertat_2_label), GTK_JUSTIFY_LEFT);
gtk_label_set_line_wrap (GTK_LABEL (g_llibertat_2_label), TRUE);

g_llibertat_3_label = gtk_label_new (_("angle 3"));
gtk_widget_show (g_llibertat_3_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_3_label, 8, 184);
gtk_widget_set_size_request (g_llibertat_3_label, 50, 16);
gtk_label_set_justify (GTK_LABEL (g_llibertat_3_label), GTK_JUSTIFY_LEFT);
gtk_label_set_line_wrap (GTK_LABEL (g_llibertat_3_label), TRUE);

g_llibertat_4_label = gtk_label_new (_("angle 4"));
gtk_widget_show (g_llibertat_4_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_4_label, 8, 216);
gtk_widget_set_size_request (g_llibertat_4_label, 50, 16);
gtk_label_set_justify (GTK_LABEL (g_llibertat_4_label), GTK_JUSTIFY_LEFT);
gtk_label_set_line_wrap (GTK_LABEL (g_llibertat_4_label), TRUE);

g_llibertat_5_label = gtk_label_new (_("angle 5"));
gtk_widget_show (g_llibertat_5_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_5_label, 8, 248);
gtk_widget_set_size_request (g_llibertat_5_label, 50, 16);
gtk_label_set_justify (GTK_LABEL (g_llibertat_5_label), GTK_JUSTIFY_LEFT);
gtk_label_set_line_wrap (GTK_LABEL (g_llibertat_5_label), TRUE);

g_llibertat_6_label = gtk_label_new (_("angle 6"));
gtk_widget_show (g_llibertat_6_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_6_label, 8, 280);
gtk_widget_set_size_request (g_llibertat_6_label, 50, 16);
gtk_label_set_justify (GTK_LABEL (g_llibertat_6_label), GTK_JUSTIFY_LEFT);
gtk_label_set_line_wrap (GTK_LABEL (g_llibertat_6_label), TRUE);

obertura_pinca_label = gtk_label_new (_("ob. Ã§pin"));
gtk_widget_show (obertura_pinca_label);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), obertura_pinca_label, 8, 312);
gtk_widget_set_size_request (obertura_pinca_label, 56, 16);
gtk_label_set_justify (GTK_LABEL (obertura_pinca_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (obertura_pinca_label), 0, 0.5);

g_llibertat_1_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (g_llibertat_1_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_1_hscale, 64, 112);
gtk_widget_set_size_request (g_llibertat_1_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (g_llibertat_1_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (g_llibertat_1_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (g_llibertat_1_hscale), 2);

g_llibertat_2_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (g_llibertat_2_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_2_hscale, 64, 144);
gtk_widget_set_size_request (g_llibertat_2_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (g_llibertat_2_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (g_llibertat_2_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (g_llibertat_2_hscale), 2);

g_llibertat_3_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (g_llibertat_3_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_3_hscale, 64, 176);
gtk_widget_set_size_request (g_llibertat_3_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (g_llibertat_3_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (g_llibertat_3_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (g_llibertat_3_hscale), 2);

g_llibertat_4_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (g_llibertat_4_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_4_hscale, 64, 208);
gtk_widget_set_size_request (g_llibertat_4_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (g_llibertat_4_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (g_llibertat_4_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (g_llibertat_4_hscale), 2);
```

```
g_llibertat_5_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (g_llibertat_5_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_5_hscale, 64, 240);
gtk_widget_set_size_request (g_llibertat_5_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (g_llibertat_5_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (g_llibertat_5_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (g_llibertat_5_hscale), 2);

g_llibertat_6_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (g_llibertat_6_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_6_hscale, 64, 272);
gtk_widget_set_size_request (g_llibertat_6_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (g_llibertat_6_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (g_llibertat_6_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (g_llibertat_6_hscale), 2);

obertura_pinca_hscale = gtk_hscale_new (GTK_ADJUSTMENT (gtk_adjustment_new (0, -3.14, 3.14, 0, 0, 0)));
gtk_widget_show (obertura_pinca_hscale);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), obertura_pinca_hscale, 64, 304);
gtk_widget_set_size_request (obertura_pinca_hscale, 80, 37);
gtk_scale_set_draw_value (GTK_SCALE (obertura_pinca_hscale), FALSE);
gtk_scale_set_value_pos (GTK_SCALE (obertura_pinca_hscale), GTK_POS_BOTTOM);
gtk_scale_set_digits (GTK_SCALE (obertura_pinca_hscale), 2);


vseparator1 = gtk_vseparator_new ();
gtk_widget_show (vseparator1);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), vseparator1, 80, 8);
gtk_widget_set_size_request (vseparator1, 17, 66);

esq_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("esq"));
gtk_widget_show (esq_radiobutton);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), esq_radiobutton, 96, 32);
gtk_widget_set_size_request (esq_radiobutton, 48, 24);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (esq_radiobutton), esq_radiobutton_group);
esq_radiobutton_group = gtk_radio_button_get_group (GTK_RADIO_BUTTON (esq_radiobutton));

dret_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("dret"));
gtk_widget_show (dret_radiobutton);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), dret_radiobutton, 160, 32);
gtk_widget_set_size_request (dret_radiobutton, 48, 24);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (dret_radiobutton), esq_radiobutton_group);
esq_radiobutton_group = gtk_radio_button_get_group (GTK_RADIO_BUTTON (dret_radiobutton));

g_llibertat_1_entry = gtk_entry_new ();
gtk_widget_show (g_llibertat_1_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_1_entry, 160, 120);
gtk_widget_set_size_request (g_llibertat_1_entry, 50, 24);

g_llibertat_2_entry = gtk_entry_new ();
gtk_widget_show (g_llibertat_2_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_2_entry, 160, 152);
gtk_widget_set_size_request (g_llibertat_2_entry, 50, 24);

g_llibertat_3_entry = gtk_entry_new ();
gtk_widget_show (g_llibertat_3_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_3_entry, 160, 184);
gtk_widget_set_size_request (g_llibertat_3_entry, 50, 24);

g_llibertat_4_entry = gtk_entry_new ();
gtk_widget_show (g_llibertat_4_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_4_entry, 160, 216);
gtk_widget_set_size_request (g_llibertat_4_entry, 50, 24);

g_llibertat_5_entry = gtk_entry_new ();
gtk_widget_show (g_llibertat_5_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_5_entry, 160, 248);
gtk_widget_set_size_request (g_llibertat_5_entry, 50, 24);

g_llibertat_6_entry = gtk_entry_new ();
gtk_widget_show (g_llibertat_6_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), g_llibertat_6_entry, 160, 280);
gtk_widget_set_size_request (g_llibertat_6_entry, 50, 24);

obertura_pinca_entry = gtk_entry_new ();
gtk_widget_show (obertura_pinca_entry);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), obertura_pinca_entry, 160, 312);
gtk_widget_set_size_request (obertura_pinca_entry, 50, 24);


home_button = gtk_button_new_with_mnemonic (_("home"));
gtk_widget_show (home_button);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), home_button, 80, 344);
gtk_widget_set_size_request (home_button, 53, 26);

prensio_on_line_CheckButton = gtk_check_button_new_with_mnemonic (_("prensio_on-line"));
gtk_widget_show (prensio_on_line_CheckButton);
gtk_layout_put (GTK_LAYOUT (g_llibertat_layout), prensio_on_line_CheckButton, 96, 80);
gtk_widget_set_size_request (prensio_on_line_CheckButton, 124, 20);
```

```
if(NEW(g_sis_hscale,1,SIS_WIDGETS)==NULL)
   cerror("Error: en malloc en create_graus_llibertat_window()");
if(NEW(g_sis_label,1,SIS_WIDGETS)==NULL)
   cerror("Error: en malloc en create_graus_llibertat_window()");
if(NEW(g_tres_toggle,1,TRES_TOGGLE)==NULL)
 cerror("Error: en malloc en create_graus_llibertat_window()");


g_tres_toggle->robot_togglebutton=g_llibertat_robot_radiobutton;
g_tres_toggle->prensio_togglebutton=g_llibertat_prensio_radiobutton;
g_tres_toggle->pinca_togglebutton=g_llibertat_pinca_radiobutton;



g_sis_hscale->g_widget_1=g_llibertat_1_hscale;
g_sis_hscale->g_widget_2=g_llibertat_2_hscale;
g_sis_hscale->g_widget_3=g_llibertat_3_hscale;
g_sis_hscale->g_widget_4=g_llibertat_4_hscale;
g_sis_hscale->g_widget_5=g_llibertat_5_hscale;
g_sis_hscale->g_widget_6=g_llibertat_6_hscale;

g_sis_label->g_widget_1=g_llibertat_1_label;
g_sis_label->g_widget_2=g_llibertat_2_label;
g_sis_label->g_widget_3=g_llibertat_3_label;
g_sis_label->g_widget_4=g_llibertat_4_label;
g_sis_label->g_widget_5=g_llibertat_5_label;
g_sis_label->g_widget_6=g_llibertat_6_label;

g_signal_connect ((gpointer) g_llibertat_robot_radiobutton, "toggled",
                  G_CALLBACK (on_g_llibertat_robot_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_objecte_radiobutton, "toggled",
                  G_CALLBACK (on_g_llibertat_objecte_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_prensio_radiobutton, "toggled",
                  G_CALLBACK (on_g_llibertat_prensio_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_pinca_radiobutton, "toggled",
                  G_CALLBACK (on_g_llibertat_pinca_radiobutton_toggled),
                  NULL);

g_signal_connect ((gpointer) esq_radiobutton, "toggled",
                  G_CALLBACK (on_esq_radiobutton_toggled),
                   (gpointer)g_tres_toggle);
g_signal_connect ((gpointer) dret_radiobutton, "toggled",
                  G_CALLBACK (on_dret_radiobutton_toggled),
                   (gpointer)g_tres_toggle);


//Movem els diferents graus de llibertat amb els sliders.
g_signal_connect ((gpointer) g_llibertat_1_hscale, "value_changed",
                  G_CALLBACK (on_g_llibertat_1_hscale_value_changed),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_2_hscale, "value_changed",
                  G_CALLBACK (on_g_llibertat_2_hscale_value_changed),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_3_hscale, "value_changed",
                  G_CALLBACK (on_g_llibertat_3_hscale_value_changed),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_4_hscale, "value_changed",
                  G_CALLBACK (on_g_llibertat_4_hscale_value_changed),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_5_hscale, "value_changed",
                  G_CALLBACK (on_g_llibertat_5_hscale_value_changed),
                  NULL);
g_signal_connect ((gpointer) g_llibertat_6_hscale, "value_changed",
                  G_CALLBACK (on_g_llibertat_6_hscale_value_changed),
                  NULL);

g_signal_connect ((gpointer) obertura_pinca_hscale, "value_changed",
                  G_CALLBACK (on_obertura_pinca_hscale_value_changed),
                  NULL);

g_signal_connect ((gpointer) prensio_on_line_CheckButton, "toggled",
                  G_CALLBACK (on_prensio_on_line_CheckButton_toggled),
                  NULL);

//Controlador per actualitzar els noms dels camps, si activem el boto
//toggled per moure l objecte o els robots.
g_signal_connect ((gpointer) g_llibertat_robot_radiobutton, "toggled",
                  G_CALLBACK (on_convertir_labels_a_g_llibertat_robot_radiobutton_toggled),
                   (gpointer)g_sis_label);

g_signal_connect ((gpointer) g_llibertat_objecte_radiobutton, "toggled",
                    G_CALLBACK (on_convertir_labels_a_g_llibertat_solid_radiobutton_toggled),
                   (gpointer)g_sis_label);

g_signal_connect ((gpointer) g_llibertat_prensio_radiobutton, "toggled",
                  G_CALLBACK (on_convertir_labels_a_g_llibertat_solid_radiobutton_toggled),
                   (gpointer)g_sis_label);

//Controladors de senyals creats per mi, que serveixen per actualitzar
```

```
//els valors dels Widgets Entry quan movem els Widget hScale.
gtk_signal_connect(GTK_OBJECT(g_llibertat_1_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(g_llibertat_1_entry));

gtk_signal_connect(GTK_OBJECT(g_llibertat_2_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(g_llibertat_2_entry));

gtk_signal_connect(GTK_OBJECT(g_llibertat_3_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(g_llibertat_3_entry));

gtk_signal_connect(GTK_OBJECT(g_llibertat_4_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(g_llibertat_4_entry));

gtk_signal_connect(GTK_OBJECT(g_llibertat_5_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(g_llibertat_5_entry));

gtk_signal_connect(GTK_OBJECT(g_llibertat_6_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(g_llibertat_6_entry));

gtk_signal_connect(GTK_OBJECT(obertura_pinca_hscale),"value_changed",
                   GTK_SIGNAL_FUNC(set_value_entry_by_hscale),(gpointer)(obertura_pinca_entry));

//Controladors de senyals creats per mi, que serveixen per actualitzar
//els valors dels Widgets hScale, quan entrem un valor als Widget Entry.
gtk_signal_connect(GTK_OBJECT(g_llibertat_1_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(g_llibertat_1_hscale));

gtk_signal_connect(GTK_OBJECT(g_llibertat_2_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(g_llibertat_2_hscale));

gtk_signal_connect(GTK_OBJECT(g_llibertat_3_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(g_llibertat_3_hscale));

gtk_signal_connect(GTK_OBJECT(g_llibertat_4_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(g_llibertat_4_hscale));

gtk_signal_connect(GTK_OBJECT(g_llibertat_5_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(g_llibertat_5_hscale));

gtk_signal_connect(GTK_OBJECT(g_llibertat_6_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(g_llibertat_6_hscale));

gtk_signal_connect(GTK_OBJECT(obertura_pinca_entry),"activate",
                   GTK_SIGNAL_FUNC(set_value_hscale_by_entry),(gpointer)(obertura_pinca_hscale));

//Controladors de senyals creats per mi, que serveixen per actualitzar
//els valors dels Widgets hScale, quan entrem un valor als Widget Entry de 0
//es per per portar els robots a la home position.
gtk_signal_connect(GTK_OBJECT(home_button),"clicked",
                   GTK_SIGNAL_FUNC(set_value_hscale_to_0),(gpointer)(g_llibertat_1_hscale));

gtk_signal_connect(GTK_OBJECT(home_button),"clicked",
                   GTK_SIGNAL_FUNC(set_value_hscale_to_0),(gpointer)(g_llibertat_2_hscale));

gtk_signal_connect(GTK_OBJECT(home_button),"clicked",
                   GTK_SIGNAL_FUNC(set_value_hscale_to_0),(gpointer)(g_llibertat_3_hscale));

gtk_signal_connect(GTK_OBJECT(home_button),"clicked",
                   GTK_SIGNAL_FUNC(set_value_hscale_to_0),(gpointer)(g_llibertat_4_hscale));

gtk_signal_connect(GTK_OBJECT(home_button),"clicked",
                   GTK_SIGNAL_FUNC(set_value_hscale_to_0),(gpointer)(g_llibertat_5_hscale));

gtk_signal_connect(GTK_OBJECT(home_button),"clicked",
                   GTK_SIGNAL_FUNC(set_value_hscale_to_0),(gpointer)(g_llibertat_6_hscale));

//Refresc dels valors de la scene tree als sliders.
gtk_signal_connect ((gpointer) g_llibertat_robot_radiobutton, "toggled",
                    G_CALLBACK (on_g_sis_hscale_refresc_by_left_rob_scene),(gpointer)(g_sis_hscale));
gtk_signal_connect ((gpointer) g_llibertat_prensio_radiobutton, "toggled",
                    G_CALLBACK (on_g_sis_hscale_refresc_by_left_rob_scene),(gpointer)(g_sis_hscale));
gtk_signal_connect ((gpointer) g_llibertat_pinca_radiobutton, "toggled",
                    G_CALLBACK (on_g_sis_hscale_refresc_by_left_rob_scene),(gpointer)(g_sis_hscale));
gtk_signal_connect ((gpointer) g_llibertat_objecte_radiobutton, "toggled",
                    G_CALLBACK (on_g_sis_hscale_refresc_by_object_scene),(gpointer)(g_sis_hscale));


gtk_signal_connect ((gpointer)esq_radiobutton, "toggled",
                    G_CALLBACK (on_g_sis_hscale_refresc_by_left_rob_scene),(gpointer)(g_sis_hscale));
gtk_signal_connect ((gpointer)dret_radiobutton, "toggled",
                    G_CALLBACK (on_g_sis_hscale_refresc_by_left_rob_scene),(gpointer)(g_sis_hscale));



  /* Store pointers to all widgets, for use by lookup_widget(). */
GLADE_HOOKUP_OBJECT_NO_REF (graus_llibertat_window, graus_llibertat_window,
                            "graus_llibertat_window");
GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_layout,
                     "g_llibertat_layout");
GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_robot_radiobutton,
                     "g_llibertat_robot_radiobutton");
GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_objecte_radiobutton,
```

```
                        "g_llibertat_objecte_radiobutton");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_prensio_radiobutton,
                        "g_llibertat_prensio_radiobutton");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_pinca_radiobutton,
                        "g_llibertat_pinca_radiobutton");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_1_label,
                        "g_llibertat_1_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_2_label,
                        "g_llibertat_2_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_3_label,
                        "g_llibertat_3_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_4_label,
                        "g_llibertat_4_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_5_label,
                        "g_llibertat_5_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_6_label,
                        "g_llibertat_6_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_1_hscale,
                        "g_llibertat_1_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_2_hscale,
                        "g_llibertat_2_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_3_hscale,
                        "g_llibertat_3_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_4_hscale,
                        "g_llibertat_4_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_5_hscale,
                        "g_llibertat_5_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_6_hscale,
                        "g_llibertat_6_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, vseparator1,
                        "vseparator1");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, esq_radiobutton,
                        "esq_radiobutton");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, dret_radiobutton,
                        "dret_radiobutton");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_1_entry,
                        "g_llibertat_1_entry");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_2_entry,
                        "g_llibertat_2_entry");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_3_entry,
                        "g_llibertat_3_entry");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_4_entry,
                        "g_llibertat_4_entry");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_5_entry,
                        "g_llibertat_5_entry");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, g_llibertat_6_entry,
                        "g_llibertat_6_entry");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, home_button,
                        "home_button");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, prensio_on_line_CheckButton,
                        "prensio_on_line_CheckButton");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, obertura_pinca_label,
                        "obertura_pinca_label");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, obertura_pinca_hscale,
                        "obertura_pinca_hscale");
  GLADE_HOOKUP_OBJECT (graus_llibertat_window, obertura_pinca_entry,
                        "obertura_pinca_entry");

  return graus_llibertat_window;
}

GtkWidget*
create_window_parametres (void)
{
  GtkWidget *window_parametres;
  GtkWidget *hbox3;
  GtkWidget *esquerra_layout;
  GtkWidget *fase_aprenentatge_label;
  GtkWidget *hseparator14;
  GtkWidget *max_nodes_label;
  GtkWidget *percen_expansio_label;
  GtkWidget *min_nodes_label;
  GtkWidget *num_veins_label;
  GtkWidget *max_dist_label;
  GtkWidget *cami_local_label;
  GtkWidget *pas_cami_local_label;
  GtkWidget *interpol_label;
  GSList *rob_esq_interpol_radiobutton_group = NULL;
  GtkWidget *hseparator15;
  GSList *rob_esq_aleat_radiobutton_group = NULL;
  GtkWidget *gene_aleatori_label;
  GtkWidget *fase_cerca_label;
  GtkWidget *hseparator16;
  GtkWidget *max_profund_label;
  GtkWidget *sortida_label;
  GtkWidget *cerca_sortida_hseparator;
  GtkWidget *nom_base_label;
  GtkWidget *nom_fitxer_debug_label;
  GtkWidget *desar_par_button;
  GtkWidget *num_pasos_label;
  GtkWidget *fora_dist_max_label;
  GtkWidget *max_no_cami_local_label;
```

```
GtkWidget *central_layout;
GtkWidget *hseparator21;
GtkWidget *hseparator22;
GtkWidget *hseparator23;
GtkWidget *hseparator24;
GtkWidget *checkbutton1;
GtkWidget *vseparator2;
GtkWidget *dret_layout;
GtkWidget *robot_label;
GtkWidget *hseparator18;
GtkWidget *angle_1_label;
GtkWidget *angle_2_label;
GtkWidget *angle_3_abel;
GtkWidget *angle_4_label;
GtkWidget *angle_5_label;
GtkWidget *angle_6_label;
GtkWidget *min_robot_label;
GtkWidget *max_robot_label;
GtkWidget *config_rob_esq_label;
GtkWidget *config_rob_dret_label;
GtkWidget *config_label;
GtkWidget *objecte_label;
GtkWidget *hseparator19;
GtkWidget *x_label;
GtkWidget *y_label;
GtkWidget *z_label;
GtkWidget *rot_x_label;
GtkWidget *rot_y_label;
GtkWidget *rot_z_label;
GtkWidget *min_objecte_label;
GtkWidget *max_objecte_label;
GtkWidget *prensio_label;
GtkWidget *hseparator20;
GtkWidget *min_prensio_label;
GtkWidget *max_prensio_label;
GtkWidget *label45;
GtkWidget *hseparator25;
GtkWidget *entrada_label;
GtkWidget *carregar_par_button;

if(NEW(widgets_parametres,1,WIDGETS_PARAMETRES)==NULL)
  cerror("Error: en malloc en create_window_parametres ()");

window_parametres = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_widget_set_size_request (window_parametres, 423, 665);
gtk_window_set_title (GTK_WINDOW (window_parametres), _("Parametres"));

hbox3 = gtk_hbox_new (FALSE, 0);
gtk_widget_show (hbox3);
gtk_container_add (GTK_CONTAINER (window_parametres), hbox3);

esquerra_layout = gtk_layout_new (NULL, NULL);
gtk_widget_show (esquerra_layout);
gtk_box_pack_start (GTK_BOX (hbox3), esquerra_layout, TRUE, TRUE, 0);
gtk_widget_set_size_request (esquerra_layout, 0, -2);
gtk_layout_set_size (GTK_LAYOUT (esquerra_layout), 400, 400);
GTK_ADJUSTMENT (GTK_LAYOUT (esquerra_layout)->hadjustment)->step_increment = 0;
GTK_ADJUSTMENT (GTK_LAYOUT (esquerra_layout)->vadjustment)->step_increment = 0;

fase_aprenentatge_label = gtk_label_new (_("Fase Aprenentage"));
gtk_widget_show (fase_aprenentatge_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), fase_aprenentatge_label, 64, 0);
gtk_widget_set_size_request (fase_aprenentatge_label, 127, 16);

hseparator14 = gtk_hseparator_new ();
gtk_widget_show (hseparator14);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), hseparator14, 0, 8);
gtk_widget_set_size_request (hseparator14, 230, 16);

widgets_parametres->borrar_graf_checkbutton = gtk_check_button_new_with_mnemonic (_("borrar graf"));
gtk_widget_show (widgets_parametres->borrar_graf_checkbutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->borrar_graf_checkbutton, 16, 16);
gtk_widget_set_size_request (widgets_parametres->borrar_graf_checkbutton, 100, 20);

max_nodes_label = gtk_label_new (_("max. nodes"));
gtk_widget_show (max_nodes_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), max_nodes_label, 8, 77);
gtk_widget_set_size_request (max_nodes_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (max_nodes_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (max_nodes_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (max_nodes_label), 9, 0);

percen_expansio_label = gtk_label_new (_("% expansio"));
gtk_widget_show (percen_expansio_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), percen_expansio_label, 8, 109);
gtk_widget_set_size_request (percen_expansio_label, 86, 16);
gtk_label_set_justify (GTK_LABEL (percen_expansio_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (percen_expansio_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (percen_expansio_label), 9, 0);

min_nodes_label = gtk_label_new (_("min. nodes"));
gtk_widget_show (min_nodes_label);
```

```
gtk_layout_put (GTK_LAYOUT (esquerra_layout), min_nodes_label, 8, 141);
gtk_widget_set_size_request (min_nodes_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (min_nodes_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (min_nodes_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (min_nodes_label), 9, 0);

num_veins_label = gtk_label_new (_("num. veins"));
gtk_widget_show (num_veins_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), num_veins_label, 8, 173);
gtk_widget_set_size_request (num_veins_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (num_veins_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (num_veins_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (num_veins_label), 9, 0);

max_dist_label = gtk_label_new (_("max. dist"));
gtk_widget_show (max_dist_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), max_dist_label, 8, 205);
gtk_widget_set_size_request (max_dist_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (max_dist_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (max_dist_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (max_dist_label), 9, 0);

widgets_parametres->nom_base_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->nom_base_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->nom_base_entry, 96, 40);
gtk_widget_set_size_request (widgets_parametres->nom_base_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->nom_base_entry), _("problema"));


widgets_parametres->max_nodes_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->max_nodes_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->max_nodes_entry, 96, 72);
gtk_widget_set_size_request (widgets_parametres->max_nodes_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->max_nodes_entry), _("50"));

widgets_parametres->percen_expansio_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->percen_expansio_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->percen_expansio_entry, 96, 104);
gtk_widget_set_size_request (widgets_parametres->percen_expansio_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->percen_expansio_entry), _("30"));

widgets_parametres->min_nodes_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->min_nodes_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->min_nodes_entry, 96, 136);
gtk_widget_set_size_request (widgets_parametres->min_nodes_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->min_nodes_entry), _("100"));

widgets_parametres->num_veins_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->num_veins_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->num_veins_entry, 96, 168);
gtk_widget_set_size_request (widgets_parametres->num_veins_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->num_veins_entry), _("30"));

widgets_parametres->max_dist_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->max_dist_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->max_dist_entry, 96, 200);
gtk_widget_set_size_request (widgets_parametres->max_dist_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->max_dist_entry), _("3.3"));

cami_local_label = gtk_label_new (_("Cami Local"));
gtk_widget_show (cami_local_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), cami_local_label, 72, 296);
gtk_widget_set_size_request (cami_local_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (cami_local_label), GTK_JUSTIFY_LEFT);

pas_cami_local_label = gtk_label_new (_("pas cami\nlocal"));
gtk_widget_show (pas_cami_local_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), pas_cami_local_label, 8, 320);
gtk_widget_set_size_request (pas_cami_local_label, 81, 32);
gtk_label_set_justify (GTK_LABEL (pas_cami_local_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (pas_cami_local_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (pas_cami_local_label), 9, 0);

widgets_parametres->pas_cami_local_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->pas_cami_local_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->pas_cami_local_entry, 96, 320);
gtk_widget_set_size_request (widgets_parametres->pas_cami_local_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->pas_cami_local_entry), _("0.045"));

interpol_label = gtk_label_new (_("interpol."));
gtk_widget_show (interpol_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), interpol_label, 8, 360);
gtk_widget_set_size_request (interpol_label, 56, 16);
gtk_label_set_justify (GTK_LABEL (interpol_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (interpol_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (interpol_label), 9, 0);

widgets_parametres->rob_esq_interpol_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("rob. esq"));
gtk_widget_show (widgets_parametres->rob_esq_interpol_radiobutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->rob_esq_interpol_radiobutton, 96, 344);
gtk_widget_set_size_request (widgets_parametres->rob_esq_interpol_radiobutton, 93, 20);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (widgets_parametres->rob_esq_interpol_radiobutton),
```

```
                                rob_esq_interpol_radiobutton_group);
rob_esq_interpol_radiobutton_group = gtk_radio_button_get_group
  (GTK_RADIO_BUTTON (widgets_parametres->rob_esq_interpol_radiobutton));
gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widgets_parametres->rob_esq_interpol_radiobutton), TRUE);

widgets_parametres->rob_dret_interpol_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("rob. dret"));
gtk_widget_show (widgets_parametres->rob_dret_interpol_radiobutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->rob_dret_interpol_radiobutton, 96, 368);
gtk_widget_set_size_request (widgets_parametres->rob_dret_interpol_radiobutton, 93, 20);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (widgets_parametres->rob_dret_interpol_radiobutton),
                                rob_esq_interpol_radiobutton_group);
rob_esq_interpol_radiobutton_group = gtk_radio_button_get_group
  (GTK_RADIO_BUTTON (widgets_parametres->rob_dret_interpol_radiobutton));

hseparator15 = gtk_hseparator_new ();
gtk_widget_show (hseparator15);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), hseparator15,0, 304);
gtk_widget_set_size_request (hseparator15, 230, 16);

widgets_parametres->rob_esq_aleat_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("rob. esq"));
gtk_widget_show (widgets_parametres->rob_esq_aleat_radiobutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout),
                widgets_parametres->rob_esq_aleat_radiobutton, 96, 224);
gtk_widget_set_size_request (widgets_parametres->rob_esq_aleat_radiobutton, 93, 20);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (widgets_parametres->rob_esq_aleat_radiobutton),
                                rob_esq_aleat_radiobutton_group);
rob_esq_aleat_radiobutton_group = gtk_radio_button_get_group
  (GTK_RADIO_BUTTON (widgets_parametres->rob_esq_aleat_radiobutton));
gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widgets_parametres->rob_esq_aleat_radiobutton), TRUE);

widgets_parametres->rob_dret_aleat_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("rob. dret"));
gtk_widget_show (widgets_parametres->rob_dret_aleat_radiobutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->rob_dret_aleat_radiobutton, 96, 248);
gtk_widget_set_size_request (widgets_parametres->rob_dret_aleat_radiobutton, 93, 20);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (widgets_parametres->rob_dret_aleat_radiobutton),
                                rob_esq_aleat_radiobutton_group);
rob_esq_aleat_radiobutton_group = gtk_radio_button_get_group
  (GTK_RADIO_BUTTON (widgets_parametres->rob_dret_aleat_radiobutton));

widgets_parametres->objecte_aleat_radiobutton = gtk_radio_button_new_with_mnemonic (NULL, _("objecte"));
gtk_widget_show (widgets_parametres->objecte_aleat_radiobutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->objecte_aleat_radiobutton, 96, 272);
gtk_widget_set_size_request (widgets_parametres->objecte_aleat_radiobutton, 93, 20);
gtk_radio_button_set_group (GTK_RADIO_BUTTON (widgets_parametres->objecte_aleat_radiobutton),
                                rob_esq_aleat_radiobutton_group);
rob_esq_aleat_radiobutton_group = gtk_radio_button_get_group
  (GTK_RADIO_BUTTON (widgets_parametres->objecte_aleat_radiobutton));

gene_aleatori_label = gtk_label_new (_("generacio\naleatoria\n"));
gtk_widget_show (gene_aleatori_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), gene_aleatori_label, 8, 240);
gtk_widget_set_size_request (gene_aleatori_label, 81, 34);
gtk_label_set_justify (GTK_LABEL (gene_aleatori_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (gene_aleatori_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (gene_aleatori_label), 9, 0);

fase_cerca_label = gtk_label_new (_("Fase Cerca"));
gtk_widget_show (fase_cerca_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), fase_cerca_label, 64, 384);
gtk_widget_set_size_request (fase_cerca_label, 83, 16);
gtk_label_set_justify (GTK_LABEL (fase_cerca_label), GTK_JUSTIFY_LEFT);

hseparator16 = gtk_hseparator_new ();
gtk_widget_show (hseparator16);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), hseparator16, 0, 392);
gtk_widget_set_size_request (hseparator16, 230, 16);

max_profund_label = gtk_label_new (_("max. profund."));
gtk_widget_show (max_profund_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), max_profund_label, 0, 508);
gtk_widget_set_size_request (max_profund_label, 86, 16);
gtk_label_set_justify (GTK_LABEL (max_profund_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (max_profund_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (max_profund_label), 9, 0);

sortida_label = gtk_label_new (_("Sortida"));
gtk_widget_show (sortida_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), sortida_label, 80, 536);
gtk_widget_set_size_request (sortida_label, 56, 16);
gtk_label_set_justify (GTK_LABEL (sortida_label), GTK_JUSTIFY_LEFT);

cerca_sortida_hseparator = gtk_hseparator_new ();
gtk_widget_show (cerca_sortida_hseparator);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), cerca_sortida_hseparator, 0, 544);
gtk_widget_set_size_request (cerca_sortida_hseparator, 238, 16);

widgets_parametres->fitxer_debug_checkbutton = gtk_check_button_new_with_mnemonic (_("debugar"));
gtk_widget_show (widgets_parametres->fitxer_debug_checkbutton);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->fitxer_debug_checkbutton, 8, 569);
gtk_widget_set_size_request (widgets_parametres->fitxer_debug_checkbutton, 74, 37);

widgets_parametres->nom_fitxer_debug_entry = gtk_entry_new ();
```

```
gtk_widget_show (widgets_parametres->nom_fitxer_debug_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->nom_fitxer_debug_entry, 96, 576);
gtk_widget_set_size_request (widgets_parametres->nom_fitxer_debug_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->nom_fitxer_debug_entry), _("depurar"));

nom_fitxer_debug_label = gtk_label_new (_("nom_fitxer"));
gtk_widget_show (nom_fitxer_debug_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), nom_fitxer_debug_label, 104, 552);
gtk_widget_set_size_request (nom_fitxer_debug_label, 75, 16);
gtk_label_set_justify (GTK_LABEL (nom_fitxer_debug_label), GTK_JUSTIFY_LEFT);

desar_par_button = gtk_button_new_with_mnemonic (_("desar_*.par"));
gtk_widget_show (desar_par_button);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), desar_par_button, 56, 632);
gtk_widget_set_size_request (desar_par_button, 88, 25);

nom_base_label = gtk_label_new (_("nom_base"));
gtk_widget_show (nom_base_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), nom_base_label, 8, 45);
gtk_widget_set_size_request (nom_base_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (nom_base_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (nom_base_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (nom_base_label), 9, 0);

widgets_parametres->num_pasos_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->num_pasos_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->num_pasos_entry, 96, 408);
gtk_widget_set_size_request (widgets_parametres->num_pasos_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->num_pasos_entry), _("20"));

num_pasos_label = gtk_label_new (_("num._pasos"));
gtk_widget_show (num_pasos_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), num_pasos_label, 0, 412);
gtk_widget_set_size_request (num_pasos_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (num_pasos_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (num_pasos_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (num_pasos_label), 9, 0);


fora_dist_max_label = gtk_label_new (_("fora_dist_max."));
gtk_widget_show (fora_dist_max_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), fora_dist_max_label, 0, 443);
gtk_widget_set_size_request (fora_dist_max_label, 96, 16);
gtk_label_set_justify (GTK_LABEL (fora_dist_max_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (fora_dist_max_label), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (fora_dist_max_label), 9, 0);

central_layout = gtk_layout_new (NULL, NULL);
gtk_widget_show (central_layout);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), central_layout, 0, 0);
gtk_widget_set_size_request (central_layout, 0, 0);
gtk_layout_set_size (GTK_LAYOUT (central_layout), 400, 400);
GTK_ADJUSTMENT (GTK_LAYOUT (central_layout)->hadjustment)->step_increment = 0;
GTK_ADJUSTMENT (GTK_LAYOUT (central_layout)->vadjustment)->step_increment = 0;

hseparator21 = gtk_hseparator_new ();
gtk_widget_show (hseparator21);
gtk_layout_put (GTK_LAYOUT (central_layout), hseparator21, 0, 8);
gtk_widget_set_size_request (hseparator21, 230, 16);

hseparator22 = gtk_hseparator_new ();
gtk_widget_show (hseparator22);
gtk_layout_put (GTK_LAYOUT (central_layout), hseparator22, 8, 264);
gtk_widget_set_size_request (hseparator22, 230, 16);

hseparator23 = gtk_hseparator_new ();
gtk_widget_show (hseparator23);
gtk_layout_put (GTK_LAYOUT (central_layout), hseparator23, 0, 360);
gtk_widget_set_size_request (hseparator23, 230, 16);

hseparator24 = gtk_hseparator_new ();
gtk_widget_show (hseparator24);
gtk_layout_put (GTK_LAYOUT (central_layout), hseparator24, 0, 560);
gtk_widget_set_size_request (hseparator24, 230, 16);

checkbutton1 = gtk_check_button_new_with_mnemonic (_("guardar_fitxer\ndebug"));
gtk_widget_show (checkbutton1);
gtk_layout_put (GTK_LAYOUT (central_layout), checkbutton1, 16, 608);
gtk_widget_set_size_request (checkbutton1, 107, 37);

widgets_parametres->max_profunditat_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->max_profunditat_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->max_profunditat_entry, 96, 504);
gtk_widget_set_size_request (widgets_parametres->max_profunditat_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->max_profunditat_entry), _("10"));

widgets_parametres->fora_dist_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->fora_dist_max_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->fora_dist_max_entry, 96, 440);
gtk_widget_set_size_request (widgets_parametres->fora_dist_max_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->fora_dist_max_entry), _("100"));
```

```
max_no_cami_local_label = gtk_label_new (_("max.no_local"));
gtk_widget_show (max_no_cami_local_label);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), max_no_cami_local_label, 8, 477);
gtk_widget_set_size_request (max_no_cami_local_label, 81, 16);
gtk_label_set_justify (GTK_LABEL (max_no_cami_local_label), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (max_no_cami_local_label), 0, 0.5);

widgets_parametres->max_no_local_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->max_no_local_entry);
gtk_layout_put (GTK_LAYOUT (esquerra_layout), widgets_parametres->max_no_local_entry, 96, 472);
gtk_widget_set_size_request (widgets_parametres->max_no_local_entry, 86, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->max_no_local_entry), _("20"));

vseparator2 = gtk_vseparator_new ();
gtk_widget_show (vseparator2);
gtk_box_pack_start (GTK_BOX (hbox3), vseparator2, FALSE, TRUE, 0);

dret_layout = gtk_layout_new (NULL, NULL);
gtk_widget_show (dret_layout);
gtk_box_pack_start (GTK_BOX (hbox3), dret_layout, TRUE, TRUE, 0);
gtk_layout_set_size (GTK_LAYOUT (dret_layout), 400, 400);
GTK_ADJUSTMENT (GTK_LAYOUT (dret_layout)->hadjustment)->step_increment = 0;
GTK_ADJUSTMENT (GTK_LAYOUT (dret_layout)->vadjustment)->step_increment = 0;

robot_label = gtk_label_new (_("Robot"));
gtk_widget_show (robot_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), robot_label, 88, 0);
gtk_widget_set_size_request (robot_label, 49, 16);
gtk_label_set_justify (GTK_LABEL (robot_label), GTK_JUSTIFY_LEFT);

hseparator18 = gtk_hseparator_new ();
gtk_widget_show (hseparator18);
gtk_layout_put (GTK_LAYOUT (dret_layout), hseparator18, 0, 8);
gtk_widget_set_size_request (hseparator18, 230, 16);

angle_1_label = gtk_label_new (_("angle_1"));
gtk_widget_show (angle_1_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), angle_1_label, 8, 48);
gtk_widget_set_size_request (angle_1_label, 45, 16);
gtk_label_set_justify (GTK_LABEL (angle_1_label), GTK_JUSTIFY_LEFT);

angle_2_label = gtk_label_new (_("angle_2"));
gtk_widget_show (angle_2_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), angle_2_label, 8, 84);
gtk_widget_set_size_request (angle_2_label, 45, 16);
gtk_label_set_justify (GTK_LABEL (angle_2_label), GTK_JUSTIFY_LEFT);

angle_3_abel = gtk_label_new (_("angle_3"));
gtk_widget_show (angle_3_abel);
gtk_layout_put (GTK_LAYOUT (dret_layout), angle_3_abel, 8, 117);
gtk_widget_set_size_request (angle_3_abel, 45, 16);
gtk_label_set_justify (GTK_LABEL (angle_3_abel), GTK_JUSTIFY_LEFT);

angle_4_label = gtk_label_new (_("angle_4"));
gtk_widget_show (angle_4_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), angle_4_label, 8, 148);
gtk_widget_set_size_request (angle_4_label, 45, 16);
gtk_label_set_justify (GTK_LABEL (angle_4_label), GTK_JUSTIFY_LEFT);

angle_5_label = gtk_label_new (_("angle_5"));
gtk_widget_show (angle_5_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), angle_5_label, 8, 180);
gtk_widget_set_size_request (angle_5_label, 45, 16);
gtk_label_set_justify (GTK_LABEL (angle_5_label), GTK_JUSTIFY_LEFT);

angle_6_label = gtk_label_new (_("angle_6"));
gtk_widget_show (angle_6_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), angle_6_label, 8, 211);
gtk_widget_set_size_request (angle_6_label, 45, 16);
gtk_label_set_justify (GTK_LABEL (angle_6_label), GTK_JUSTIFY_LEFT);

min_robot_label = gtk_label_new (_("min."));
gtk_widget_show (min_robot_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), min_robot_label, 64, 24);
gtk_widget_set_size_request (min_robot_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (min_robot_label), GTK_JUSTIFY_LEFT);

max_robot_label = gtk_label_new (_("max."));
gtk_widget_show (max_robot_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), max_robot_label, 152, 24);
gtk_widget_set_size_request (max_robot_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (max_robot_label), GTK_JUSTIFY_LEFT);

widgets_parametres->angle_1_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_1_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_1_min_entry, 64, 48);
gtk_widget_set_size_request (widgets_parametres->angle_1_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_1_min_entry), _("-2.7925"));

widgets_parametres->angle_1_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_1_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_1_max_entry, 144, 48);
```

```
gtk_widget_set_size_request (widgets_parametres->angle_1_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_1_max_entry), _("2.7925"));

widgets_parametres->angle_2_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_2_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_2_min_entry, 64, 80);
gtk_widget_set_size_request (widgets_parametres->angle_2_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_2_min_entry), _("-2.2253"));

widgets_parametres->angle_2_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_2_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_2_max_entry, 144, 80);
gtk_widget_set_size_request (widgets_parametres->angle_2_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_2_max_entry), _("2.2253"));

widgets_parametres->angle_3_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_3_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_3_min_entry, 64, 112);
gtk_widget_set_size_request (widgets_parametres->angle_3_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_3_min_entry), _("-2.3475"));

widgets_parametres->angle_4_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_4_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_4_max_entry, 144, 144);
gtk_widget_set_size_request (widgets_parametres->angle_4_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_4_max_entry), _("3.1415"));

widgets_parametres->angle_4_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_4_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_4_min_entry, 64, 144);
gtk_widget_set_size_request (widgets_parametres->angle_4_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_4_min_entry), _("-3.1415"));

widgets_parametres->angle_3_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_3_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_3_max_entry, 144, 112);
gtk_widget_set_size_request (widgets_parametres->angle_3_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_3_max_entry), _("2.3475"));

widgets_parametres->angle_6_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_6_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_6_max_entry, 144, 208);
gtk_widget_set_size_request (widgets_parametres->angle_6_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_6_max_entry), _("3.1415"));

widgets_parametres->angle_6_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_6_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_6_min_entry, 64, 208);
gtk_widget_set_size_request (widgets_parametres->angle_6_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_6_min_entry), _("-3.1415"));

widgets_parametres->angle_5_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_5_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_5_max_entry, 144, 176);
gtk_widget_set_size_request (widgets_parametres->angle_5_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_5_max_entry), _("2.1031"));

widgets_parametres->angle_5_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->angle_5_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->angle_5_min_entry, 64, 176);
gtk_widget_set_size_request (widgets_parametres->angle_5_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->angle_5_min_entry), _("-1.7"));

config_rob_esq_label = gtk_label_new (_("rob._esq"));
gtk_widget_show (config_rob_esq_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), config_rob_esq_label, 64, 240);
gtk_widget_set_size_request (config_rob_esq_label, 51, 16);
gtk_label_set_justify (GTK_LABEL (config_rob_esq_label), GTK_JUSTIFY_LEFT);

config_rob_dret_label = gtk_label_new (_("rob._dret"));
gtk_widget_show (config_rob_dret_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), config_rob_dret_label, 144, 240);
gtk_widget_set_size_request (config_rob_dret_label, 52, 16);
gtk_label_set_justify (GTK_LABEL (config_rob_dret_label), GTK_JUSTIFY_LEFT);

config_label = gtk_label_new (_("config."));
gtk_widget_show (config_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), config_label, 8, 268);
gtk_widget_set_size_request (config_label, 40, 16);
gtk_label_set_justify (GTK_LABEL (config_label), GTK_JUSTIFY_LEFT);

widgets_parametres->config_rob_esq_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->config_rob_esq_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->config_rob_esq_entry, 64, 264);
gtk_widget_set_size_request (widgets_parametres->config_rob_esq_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->config_rob_esq_entry), _("run"));

widgets_parametres->config_rob_dret_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->config_rob_dret_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->config_rob_dret_entry, 144, 264);
gtk_widget_set_size_request (widgets_parametres->config_rob_dret_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->config_rob_dret_entry), _("run"));
```

```
objecte_label = gtk_label_new (_("Objecte"));
gtk_widget_show (objecte_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), objecte_label, 88, 296);
gtk_widget_set_size_request (objecte_label, 63, 16);
gtk_label_set_justify (GTK_LABEL (objecte_label), GTK_JUSTIFY_LEFT);

hseparator19 = gtk_hseparator_new ();
gtk_widget_show (hseparator19);
gtk_layout_put (GTK_LAYOUT (dret_layout), hseparator19, 0, 304);
gtk_widget_set_size_request (hseparator19, 230, 16);

x_label = gtk_label_new (_("x"));
gtk_widget_show (x_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), x_label, 0, 348);
gtk_widget_set_size_request (x_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (x_label), GTK_JUSTIFY_LEFT);

y_label = gtk_label_new (_("y"));
gtk_widget_show (y_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), y_label, 0, 380);
gtk_widget_set_size_request (y_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (y_label), GTK_JUSTIFY_LEFT);

z_label = gtk_label_new (_("z"));
gtk_widget_show (z_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), z_label, 0, 412);
gtk_widget_set_size_request (z_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (z_label), GTK_JUSTIFY_LEFT);

rot_x_label = gtk_label_new (_("rot_x"));
gtk_widget_show (rot_x_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), rot_x_label, 0, 445);
gtk_widget_set_size_request (rot_x_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (rot_x_label), GTK_JUSTIFY_LEFT);

rot_y_label = gtk_label_new (_("rot_y"));
gtk_widget_show (rot_y_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), rot_y_label, 0, 477);
gtk_widget_set_size_request (rot_y_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (rot_y_label), GTK_JUSTIFY_LEFT);

rot_z_label = gtk_label_new (_("rot_z"));
gtk_widget_show (rot_z_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), rot_z_label, 0, 509);
gtk_widget_set_size_request (rot_z_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (rot_z_label), GTK_JUSTIFY_LEFT);

min_objecte_label = gtk_label_new (_("min."));
gtk_widget_show (min_objecte_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), min_objecte_label, 64, 320);
gtk_widget_set_size_request (min_objecte_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (min_objecte_label), GTK_JUSTIFY_LEFT);

max_objecte_label = gtk_label_new (_("max."));
gtk_widget_show (max_objecte_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), max_objecte_label, 152, 320);
gtk_widget_set_size_request (max_objecte_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (max_objecte_label), GTK_JUSTIFY_LEFT);

widgets_parametres->rot_z_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->rot_z_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->rot_z_max_entry, 144, 504);
gtk_widget_set_size_request (widgets_parametres->rot_z_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->rot_z_max_entry), _("3.1415"));

widgets_parametres->rot_z_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->rot_z_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->rot_z_min_entry, 64, 504);
gtk_widget_set_size_request (widgets_parametres->rot_z_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->rot_z_min_entry), _("-3.1415"));

widgets_parametres->rot_y_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->rot_y_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->rot_y_max_entry, 144, 472);
gtk_widget_set_size_request (widgets_parametres->rot_y_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->rot_y_max_entry), _("3.1415"));

widgets_parametres->rot_y_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->rot_y_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->rot_y_min_entry, 64, 472);
gtk_widget_set_size_request (widgets_parametres->rot_y_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->rot_y_min_entry), _("-3.1415"));

widgets_parametres->rot_x_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->rot_x_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->rot_x_max_entry, 144, 440);
gtk_widget_set_size_request (widgets_parametres->rot_x_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->rot_x_max_entry), _("3.1415"));

widgets_parametres->rot_x_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->rot_x_min_entry);
```

```
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->rot_x_min_entry, 64, 440);
gtk_widget_set_size_request (widgets_parametres->rot_x_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->rot_x_min_entry), _("-3.1415"));

widgets_parametres->z_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->z_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->z_max_entry, 144, 408);
gtk_widget_set_size_request (widgets_parametres->z_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->z_max_entry), _("0.6"));

widgets_parametres->z_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->z_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->z_min_entry, 64, 408);
gtk_widget_set_size_request (widgets_parametres->z_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->z_min_entry), _("-0.6"));

widgets_parametres->y_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->y_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->y_max_entry, 144, 376);
gtk_widget_set_size_request (widgets_parametres->y_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->y_max_entry), _("-0.075"));

widgets_parametres->y_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->y_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->y_min_entry, 64, 376);
gtk_widget_set_size_request (widgets_parametres->y_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->y_min_entry), _("-1"));

widgets_parametres->x_max_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->x_max_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->x_max_entry, 144, 344);
gtk_widget_set_size_request (widgets_parametres->x_max_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->x_max_entry), _("0.34"));

widgets_parametres->x_min_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->x_min_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->x_min_entry, 64, 344);
gtk_widget_set_size_request (widgets_parametres->x_min_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->x_min_entry), _("-0.34"));

prensio_label = gtk_label_new (_("Prensio"));
gtk_widget_show (prensio_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), prensio_label, 88, 536);
gtk_widget_set_size_request (prensio_label, 62, 16);
gtk_label_set_justify (GTK_LABEL (prensio_label), GTK_JUSTIFY_LEFT);

hseparator20 = gtk_hseparator_new ();
gtk_widget_show (hseparator20);
gtk_layout_put (GTK_LAYOUT (dret_layout), hseparator20, 0, 544);
gtk_widget_set_size_request (hseparator20, 215, 16);

widgets_parametres->coord_cart_min_prensio_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->coord_cart_min_prensio_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->coord_cart_min_prensio_entry, 64, 576);
gtk_widget_set_size_request (widgets_parametres->coord_cart_min_prensio_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->coord_cart_min_prensio_entry), _("-0.5"));

widgets_parametres->coord_cart_max_prensio_entry = gtk_entry_new ();
gtk_widget_show (widgets_parametres->coord_cart_max_prensio_entry);
gtk_layout_put (GTK_LAYOUT (dret_layout), widgets_parametres->coord_cart_max_prensio_entry, 144, 576);
gtk_widget_set_size_request (widgets_parametres->coord_cart_max_prensio_entry, 55, 24);
gtk_entry_set_text (GTK_ENTRY (widgets_parametres->coord_cart_max_prensio_entry), _("0.5"));

min_prensio_label = gtk_label_new (_("min."));
gtk_widget_show (min_prensio_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), min_prensio_label, 72, 552);
gtk_widget_set_size_request (min_prensio_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (min_prensio_label), GTK_JUSTIFY_LEFT);

max_prensio_label = gtk_label_new (_("max."));
gtk_widget_show (max_prensio_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), max_prensio_label, 152, 552);
gtk_widget_set_size_request (max_prensio_label, 41, 16);
gtk_label_set_justify (GTK_LABEL (max_prensio_label), GTK_JUSTIFY_LEFT);

label45 = gtk_label_new (_("coord.\ncart."));
gtk_widget_show (label45);
gtk_layout_put (GTK_LAYOUT (dret_layout), label45, 0, 568);
gtk_widget_set_size_request (label45, 50, 34);
gtk_label_set_justify (GTK_LABEL (label45), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (label45), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (label45), 9, 0);

hseparator25 = gtk_hseparator_new ();
gtk_widget_show (hseparator25);
gtk_layout_put (GTK_LAYOUT (dret_layout), hseparator25, 0, 616);
gtk_widget_set_size_request (hseparator25, 208, 16);

entrada_label = gtk_label_new (_("Entrada"));
gtk_widget_show (entrada_label);
gtk_layout_put (GTK_LAYOUT (dret_layout), entrada_label, 88, 608);
gtk_widget_set_size_request (entrada_label, 64, 16);
```

```
gtk_label_set_justify (GTK_LABEL (entrada_label), GTK_JUSTIFY_LEFT);

carregar_par_button = gtk_button_new_with_mnemonic (_("carregar *.par"));
gtk_widget_show (carregar_par_button);
gtk_layout_put (GTK_LAYOUT (dret_layout), carregar_par_button, 72, 632);
gtk_widget_set_size_request (carregar_par_button, 96, 25);


g_signal_connect ((gpointer) widgets_parametres->borrar_graf_checkbutton, "toggled",
                  G_CALLBACK (on_borrar_graf_checkbutton_toggled),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->nom_base_entry, "activate",
                  G_CALLBACK (on_nom_base_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->max_nodes_entry, "activate",
                  G_CALLBACK (on_max_nodes_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->percen_expansio_entry, "activate",
                  G_CALLBACK (on_percen_expansio_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->min_nodes_entry, "activate",
                  G_CALLBACK (on_min_nodes_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->num_veins_entry, "activate",
                  G_CALLBACK (on_num_veins_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->max_dist_entry, "activate",
                  G_CALLBACK (on_max_dist_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->pas_cami_local_entry, "activate",
                  G_CALLBACK (on_pas_cami_local_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->rob_esq_interpol_radiobutton, "toggled",
                  G_CALLBACK (on_rob_esq_interpol_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->rob_dret_interpol_radiobutton, "toggled",
                  G_CALLBACK (on_rob_dret_interpol_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->rob_esq_aleat_radiobutton, "toggled",
                  G_CALLBACK (on_rob_esq_aleat_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->rob_dret_aleat_radiobutton, "toggled",
                  G_CALLBACK (on_rob_dret_aleat_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->objecte_aleat_radiobutton, "toggled",
                  G_CALLBACK (on_objecte_aleat_radiobutton_toggled),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->num_pasos_entry, "activate",
                  G_CALLBACK (on_num_pasos_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->max_profunditat_entry, "activate",
                  G_CALLBACK (on_max_profunditat_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->fora_dist_max_entry, "activate",
                  G_CALLBACK (on_fora_dist_max_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->max_no_local_entry, "activate",
                  G_CALLBACK (on_max_no_local_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->nom_fitxer_debug_entry, "activate",
                  G_CALLBACK (on_nom_fitxer_debug_entry_activate),
                  NULL);
g_signal_connect ((gpointer) desar_par_button, "clicked",
                  G_CALLBACK (on_desar_par_button_clicked),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_1_min_entry, "activate",
                  G_CALLBACK (on_angle_1_min_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_1_max_entry, "activate",
                  G_CALLBACK (on_angle_1_max_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_2_min_entry, "activate",
                  G_CALLBACK (on_angle_2_min_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_2_max_entry, "activate",
                  G_CALLBACK (on_angle_2_max_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_3_min_entry, "activate",
                  G_CALLBACK (on_angle_3_min_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_4_max_entry, "activate",
                  G_CALLBACK (on_angle_4_max_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_4_min_entry, "activate",
                  G_CALLBACK (on_angle_4_min_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_3_max_entry, "activate",
                  G_CALLBACK (on_angle_3_max_entry_activate),
                  NULL);
g_signal_connect ((gpointer) widgets_parametres->angle_6_max_entry, "activate",
                  G_CALLBACK (on_angle_6_max_entry_activate),
```

```
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->angle_6_min_entry, "activate",
                    G_CALLBACK (on_angle_6_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->angle_5_max_entry, "activate",
                    G_CALLBACK (on_angle_5_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->angle_5_min_entry, "activate",
                    G_CALLBACK (on_angle_5_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->config_rob_esq_entry, "activate",
                    G_CALLBACK (on_config_rob_esq_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->config_rob_dret_entry, "activate",
                    G_CALLBACK (on_config_rob_dret_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->rot_z_max_entry, "activate",
                    G_CALLBACK (on_rot_z_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->rot_z_min_entry, "activate",
                    G_CALLBACK (on_rot_z_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->rot_y_max_entry, "activate",
                    G_CALLBACK (on_rot_y_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->rot_y_min_entry, "activate",
                    G_CALLBACK (on_rot_y_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->rot_x_max_entry, "activate",
                    G_CALLBACK (on_rot_x_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->rot_x_min_entry, "activate",
                    G_CALLBACK (on_rot_x_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->z_max_entry, "activate",
                    G_CALLBACK (on_z_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->z_min_entry, "activate",
                    G_CALLBACK (on_z_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->y_max_entry, "activate",
                    G_CALLBACK (on_y_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->y_min_entry, "activate",
                    G_CALLBACK (on_y_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->x_max_entry, "activate",
                    G_CALLBACK (on_x_max_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->x_min_entry, "activate",
                    G_CALLBACK (on_x_min_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->coord_cart_min_prensio_entry, "activate",
                    G_CALLBACK (on_coord_cart_min_prensio_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) widgets_parametres->coord_cart_max_prensio_entry, "activate",
                    G_CALLBACK (on_coord_cart_max_prensio_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) carregar_par_button, "clicked",
                    G_CALLBACK (on_carregar_par_button_clicked),
                    (gpointer) widgets_parametres);


  /* Store pointers to all widgets, for use by lookup_widget(). */
  GLADE_HOOKUP_OBJECT_NO_REF (window_parametres, window_parametres, "window_parametres");
  GLADE_HOOKUP_OBJECT (window_parametres, hbox3, "hbox3");
  GLADE_HOOKUP_OBJECT (window_parametres, esquerra_layout, "esquerra_layout");
  GLADE_HOOKUP_OBJECT (window_parametres, fase_aprenentatge_label, "fase_aprenentatge_label");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator14, "hseparator14");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->borrar_graf_checkbutton,
                       "widgets_parametres->borrar_graf_checkbutton");
  GLADE_HOOKUP_OBJECT (window_parametres, max_nodes_label, "max_nodes_label");
  GLADE_HOOKUP_OBJECT (window_parametres, min_nodes_label, "min_nodes_label");
  GLADE_HOOKUP_OBJECT (window_parametres, num_veins_label, "num_veins_label");
  GLADE_HOOKUP_OBJECT (window_parametres, max_dist_label, "max_dist_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->nom_base_entry,
                       "widgets_parametres->nom_base_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->max_nodes_entry,
                       "widgets_parametres->max_nodes_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->min_nodes_entry,
                       "widgets_parametres->min_nodes_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->num_veins_entry,
                       "widgets_parametres->num_veins_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->max_dist_entry,
                       "widgets_parametres->max_dist_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, cami_local_label, "cami_local_label");
  GLADE_HOOKUP_OBJECT (window_parametres, pas_cami_local_label, "pas_cami_local_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->pas_cami_local_entry,
                       "widgets_parametres->pas_cami_local_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, interpol_label, "interpol_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rob_esq_interpol_radiobutton,
```

```
                        "widgets_parametres->rob_esq_interpol_radiobutton");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rob_dret_interpol_radiobutton,
                        "widgets_parametres->rob_dret_interpol_radiobutton");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator15, "hseparator15");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rob_esq_aleat_radiobutton,
                        "widgets_parametres->rob_esq_aleat_radiobutton");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rob_dret_aleat_radiobutton,
                        "widgets_parametres->rob_dret_aleat_radiobutton");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->objecte_aleat_radiobutton,
                        "widgets_parametres->objecte_aleat_radiobutton");
  GLADE_HOOKUP_OBJECT (window_parametres, gene_aleatori_label, "gene_aleatori_label");
  GLADE_HOOKUP_OBJECT (window_parametres, fase_cerca_label, "fase_cerca_label");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator16, "hseparator16");
  GLADE_HOOKUP_OBJECT (window_parametres, max_profund_label, "max_profund_label");
  GLADE_HOOKUP_OBJECT (window_parametres, sortida_label, "sortida_label");
  GLADE_HOOKUP_OBJECT (window_parametres, cerca_sortida_hseparator, "cerca_sortida_hseparator");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->fitxer_debug_checkbutton,
                        "widgets_parametres->fitxer_debug_checkbutton");
  GLADE_HOOKUP_OBJECT (window_parametres, nom_fitxer_debug_label, "nom_fitxer_debug_label");
  GLADE_HOOKUP_OBJECT (window_parametres, desar_par_button, "desar_par_button");
  GLADE_HOOKUP_OBJECT (window_parametres, nom_base_label, "nom_base_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->num_pasos_entry,
                        "widgets_parametres->num_pasos_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, num_pasos_label, "num_pasos_label");
  GLADE_HOOKUP_OBJECT (window_parametres, fora_dist_max_label, "fora_dist_max_label");
  GLADE_HOOKUP_OBJECT (window_parametres, central_layout, "central_layout");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator21, "hseparator21");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator22, "hseparator22");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator23, "hseparator23");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator24, "hseparator24");
  GLADE_HOOKUP_OBJECT (window_parametres, checkbutton1, "checkbutton1");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->max_profunditat_entry,
                        "widgets_parametres->max_profunditat_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->fora_dist_max_entry,
                        "widgets_parametres->fora_dist_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, max_no_cami_local_label, "max_no_cami_local_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->max_no_local_entry,
                        "widgets_parametres->max_no_local_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->nom_fitxer_debug_entry,
                        "widgets_parametres->nom_fitxer_debug_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, vseparator2, "vseparator2");
  GLADE_HOOKUP_OBJECT (window_parametres, dret_layout, "dret_layout");
  GLADE_HOOKUP_OBJECT (window_parametres, robot_label, "robot_label");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator18, "hseparator18");
  GLADE_HOOKUP_OBJECT (window_parametres, angle_1_label, "angle_1_label");
  GLADE_HOOKUP_OBJECT (window_parametres, angle_2_label, "angle_2_label");
  GLADE_HOOKUP_OBJECT (window_parametres, angle_3_abel, "angle_3_abel");
  GLADE_HOOKUP_OBJECT (window_parametres, angle_4_label, "angle_4_label");
  GLADE_HOOKUP_OBJECT (window_parametres, angle_5_label, "angle_5_label");
  GLADE_HOOKUP_OBJECT (window_parametres, angle_6_label, "angle_6_label");
  GLADE_HOOKUP_OBJECT (window_parametres, min_robot_label, "min_robot_label");
  GLADE_HOOKUP_OBJECT (window_parametres, max_robot_label, "max_robot_label");
  GLADE_HOOKUP_OBJECT (window_parametres, percen_expansio_label, "percen_expansio_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->percen_expansio_entry,
                        "widgets_parametres->percen_expansio_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_1_min_entry,
                        "widgets_parametres->angle_1_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_1_max_entry,
                        "widgets_parametres->angle_1_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_2_min_entry,
                        "widgets_parametres->angle_2_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_2_max_entry,
                        "widgets_parametres->angle_2_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_3_min_entry,
                        "widgets_parametres->angle_3_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_4_max_entry,
                        "widgets_parametres->angle_4_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_4_min_entry,
                        "widgets_parametres->angle_4_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_3_max_entry,
                        "widgets_parametres->angle_3_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_6_max_entry,
                        "widgets_parametres->angle_6_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_6_min_entry,
                        "widgets_parametres->angle_6_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_5_max_entry,
                        "widgets_parametres->angle_5_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->angle_5_min_entry,
                        "widgets_parametres->angle_5_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, config_rob_esq_label, "config_rob_esq_label");
  GLADE_HOOKUP_OBJECT (window_parametres, config_rob_dret_label, "config_rob_dret_label");
  GLADE_HOOKUP_OBJECT (window_parametres, config_label, "config_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->config_rob_esq_entry,
                        "widgets_parametres->config_rob_esq_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->config_rob_dret_entry,
                        "widgets_parametres->config_rob_dret_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, objecte_label, "objecte_label");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator19, "hseparator19");
  GLADE_HOOKUP_OBJECT (window_parametres, x_label, "x_label");
  GLADE_HOOKUP_OBJECT (window_parametres, y_label, "y_label");
  GLADE_HOOKUP_OBJECT (window_parametres, z_label, "z_label");
  GLADE_HOOKUP_OBJECT (window_parametres, rot_x_label, "rot_x_label");
```

```
  GLADE_HOOKUP_OBJECT (window_parametres, rot_y_label, "rot_y_label");
  GLADE_HOOKUP_OBJECT (window_parametres, rot_z_label, "rot_z_label");
  GLADE_HOOKUP_OBJECT (window_parametres, min_objecte_label, "min_objecte_label");
  GLADE_HOOKUP_OBJECT (window_parametres, max_objecte_label, "max_objecte_label");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rot_z_max_entry,
                       "widgets_parametres->rot_z_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rot_z_min_entry,
                       "widgets_parametres->rot_z_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rot_y_max_entry,
                       "widgets_parametres->rot_y_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rot_y_min_entry,
                       "widgets_parametres->rot_y_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rot_x_max_entry,
                       "widgets_parametres->rot_x_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->rot_x_min_entry,
                       "widgets_parametres->rot_x_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->z_max_entry,
                       "widgets_parametres->z_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->z_min_entry,
                       "widgets_parametres->z_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->y_max_entry,
                       "widgets_parametres->y_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->y_min_entry,
                       "widgets_parametres->y_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->x_max_entry,
                       "widgets_parametres->x_max_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->x_min_entry,
                       "widgets_parametres->x_min_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, prensio_label, "prensio_label");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator20, "hseparator20");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->coord_cart_min_prensio_entry,
                       "widgets_parametres->coord_cart_min_prensio_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, widgets_parametres->coord_cart_max_prensio_entry,
                       "widgets_parametres->coord_cart_max_prensio_entry");
  GLADE_HOOKUP_OBJECT (window_parametres, min_prensio_label, "min_prensio_label");
  GLADE_HOOKUP_OBJECT (window_parametres, max_prensio_label, "max_prensio_label");
  GLADE_HOOKUP_OBJECT (window_parametres, label45, "label45");
  GLADE_HOOKUP_OBJECT (window_parametres, hseparator25, "hseparator25");
  GLADE_HOOKUP_OBJECT (window_parametres, carregar_par_button, "carregar_par_button");

  return window_parametres;
}

GtkWidget*
create_tasques_window (void)
{
  GtkWidget *tasques_window;
  GtkWidget *fixed1;
  GtkWidget *fitxer_infi_entry;
  GtkWidget *fitxer_trj_entry;
  GtkWidget *fitxer_trj_button;
  GtkWidget *inicialitzar_button;
  GtkWidget *carregar_infi_button;
  GtkWidget *fitxer_infi_button;
  GtkWidget *suavitzar_button;
  GtkWidget *cercar_button;
  GtkWidget *carregar_trj_button;
  GtkWidget *desar_trj_button;
  GtkWidget *visualitzar_button;
  GtkWidget *memoritzar_config_button;
  GtkWidget *borrar_config_button;
  GtkWidget *executar_button;
  GtkWidget *construir_button;
  GtkWidget *extendre_button;
  GtkWidget *carregar_map_button;
  GtkWidget *desar_map_button;
  GtkWidget *marcar_ini_button;
  GtkWidget *marcar_fi_button;
  GtkWidget *desar_infi_button;
  GtkWidget *veure_ini_fi_button;

  tasques_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_widget_set_size_request (tasques_window, 792, 73);
  gtk_window_set_title (GTK_WINDOW (tasques_window), _("Tasques"));

  fixed1 = gtk_fixed_new ();
  gtk_widget_show (fixed1);
  gtk_container_add (GTK_CONTAINER (tasques_window), fixed1);

  fitxer_infi_entry = gtk_entry_new ();
  gtk_widget_show (fitxer_infi_entry);
  gtk_fixed_put (GTK_FIXED (fixed1), fitxer_infi_entry, 280, 48);
  gtk_widget_set_size_request (fitxer_infi_entry, 128, 25);

  fitxer_trj_entry = gtk_entry_new ();
  gtk_widget_show (fitxer_trj_entry);
  gtk_fixed_put (GTK_FIXED (fixed1), fitxer_trj_entry, 472, 48);
  gtk_widget_set_size_request (fitxer_trj_entry, 128, 25);

  fitxer_trj_button = gtk_button_new_with_mnemonic (_("fitxer_trj"));
  gtk_widget_show (fitxer_trj_button);
  gtk_fixed_put (GTK_FIXED (fixed1), fitxer_trj_button, 416, 48);
```

```
gtk_widget_set_size_request (fitxer_trj_button, 56, 25);

inicialitzar_button = gtk_button_new_with_mnemonic (_("inicialitzar"));
gtk_widget_show (inicialitzar_button);
gtk_fixed_put (GTK_FIXED (fixed1), inicialitzar_button, 0, 0);
gtk_widget_set_size_request (inicialitzar_button, 64, 73);

carregar_infi_button = gtk_button_new_with_mnemonic (_("carregar *.infi"));
gtk_widget_show (carregar_infi_button);
gtk_fixed_put (GTK_FIXED (fixed1), carregar_infi_button, 216, 24);
gtk_widget_set_size_request (carregar_infi_button, 96, 25);

fitxer_infi_button = gtk_button_new_with_mnemonic (_("fitxer infi"));
gtk_widget_show (fitxer_infi_button);
gtk_fixed_put (GTK_FIXED (fixed1), fitxer_infi_button, 216, 48);
gtk_widget_set_size_request (fitxer_infi_button, 64, 25);

suavitzar_button = gtk_button_new_with_mnemonic (_("suavitzar"));
gtk_widget_show (suavitzar_button);
gtk_fixed_put (GTK_FIXED (fixed1), suavitzar_button, 504, 0);
gtk_widget_set_size_request (suavitzar_button, 96, 25);

cercar_button = gtk_button_new_with_mnemonic (_("cercar"));
gtk_widget_show (cercar_button);
gtk_fixed_put (GTK_FIXED (fixed1), cercar_button, 416, 0);
gtk_widget_set_size_request (cercar_button, 88, 25);

carregar_trj_button = gtk_button_new_with_mnemonic (_("carregar *.trj"));
gtk_widget_show (carregar_trj_button);
gtk_fixed_put (GTK_FIXED (fixed1), carregar_trj_button, 416, 24);
gtk_widget_set_size_request (carregar_trj_button, 88, 25);

desar_trj_button = gtk_button_new_with_mnemonic (_("desar *.trj"));
gtk_widget_show (desar_trj_button);
gtk_fixed_put (GTK_FIXED (fixed1), desar_trj_button, 504, 24);
gtk_widget_set_size_request (desar_trj_button, 96, 25);

visualitzar_button = gtk_button_new_with_mnemonic (_("visualitzar"));
gtk_widget_show (visualitzar_button);
gtk_fixed_put (GTK_FIXED (fixed1), visualitzar_button, 608, 0);
gtk_widget_set_size_request (visualitzar_button, 112, 25);

memoritzar_config_button = gtk_button_new_with_mnemonic (_("memoritzar config."));
gtk_widget_show (memoritzar_config_button);
gtk_fixed_put (GTK_FIXED (fixed1), memoritzar_config_button, 608, 24);
gtk_widget_set_size_request (memoritzar_config_button, 112, 25);

borrar_config_button = gtk_button_new_with_mnemonic (_("borrar config."));
gtk_widget_show (borrar_config_button);
gtk_fixed_put (GTK_FIXED (fixed1), borrar_config_button, 608, 48);
gtk_widget_set_size_request (borrar_config_button, 112, 25);

executar_button = gtk_button_new_with_mnemonic (_("executar"));
gtk_widget_show (executar_button);
gtk_fixed_put (GTK_FIXED (fixed1), executar_button, 728, 0);
gtk_widget_set_size_request (executar_button, 64, 73);

construir_button = gtk_button_new_with_mnemonic (_("construir"));
gtk_widget_show (construir_button);
gtk_fixed_put (GTK_FIXED (fixed1), construir_button, 72, 0);
gtk_widget_set_size_request (construir_button, 72, 25);

extendre_button = gtk_button_new_with_mnemonic (_("extendre"));
gtk_widget_show (extendre_button);
gtk_fixed_put (GTK_FIXED (fixed1), extendre_button, 144, 0);
gtk_widget_set_size_request (extendre_button, 64, 25);

carregar_map_button = gtk_button_new_with_mnemonic (_("carregar *.map"));
gtk_widget_show (carregar_map_button);
gtk_fixed_put (GTK_FIXED (fixed1), carregar_map_button, 72, 24);
gtk_widget_set_size_request (carregar_map_button, 136, 25);

desar_map_button = gtk_button_new_with_mnemonic (_("desar *.map"));
gtk_widget_show (desar_map_button);
gtk_fixed_put (GTK_FIXED (fixed1), desar_map_button, 72, 48);
gtk_widget_set_size_request (desar_map_button, 136, 25);

marcar_ini_button = gtk_button_new_with_mnemonic (_("marcar ini"));
gtk_widget_show (marcar_ini_button);
gtk_fixed_put (GTK_FIXED (fixed1), marcar_ini_button, 216, 0);
gtk_widget_set_size_request (marcar_ini_button, 64, 25);

marcar_fi_button = gtk_button_new_with_mnemonic (_("marcar fi"));
gtk_widget_show (marcar_fi_button);
gtk_fixed_put (GTK_FIXED (fixed1), marcar_fi_button, 280, 0);
gtk_widget_set_size_request (marcar_fi_button, 56, 25);

desar_infi_button = gtk_button_new_with_mnemonic (_("desar *.infi"));
gtk_widget_show (desar_infi_button);
gtk_fixed_put (GTK_FIXED (fixed1), desar_infi_button, 312, 24);
gtk_widget_set_size_request (desar_infi_button, 96, 25);
```

```
  veure_ini_fi_button = gtk_button_new_with_mnemonic (_("veure_ini/fi"));
  gtk_widget_show (veure_ini_fi_button);
  gtk_fixed_put (GTK_FIXED (fixed1), veure_ini_fi_button, 336, 0);
  gtk_widget_set_size_request (veure_ini_fi_button, 72, 25);

  g_signal_connect ((gpointer) fitxer_infi_entry, "activate",
                    G_CALLBACK (on_fitxer_infi_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) fitxer_trj_entry, "activate",
                    G_CALLBACK (on_fitxer_trj_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) inicialitzar_button, "clicked",
                    G_CALLBACK (on_inicialitzar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) carregar_infi_button, "clicked",
                    G_CALLBACK (on_carregar_infi_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) suavitzar_button, "clicked",
                    G_CALLBACK (on_suavitzar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) cercar_button, "clicked",
                    G_CALLBACK (on_cercar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) carregar_trj_button, "clicked",
                    G_CALLBACK (on_carregar_trj_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) desar_trj_button, "clicked",
                    G_CALLBACK (on_desar_trj_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) visualitzar_button, "clicked",
                    G_CALLBACK (on_visualitzar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) memoritzar_config_button, "clicked",
                    G_CALLBACK (on_memoritzar_config_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) borrar_config_button, "clicked",
                    G_CALLBACK (on_borrar_config_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) executar_button, "clicked",
                    G_CALLBACK (on_executar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) construir_button, "clicked",
                    G_CALLBACK (on_construir_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) extendre_button, "clicked",
                    G_CALLBACK (on_extendre_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) carregar_map_button, "clicked",
                    G_CALLBACK (on_carregar_map_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) desar_map_button, "clicked",
                    G_CALLBACK (on_desar_map_button_clicked),
                    NULL);

  g_signal_connect ((gpointer) marcar_ini_button, "clicked",
                    G_CALLBACK (on_marcar_ini_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) marcar_fi_button, "clicked",
                    G_CALLBACK (on_marcar_fi_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) desar_infi_button, "clicked",
                    G_CALLBACK (on_desar_infi_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) veure_ini_fi_button, "clicked",
                    G_CALLBACK (on_veure_ini_fi_button_clicked),
                    NULL);

  //Quan es desi el mapa tambe vull que es desi el fitxer de parametres


  g_signal_connect ((gpointer) desar_map_button, "clicked",
                    G_CALLBACK (on_desar_par_button_clicked),
                    NULL);

 //Quan es carregui el mapa tambe vull que es carregui el fitxer de parametres

  g_signal_connect ((gpointer) carregar_map_button, "clicked",
                    G_CALLBACK (on_carregar_par_button_clicked),
                    (gpointer) widgets_parametres);


  /* Store pointers to all widgets, for use by lookup_widget(). */
  GLADE_HOOKUP_OBJECT_NO_REF (tasques_window, tasques_window, "tasques_window");
  GLADE_HOOKUP_OBJECT (tasques_window, fixed1, "fixed1");
  GLADE_HOOKUP_OBJECT (tasques_window, fitxer_infi_entry, "fitxer_infi_entry");
  GLADE_HOOKUP_OBJECT (tasques_window, fitxer_trj_entry, "fitxer_trj_entry");
  GLADE_HOOKUP_OBJECT (tasques_window, fitxer_trj_button, "fitxer_trj_button");
  GLADE_HOOKUP_OBJECT (tasques_window, inicialitzar_button, "inicialitzar_button");
  GLADE_HOOKUP_OBJECT (tasques_window, carregar_infi_button, "carregar_infi_button");
  GLADE_HOOKUP_OBJECT (tasques_window, fitxer_infi_button, "fitxer_infi_button");
  GLADE_HOOKUP_OBJECT (tasques_window, suavitzar_button, "suavitzar_button");
  GLADE_HOOKUP_OBJECT (tasques_window, cercar_button, "cercar_button");
```

```
  GLADE_HOOKUP_OBJECT (tasques_window, carregar_trj_button, "carregar_trj_button");
  GLADE_HOOKUP_OBJECT (tasques_window, desar_trj_button, "desar_trj_button");
  GLADE_HOOKUP_OBJECT (tasques_window, visualitzar_button, "visualitzar_button");
  GLADE_HOOKUP_OBJECT (tasques_window, memoritzar_config_button, "memoritzar_config_button");
  GLADE_HOOKUP_OBJECT (tasques_window, borrar_config_button, "borrar_config_button");
  GLADE_HOOKUP_OBJECT (tasques_window, executar_button, "executar_button");
  GLADE_HOOKUP_OBJECT (tasques_window, construir_button, "construir_button");
  GLADE_HOOKUP_OBJECT (tasques_window, extendre_button, "extendre_button");
  GLADE_HOOKUP_OBJECT (tasques_window, carregar_map_button, "carregar_map_button");
  GLADE_HOOKUP_OBJECT (tasques_window, desar_map_button, "desar_map_button");
  GLADE_HOOKUP_OBJECT (tasques_window, marcar_ini_button, "marcar_ini_button");
  GLADE_HOOKUP_OBJECT (tasques_window, marcar_fi_button, "marcar_fi_button");
  GLADE_HOOKUP_OBJECT (tasques_window, desar_infi_button, "desar_infi_button");
  GLADE_HOOKUP_OBJECT (tasques_window, veure_ini_fi_button, "veure_ini_fi_button");

  return tasques_window;
}

GtkWidget*
create_analisi_graf_window (void)
{
  GtkWidget *analisi_graf_window;
  GtkWidget *fixed1;
  GtkWidget *vn_entry;
  GtkWidget *vertex_i_entry;
  GtkWidget *vertex_j_entry;
  GtkWidget *vn_label;
  GtkWidget *n_components_entry;
  GtkWidget *n_label;
  GtkWidget *max_label;
  GtkWidget *min_analisi_entry;
  GtkWidget *max_analisi_entry;
  GtkWidget *min_label;
  GtkWidget *filtrar_button;
  GtkWidget *n_components_button;
  GtkWidget *distancia_analisi_button;
  GtkWidget *visualitzar_analisi_button;
  GtkWidget *cercar_analisi_button;
  GtkWidget *vertex_ant_button;
  GtkWidget *alignment1;
  GtkWidget *hbox1;
  GtkWidget *image1;
  GtkWidget *vi_label;
  GtkWidget *vj_label;
  GtkWidget *vertex_seg_button;
  GtkWidget *alignment2;
  GtkWidget *hbox2;
  GtkWidget *image2;
  GtkWidget *hseparator2;
  GtkWidget *connectivitat_label;
  GtkWidget *visualitzacio_hseparator;
  GtkWidget *visualitzacio_label;

  analisi_graf_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (analisi_graf_window), _("Analisi_"));

  fixed1 = gtk_fixed_new ();
  gtk_widget_show (fixed1);
  gtk_container_add (GTK_CONTAINER (analisi_graf_window), fixed1);

  vn_entry = gtk_entry_new ();
  gtk_widget_show (vn_entry);
  gtk_fixed_put (GTK_FIXED (fixed1), vn_entry, 96, 40);
  gtk_widget_set_size_request (vn_entry, 48, 25);

  vertex_i_entry = gtk_entry_new ();
  gtk_widget_show (vertex_i_entry);
  gtk_fixed_put (GTK_FIXED (fixed1), vertex_i_entry, 48, 88);
  gtk_widget_set_size_request (vertex_i_entry, 48, 25);

  vertex_j_entry = gtk_entry_new ();
  gtk_widget_show (vertex_j_entry);
  gtk_fixed_put (GTK_FIXED (fixed1), vertex_j_entry, 96, 88);
  gtk_widget_set_size_request (vertex_j_entry, 48, 25);

  vn_label = gtk_label_new (_("vn"));
  gtk_widget_show (vn_label);
  gtk_fixed_put (GTK_FIXED (fixed1), vn_label, 96, 24);
  gtk_widget_set_size_request (vn_label, 48, 16);
  gtk_label_set_justify (GTK_LABEL (vn_label), GTK_JUSTIFY_LEFT);

  n_components_entry = gtk_entry_new ();
  gtk_widget_show (n_components_entry);
  gtk_fixed_put (GTK_FIXED (fixed1), n_components_entry, 96, 176);
  gtk_widget_set_size_request (n_components_entry, 48, 25);

  n_label = gtk_label_new (_("n"));
  gtk_widget_show (n_label);
  gtk_fixed_put (GTK_FIXED (fixed1), n_label, 104, 160);
  gtk_widget_set_size_request (n_label, 34, 16);
  gtk_label_set_justify (GTK_LABEL (n_label), GTK_JUSTIFY_LEFT);
```

```
max_label = gtk_label_new (_("max."));
gtk_widget_show (max_label);
gtk_fixed_put (GTK_FIXED (fixed1), max_label, 104, 200);
gtk_widget_set_size_request (max_label, 32, 16);
gtk_label_set_justify (GTK_LABEL (max_label), GTK_JUSTIFY_LEFT);

min_analisi_entry = gtk_entry_new ();
gtk_widget_show (min_analisi_entry);
gtk_fixed_put (GTK_FIXED (fixed1), min_analisi_entry, 48, 216);
gtk_widget_set_size_request (min_analisi_entry, 48, 25);

max_analisi_entry = gtk_entry_new ();
gtk_widget_show (max_analisi_entry);
gtk_fixed_put (GTK_FIXED (fixed1), max_analisi_entry, 96, 216);
gtk_widget_set_size_request (max_analisi_entry, 48, 25);

min_label = gtk_label_new (_("min."));
gtk_widget_show (min_label);
gtk_fixed_put (GTK_FIXED (fixed1), min_label, 56, 200);
gtk_widget_set_size_request (min_label, 34, 16);
gtk_label_set_justify (GTK_LABEL (min_label), GTK_JUSTIFY_LEFT);

filtrar_button = gtk_button_new_with_mnemonic (_("filtrar"));
gtk_widget_show (filtrar_button);
gtk_fixed_put (GTK_FIXED (fixed1), filtrar_button, 0, 216);
gtk_widget_set_size_request (filtrar_button, 48, 25);

n_components_button = gtk_button_new_with_mnemonic (_("n_components"));
gtk_widget_show (n_components_button);
gtk_fixed_put (GTK_FIXED (fixed1), n_components_button, 0, 176);
gtk_widget_set_size_request (n_components_button, 96, 25);

distancia_analisi_button = gtk_button_new_with_mnemonic (_("distancia_"));
gtk_widget_show (distancia_analisi_button);
gtk_fixed_put (GTK_FIXED (fixed1), distancia_analisi_button, 72, 112);
gtk_widget_set_size_request (distancia_analisi_button, 72, 25);

visualitzar_analisi_button = gtk_button_new_with_mnemonic (_("visualitzar_"));
gtk_widget_show (visualitzar_analisi_button);
gtk_fixed_put (GTK_FIXED (fixed1), visualitzar_analisi_button, 0, 112);
gtk_widget_set_size_request (visualitzar_analisi_button, 72, 25);

cercar_analisi_button = gtk_button_new_with_mnemonic (_("cercar_"));
gtk_widget_show (cercar_analisi_button);
gtk_fixed_put (GTK_FIXED (fixed1), cercar_analisi_button, 0, 88);
gtk_widget_set_size_request (cercar_analisi_button, 48, 25);

vertex_ant_button = gtk_button_new ();
gtk_widget_show (vertex_ant_button);
gtk_fixed_put (GTK_FIXED (fixed1), vertex_ant_button, 0, 40);
gtk_widget_set_size_request (vertex_ant_button, 48, 25);

alignment1 = gtk_alignment_new (0.5, 0.5, 0, 0);
gtk_widget_show (alignment1);
gtk_container_add (GTK_CONTAINER (vertex_ant_button), alignment1);

hbox1 = gtk_hbox_new (FALSE, 2);
gtk_widget_show (hbox1);
gtk_container_add (GTK_CONTAINER (alignment1), hbox1);

image1 = gtk_image_new_from_stock ("gtk-go-back", GTK_ICON_SIZE_BUTTON);
gtk_widget_show (image1);
gtk_box_pack_start (GTK_BOX (hbox1), image1, FALSE, FALSE, 0);

vi_label = gtk_label_new (_("vi"));
gtk_widget_show (vi_label);
gtk_fixed_put (GTK_FIXED (fixed1), vi_label, 40, 72);
gtk_widget_set_size_request (vi_label, 56, 16);
gtk_label_set_justify (GTK_LABEL (vi_label), GTK_JUSTIFY_LEFT);

vj_label = gtk_label_new (_("vj"));
gtk_widget_show (vj_label);
gtk_fixed_put (GTK_FIXED (fixed1), vj_label, 96, 72);
gtk_widget_set_size_request (vj_label, 48, 16);
gtk_label_set_justify (GTK_LABEL (vj_label), GTK_JUSTIFY_LEFT);

vertex_seg_button = gtk_button_new ();
gtk_widget_show (vertex_seg_button);
gtk_fixed_put (GTK_FIXED (fixed1), vertex_seg_button, 48, 40);
gtk_widget_set_size_request (vertex_seg_button, 48, 25);

alignment2 = gtk_alignment_new (0.5, 0.5, 0, 0);
gtk_widget_show (alignment2);
gtk_container_add (GTK_CONTAINER (vertex_seg_button), alignment2);

hbox2 = gtk_hbox_new (FALSE, 2);
gtk_widget_show (hbox2);
gtk_container_add (GTK_CONTAINER (alignment2), hbox2);

image2 = gtk_image_new_from_stock ("gtk-go-forward", GTK_ICON_SIZE_BUTTON);
gtk_widget_show (image2);
gtk_box_pack_start (GTK_BOX (hbox2), image2, FALSE, FALSE, 0);
```

```
  hseparator2 = gtk_hseparator_new ();
  gtk_widget_show (hseparator2);
  gtk_fixed_put (GTK_FIXED (fixed1), hseparator2, 0, 152);
  gtk_widget_set_size_request (hseparator2, 144, 16);

  connectivitat_label = gtk_label_new (_("Connectivitat"));
  gtk_widget_show (connectivitat_label);
  gtk_fixed_put (GTK_FIXED (fixed1), connectivitat_label, 0, 144);
  gtk_widget_set_size_request (connectivitat_label, 96, 16);
  gtk_label_set_justify (GTK_LABEL (connectivitat_label), GTK_JUSTIFY_LEFT);

  visualitzacio_hseparator = gtk_hseparator_new ();
  gtk_widget_show (visualitzacio_hseparator);
  gtk_fixed_put (GTK_FIXED (fixed1), visualitzacio_hseparator, 0, 16);
  gtk_widget_set_size_request (visualitzacio_hseparator, 144, 16);

  visualitzacio_label = gtk_label_new (_("Visualitzacio"));
  gtk_widget_show (visualitzacio_label);
  gtk_fixed_put (GTK_FIXED (fixed1), visualitzacio_label, 0, 8);
  gtk_widget_set_size_request (visualitzacio_label, 96, 16);
  gtk_label_set_justify (GTK_LABEL (visualitzacio_label), GTK_JUSTIFY_LEFT);

  g_signal_connect ((gpointer) vn_entry, "activate",
                    G_CALLBACK (on_vn_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) vertex_i_entry, "activate",
                    G_CALLBACK (on_vertex_i_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) vertex_j_entry, "activate",
                    G_CALLBACK (on_vertex_j_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) n_components_entry, "activate",
                    G_CALLBACK (on_n_components_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) min_analisi_entry, "activate",
                    G_CALLBACK (on_min_analisi_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) max_analisi_entry, "activate",
                    G_CALLBACK (on_max_analisi_entry_activate),
                    NULL);
  g_signal_connect ((gpointer) filtrar_button, "clicked",
                    G_CALLBACK (on_filtrar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) n_components_button, "clicked",
                    G_CALLBACK (on_n_components_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) distancia_analisi_button, "clicked",
                    G_CALLBACK (on_distancia_analisi_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) visualitzar_analisi_button, "clicked",
                    G_CALLBACK (on_visualitzar_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) cercar_analisi_button, "clicked",
                    G_CALLBACK (on_cercar_analisi_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) vertex_ant_button, "clicked",
                    G_CALLBACK (on_vertex_ant_button_clicked),
                    NULL);
  g_signal_connect ((gpointer) vertex_seg_button, "clicked",
                    G_CALLBACK (on_vertex_seg_button_clicked),
                    NULL);

  //controladors creats per mi per actualitzar el numero de vertexs, en recorrer
  //la visualitzacio del graf
  gtk_signal_connect((gpointer)(vertex_ant_button),"clicked",
                     G_CALLBACK(set_value_entry_by_button),(gpointer)(vn_entry));
  gtk_signal_connect((gpointer)(vertex_seg_button),"clicked",
                     G_CALLBACK(set_value_entry_by_button),(gpointer)(vn_entry));

 /* Store pointers to all widgets, for use by lookup_widget(). */
  GLADE_HOOKUP_OBJECT_NO_REF (analisi_graf_window, analisi_graf_window, "analisi_graf_window");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, fixed1, "fixed1");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vn_entry, "vn_entry");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vertex_i_entry, "vertex_i_entry");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vertex_j_entry, "vertex_j_entry");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vn_label, "vn_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, n_components_entry, "n_components_entry");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, n_label, "n_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, max_label, "max_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, min_analisi_entry, "min_analisi_entry");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, max_analisi_entry, "max_analisi_entry");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, min_label, "min_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, filtrar_button, "filtrar_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, n_components_button, "n_components_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, distancia_analisi_button, "distancia_analisi_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, visualitzar_analisi_button, "visualitzar_analisi_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, cercar_analisi_button, "cercar_analisi_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vertex_ant_button, "vertex_ant_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, alignment1, "alignment1");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, hbox1, "hbox1");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, image1, "image1");
```

```
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vi_label, "vi_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vj_label, "vj_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, vertex_seg_button, "vertex_seg_button");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, alignment2, "alignment2");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, hbox2, "hbox2");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, image2, "image2");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, hseparator2, "hseparator2");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, connectivitat_label, "connectivitat_label");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, visualitzacio_hseparator, "visualitzacio_hseparator");
  GLADE_HOOKUP_OBJECT (analisi_graf_window, visualitzacio_label, "visualitzacio_label");


  return analisi_graf_window;
}
```

H.6 config.h

```
/* config.h.  Generated by configure.  */
/* config.h.in.  Generated from configure.in by autoheader.  */
#define ENABLE_NLS 1
/* #undef HAVE_CATGETS */
#define HAVE_GETTEXT 1
#define GETTEXT_PACKAGE "robots"
#define HAVE_LC_MESSAGES 1
/* #undef HAVE_STPCPY */
/* #undef HAVE_LIBSM */

/* always defined to indicate that i18n is enabled */
#define ENABLE_NLS 1

/* Define to 1 if you have the `bind_textdomain_codeset' function. */
#define HAVE_BIND_TEXTDOMAIN_CODESET 1

/* Define to 1 if you have the `dcgettext' function. */
#define HAVE_DCGETTEXT 1

/* Define if the GNU gettext() function is already present or preinstalled. */
#define HAVE_GETTEXT 1

/* Define to 1 if you have the <inttypes.h> header file. */
#define HAVE_INTTYPES_H 1

/* Define if your <locale.h> file defines LC_MESSAGES. */
#define HAVE_LC_MESSAGES 1

/* Define to 1 if you have the <locale.h> header file. */
#define HAVE_LOCALE_H 1

/* Define to 1 if you have the <memory.h> header file. */
#define HAVE_MEMORY_H 1

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1

/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the <sys/stat.h> header file. */
#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/types.h> header file. */
#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have the <unistd.h> header file. */
#define HAVE_UNISTD_H 1

/* Name of package */
#define PACKAGE "robots"

/* Define to the address where bug reports for this package should be sent. */
#define PACKAGE_BUGREPORT ""

/* Define to the full name of this package. */
#define PACKAGE_NAME ""

/* Define to the full name and version of this package. */
#define PACKAGE_STRING ""

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME ""

/* Define to the version of this package. */
#define PACKAGE_VERSION ""
```

```
/* Define to 1 if you have the ANSI C header files. */
#define STDC_HEADERS 1

/* Version number of package */
#define VERSION "0.1"
```

H.7 support.h

```
/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <gtk/gtk.h>

/*
 * Standard gettext macros.
 */
#ifdef ENABLE_NLS
#  include <libintl.h>
#  undef _
#  define _(String) dgettext (PACKAGE, String)
#  ifdef gettext_noop
#    define N_(String) gettext_noop (String)
#  else
#    define N_(String) (String)
#  endif
#else
#  define textdomain(String) (String)
#  define gettext(String) (String)
#  define dgettext(Domain,Message) (Message)
#  define dcgettext(Domain,Message,Type) (Message)
#  define bindtextdomain(Domain,Directory) (Domain)
#  define _(String) (String)
#  define N_(String) (String)
#endif


/*
 * Public Functions.
 */

/*
 * This function returns a widget in a component created by Glade.
 * Call it with the toplevel widget in the component (i.e. a window/dialog),
 * or alternatively any widget in the component, and the name of the widget
 * you want returned.
 */
GtkWidget*  lookup_widget              (GtkWidget       *widget,
                                        const gchar     *widget_name);


/* Use this function to set the directory containing installed pixmaps. */
void        add_pixmap_directory       (const gchar     *directory);


/*
 * Private Functions.
 */

/* This is used to create the pixmaps used in the interface. */
GtkWidget*  create_pixmap              (GtkWidget       *widget,
                                        const gchar     *filename);

/* This is used to create the pixbufs used in the interface. */
GdkPixbuf*  create_pixbuf              (const gchar     *filename);

/* This is used to set ATK action descriptions. */
void        glade_set_atk_action_description (AtkAction       *action,
                                              const gchar     *action_name,
                                              const gchar     *description);
```

H.8 support.c

```
/*
 * DO NOT EDIT THIS FILE - it is generated by Glade.
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
```

```
#include <stdio.h>

#include <gtk/gtk.h>

#include "support.h"

GtkWidget*
lookup_widget                           (GtkWidget       *widget,
                                         const gchar     *widget_name)
{
  GtkWidget *parent, *found_widget;

  for (;;)
    {
      if (GTK_IS_MENU (widget))
        parent = gtk_menu_get_attach_widget (GTK_MENU (widget));
      else
        parent = widget->parent;
      if (!parent)
        parent = g_object_get_data (G_OBJECT (widget), "GladeParentKey");
      if (parent == NULL)
        break;
      widget = parent;
    }

  found_widget = (GtkWidget*) g_object_get_data (G_OBJECT (widget),
                                                 widget_name);
  if (!found_widget)
    g_warning ("Widget not found: %s", widget_name);
  return found_widget;
}

static GList *pixmaps_directories = NULL;

/* Use this function to set the directory containing installed pixmaps. */
void
add_pixmap_directory                    (const gchar     *directory)
{
  pixmaps_directories = g_list_prepend (pixmaps_directories,
                                        g_strdup (directory));
}

/* This is an internally used function to find pixmap files. */
static gchar*
find_pixmap_file                        (const gchar     *filename)
{
  GList *elem;

  /* We step through each of the pixmaps directory to find it. */
  elem = pixmaps_directories;
  while (elem)
    {
      gchar *pathname = g_strdup_printf ("%s%s%s", (gchar*)elem->data,
                                         G_DIR_SEPARATOR_S, filename);
      if (g_file_test (pathname, G_FILE_TEST_EXISTS))
        return pathname;
      g_free (pathname);
      elem = elem->next;
    }
  return NULL;
}

/* This is an internally used function to create pixmaps. */
GtkWidget*
create_pixmap                           (GtkWidget       *widget,
                                         const gchar     *filename)
{
  gchar *pathname = NULL;
  GtkWidget *pixmap;

  if (!filename || !filename[0])
      return gtk_image_new ();

  pathname = find_pixmap_file (filename);

  if (!pathname)
    {
      g_warning (_("Couldn't find pixmap file: %s"), filename);
      return gtk_image_new ();
    }

  pixmap = gtk_image_new_from_file (pathname);
  g_free (pathname);
  return pixmap;
}

/* This is an internally used function to create pixmaps. */
GdkPixbuf*
create_pixbuf                           (const gchar     *filename)
{
  gchar *pathname = NULL;
  GdkPixbuf *pixbuf;
```

```
  GError *error = NULL;

  if (!filename || !filename[0])
      return NULL;

  pathname = find_pixmap_file (filename);

  if (!pathname)
    {
      g_warning (_("Couldn't find pixmap file: %s"), filename);
      return NULL;
    }

  pixbuf = gdk_pixbuf_new_from_file (pathname, &error);
  if (!pixbuf)
    {
      fprintf (stderr, "Failed to load pixbuf file: %s: %s\n",
               pathname, error->message);
      g_error_free (error);
    }
  g_free (pathname);
  return pixbuf;
}

/* This is used to set ATK action descriptions. */
void
glade_set_atk_action_description       (AtkAction       *action,
                                        const gchar     *action_name,
                                        const gchar     *description)
{
  gint n_actions, i;

  n_actions = atk_action_get_n_actions (action);
  for (i = 0; i < n_actions; i++)
    {
      if (!strcmp (atk_action_get_name (action, i), action_name))
        atk_action_set_description (action, i, description);
    }
}
```

H.9 callbacks.h

```
////////////////////////////////////////////////////
/// @file callbacks.h
/// @brief Rutines associades als camps de les finestres.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par Descripicio
///      Aquestes rutines es criden quan es produeix un
///      event en alguna de les finestres de la interficie.
///      Cada un dels events va lligat amb un camp de les
///      finestres (boto, entrada,etc) i amb una rutina
///      d'aquest òmdul.


#include <gtk/gtk.h>

#include "macros_generals.h"
#include "globals.h"
#include "webots_interface.h"
#include "objecte.h"
#include "graf_config.h"
#include "fitxer.h"
#include "colisio.h"

//Includes per conectarme amb la Joana
#include "stdio.h"
#include <netinet/in.h>
#include <sys/socket.h>


//Rutines callbacks associades a la finestra de graus de llibertat.
void
on_g_llibertat_robot_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                        gpointer         user_data);
void
on_g_llibertat_objecte_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                        gpointer         user_data);

void
on_g_llibertat_prensio_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                        gpointer         user_data);
void
on_g_llibertat_pinca_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                         gpointer         user_data);
```

```
void
on_prensio_on_line_CheckButton_toggled (GtkToggleButton *togglebutton,
                                        gpointer         user_data);

void
set_value_entry_by_hscale(GtkRange  *range,
                          gpointer user_data);

void
set_value_hscale_by_entry(GtkEntry  *entry,
                          gpointer user_data);

void
set_value_hscale_to_0(GtkButton *button,
                      gpointer user_data);
void
on_g_llibertat_1_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data);

void
on_g_llibertat_2_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data);

void
on_g_llibertat_3_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data);

void
on_g_llibertat_4_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data);

void
on_g_llibertat_5_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data);

void
on_g_llibertat_6_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data);
void
on_obertura_pinca_hscale_value_changed (GtkRange        *range,
                                        gpointer         user_data);


void
on_esq_radiobutton_toggled    (GtkToggleButton *togglebutton,
                                        gpointer         user_data);

void
on_dret_radiobutton_toggled   (GtkToggleButton *togglebutton,
                                        gpointer         user_data);
void
on_g_sis_hscale_refresc_by_left_rob_scene(GtkToggleButton *togglebutton,
                                          gpointer         user_data);
void
on_g_sis_hscale_refresc_by_object_scene(GtkToggleButton *togglebutton,
                                        gpointer         user_data);
void
on_convertir_labels_a_g_llibertat_robot_radiobutton_toggled(GtkToggleButton *togglebutton,
                                                            gpointer         user_data);
void
on_convertir_labels_a_g_llibertat_solid_radiobutton_toggled(GtkToggleButton *togglebutton,
                                                            gpointer         user_data);

//Rutines callbacks associades a la finestra de àparmetres.
void
on_borrar_graf_checkbutton_toggled    (GtkToggleButton *togglebutton,
                                        gpointer         user_data);
void
on_nom_base_entry_activate          (GtkEntry        *entry,
                                     gpointer         user_data);

void
on_max_nodes_entry_activate             (GtkEntry        *entry,
                                         gpointer         user_data);
void
on_percen_expansio_entry_activate       (GtkEntry        *entry,
                                         gpointer         user_data);
     void
on_min_nodes_entry_activate             (GtkEntry        *entry,
                                         gpointer         user_data);
void
on_num_veins_entry_activate             (GtkEntry        *entry,
                                         gpointer         user_data);

void
on_max_dist_entry_activate              (GtkEntry        *entry,
                                         gpointer         user_data);

void
on_rob_esq_aleat_radiobutton_toggled   (GtkToggleButton *togglebutton,
                                        gpointer         user_data);
```

```
void
on_rob_dret_aleat_radiobutton_toggled  (GtkToggleButton *togglebutton,
                                        gpointer         user_data);

void
on_objecte_aleat_radiobutton_toggled   (GtkToggleButton *togglebutton,
                                        gpointer         user_data);

void
on_pas_cami_local_entry_activate       (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rob_esq_interpol_radiobutton_toggled  (GtkToggleButton *togglebutton,
                                          gpointer         user_data);

void
on_rob_dret_interpol_radiobutton_toggled (GtkToggleButton *togglebutton,
                                          gpointer         user_data);

void
on_num_pasos_entry_activate          (GtkEntry        *entry,
                                      gpointer         user_data);

void
on_fora_dist_max_entry_activate        (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_max_no_local_entry_activate       (GtkEntry        *entry,
                                      gpointer         user_data);

void
on_max_profunditat_entry_activate      (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_nom_fitxer_debug_entry_activate           (GtkEntry        *entry,
                                        gpointer         user_data);


void
on_desar_par_button_clicked          (GtkButton       *button,
                                      gpointer         user_data);


void
on_angle_1_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_1_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_2_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_2_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_3_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_3_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_4_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_4_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_5_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_5_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_6_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_angle_6_max_entry_activate          (GtkEntry        *entry,
```

```
                                        gpointer         user_data);

void
on_config_rob_esq_entry_activate        (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_config_rob_dret_entry_activate       (GtkEntry        *entry,
                                        gpointer         user_data);
void
on_x_max_entry_activate                 (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_x_min_entry_activate                 (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_y_max_entry_activate                 (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_y_min_entry_activate                 (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_z_max_entry_activate                 (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_z_min_entry_activate                 (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rot_x_max_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rot_x_min_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rot_y_max_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rot_y_min_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rot_z_max_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_rot_z_min_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_coord_cart_min_prensio_entry_activate
                                        (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_coord_cart_max_prensio_entry_activate
                                        (GtkEntry        *entry,
                                        gpointer         user_data);
void
on_carregar_par_button_clicked         (GtkButton       *button,
                                        gpointer         user_data);

//Rutines callback associades a la tercera finestra, a la de tasques.
void
on_inicialitzar_button_clicked          (GtkButton       *button,
                                        gpointer         user_data);

void
on_construir_button_clicked             (GtkButton       *button,
                                        gpointer         user_data);

void
on_extendre_button_clicked              (GtkButton       *button,
                                        gpointer         user_data);

void
on_carregar_map_button_clicked          (GtkButton       *button,
                                        gpointer         user_data);

void
on_desar_map_button_clicked             (GtkButton       *button,
                                        gpointer         user_data);

void
```

```
on_marcar_ini_button_clicked           (GtkButton       *button,
                                        gpointer         user_data);

void
on_marcar_fi_button_clicked            (GtkButton       *button,
                                        gpointer         user_data);

void
on_veure_ini_fi_button_clicked         (GtkButton       *button,
                                        gpointer         user_data);

void
on_carregar_infi_button_clicked        (GtkButton       *button,
                                        gpointer         user_data);

void
on_desar_infi_button_clicked           (GtkButton       *button,
                                        gpointer         user_data);

void
on_fitxer_infi_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_cercar_button_clicked               (GtkButton       *button,
                                        gpointer         user_data);

void
on_suavitzar_button_clicked            (GtkButton       *button,
                                        gpointer         user_data);

void
on_carregar_trj_button_clicked         (GtkButton       *button,
                                        gpointer         user_data);

void
on_desar_trj_button_clicked            (GtkButton       *button,
                                        gpointer         user_data);

void
on_fitxer_trj_entry_activate           (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_visualitzar_button_clicked          (GtkButton       *button,
                                        gpointer         user_data);

void
on_memoritzar_config_button_clicked    (GtkButton       *button,
                                        gpointer         user_data);

void
on_borrar_config_button_clicked        (GtkButton       *button,
                                        gpointer         user_data);

void
on_executar_button_clicked             (GtkButton       *button,
                                        gpointer         user_data);

void p_func_f(generic_ptr data);

//Rutines callbacks associades a la finestra d'analisi del graf
void
on_vn_entry_activate                   (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_vertex_i_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_vertex_j_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_n_components_entry_activate         (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_min_analisi_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_max_analisi_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data);

void
on_filtrar_button_clicked              (GtkButton       *button,
                                        gpointer         user_data);

void
on_n_components_button_clicked         (GtkButton       *button,
```

```
                                        gpointer         user_data);

void
on_distancia_analisi_button_clicked     (GtkButton       *button,
                                        gpointer         user_data);

void
on_cercar_analisi_button_clicked        (GtkButton       *button,
                                        gpointer         user_data);

void
on_vertex_seg_button_clicked            (GtkButton       *button,
                                        gpointer         user_data);

void
on_vertex_ant_button_clicked            (GtkButton       *button,
                                        gpointer         user_data);

void set_value_entry_by_button (GtkButton       *button,
                                gpointer         user_data);
```

H.10 callbacks.c

```
#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <gtk/gtk.h>
#include "callbacks.h"
#include "interface.h"
#include "support.h"

#include <unistd.h>
#include <sys/times.h>

////////////////////////////////////////////////////
// Constants
////////////////////////////////////////////////////
#define MAX_TRIEDRES 3000
#define DIRECTORI "/home/gbonals/Haydn/prc/exm/"

////////////////////////////////////////////////////
// Variables globals locals a callbacks.c
////////////////////////////////////////////////////

//Son globals locals èperque no podem passar-les
//com a parametres a les rutines callbacks.
DADES_CONFIG gl_dades_config={{"esquerra"},{"esquerra"},
                              {},{},{},{"run"},{"run"}};

char* gl_taula_conf[8]={"run","ruf","rdn","rdf","lun","luf","ldn","ldf"};
char gl_elem_moure[MAX_CADENA]="robot_esquerra";

char gl_tipus[MAX_CADENA]="esquerra";
bool gl_prensio_on_line=FALSE;
bool gl_borrar_graf=FALSE;
char gl_nom_base[MAX_CADENA]="problema";
char gl_extensio_infi[MAX_CADENA]="";
char gl_extensio_trj[MAX_CADENA]="";

CONFIG* gl_config=NULL;
CONFIG* gl_config_ini=NULL;
CONFIG* gl_config_fi=NULL;

list gl_traject;
list gl_pvertex_ini;
list gl_pvertex_fi;
list gl_pvertex_marca=NULL;

vertex gl_num_vertex=0;
vertex gl_vertex_i=-1;
vertex gl_vertex_j=-1;
int gl_filtrar_min=0;
int gl_filtrar_max=0;

graph gl_G=NULL;
real gl_GMIN=GMIN;
real gl_GMAX=GMAX;

COMPONENT* gl_taula_components;


void
on_gl_llibertat_robot_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
  if (togglebutton->active)
    {
      sprintf(gl_elem_moure,"robot_%s",gl_tipus);
      printf("L'element_que_podem_moureé_s_el_%s\n",gl_elem_moure);
```

```
      /* Si el control llega aqui el boton se encuentra pulsado */

    }
  else {

    /* El boton no esta pulsado (sobresale) */
    }
}

void
on_g_llibertat_objecte_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
if (togglebutton->active)
    {
      sprintf(gl_elem_moure,"%s","objecte");
      printf("L'element que podem mouré s l'%s\n",gl_elem_moure);

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      /* El boton no esta pulsado (sobresale) */
    }
}
void
on_g_llibertat_prensio_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
  if (togglebutton->active)
    {
      sprintf(gl_elem_moure,"prensio_%s",gl_tipus);
      printf("L'element que podem mouré s el triedre que estableix la %s\n",gl_elem_moure);

      /* Si el control llega aqui el boton se encuentra pulsado */

    }
  else {

    /* El boton no esta pulsado (sobresale) */
    }
}
void
on_g_llibertat_pinca_radiobutton_toggled
                                        (GtkToggleButton *togglebutton,
                                         gpointer         user_data)
{
  if (togglebutton->active)
    {
      sprintf(gl_elem_moure,"pinca_%s",gl_tipus);
      printf("L'element que podem obrir s la %s\n",gl_elem_moure);

      /* Si el control llega aqui el boton se encuentra pulsado */

    }
  else {

    /* El boton no esta pulsado (sobresale) */
    }

}

void
on_prensio_on_line_CheckButton_toggled (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{

  if (togglebutton->active)
    {
      gl_prensio_on_line=TRUE;
      printf("Movent l'objecte recalculem la àcinemtica inversa dels dos robots on line\n");
      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      gl_prensio_on_line=FALSE;


      /* El boton no esta pulsado (sobresale) */
    }
}
void
set_value_entry_by_hscale(GtkRange  *range,
                          gpointer user_data)

{
  double g_llibertat;
  char buffer[MAX_CADENA];
  GtkObject* object;
```

```
  gchar text[MAX_CADENA];

  object=(GtkObject*)(user_data);
  g_llibertat=(double)gtk_range_get_value(range);
  gcvt(g_llibertat,3,buffer);
  sprintf(text,"%s",buffer);
  gtk_entry_set_text (GTK_ENTRY(object),text);

}
void
set_value_hscale_by_entry(GtkEntry  *entry,
                          gpointer user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  real g_llibertat;
  GtkRange* range;

  range=(GtkRange*)(user_data);
  text_entrat=gtk_entry_get_text (entry);
  g_llibertat=atof((char*)text_entrat);
  gtk_range_set_value(range,(gdouble)g_llibertat);
}

void
set_value_hscale_to_0(GtkButton *button,
                      gpointer user_data)
{
  gdouble value=0.0;
  GtkRange* range;
  range=(GtkRange*)user_data;
  gtk_range_set_value(range,value);
}
void
on_g_llibertat_1_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data)
{
  CONFIG* config;
  real g_llibertat_1;


  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN1,(gdouble)g_JMAX1);
      g_llibertat_1=(real)gtk_range_get_value(range);

      canviar_angle_solid (ART_ESQ_1,&g_llibertat_1);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"robot_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN1,(gdouble)g_JMAX1);
      g_llibertat_1=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_DRET_1,&g_llibertat_1);
      avancar_simulacio();
    }
 if(!strcmp(gl_elem_moure,"objecte"))
    {
      gtk_range_set_range(range,(gdouble)g_XMIN,(gdouble)g_XMAX);
      g_llibertat_1=(real)gtk_range_get_value(range);
      aplicar_translacio_x_solid(TRANSL_OBJECTE,&g_llibertat_1);
      avancar_simulacio();

      if(gl_prensio_on_line)
        {
          config=crear_configuracio();
          canviar_a_configuracio_on_line(&gl_dades_config,config);
          alliberar_configuracio(&config);
        }
    }
 if(!strcmp(gl_elem_moure,"prensio_esquerra"))
   {
     gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
     g_llibertat_1=(real)gtk_range_get_value(range);
     aplicar_translacio_x_solid(TRANSL_TRIEDRE_ESQ_OBJCT,&g_llibertat_1);
     avancar_simulacio();
   }
 if(!strcmp(gl_elem_moure,"prensio_dret"))
   {
     gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
     g_llibertat_1=(real)gtk_range_get_value(range);
     aplicar_translacio_x_solid(TRANSL_TRIEDRE_DRET_OBJCT,&g_llibertat_1);
     avancar_simulacio();
   }
}
void
on_g_llibertat_2_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data)
{
  CONFIG* config;
  real g_llibertat_2;
```

```
  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN2,(gdouble)g_JMAX2);
      g_llibertat_2=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_ESQ_2,&g_llibertat_2);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"robot_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN2,(gdouble)g_JMAX2);
      g_llibertat_2=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_DRET_2,&g_llibertat_2);
      avancar_simulacio();
    }
 if(!strcmp(gl_elem_moure,"objecte"))
    {
      gtk_range_set_range(range,(gdouble)g_YMIN,(gdouble)g_YMAX);
      g_llibertat_2=(real)gtk_range_get_value(range);
      aplicar_translacio_y_solid(TRANSL_OBJECTE,&g_llibertat_2);
      avancar_simulacio();

      if(gl_prensio_on_line)
        {
         config=crear_configuracio();
         canviar_a_configuracio_on_line(&gl_dades_config,config);
         alliberar_configuracio(&config);
        }
    }
 if(!strcmp(gl_elem_moure,"prensio_esquerra"))
   {
     gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
     g_llibertat_2=(real)gtk_range_get_value(range);
     aplicar_translacio_y_solid(TRANSL_TRIEDRE_ESQ_OBJCT,&g_llibertat_2);
     avancar_simulacio();
   }
  if(!strcmp(gl_elem_moure,"prensio_dret"))
   {
     gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
     g_llibertat_2=(real)gtk_range_get_value(range);
     aplicar_translacio_y_solid(TRANSL_TRIEDRE_DRET_OBJCT,&g_llibertat_2);
     avancar_simulacio();
   }
}
void
on_g_llibertat_3_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data)
{
  CONFIG* config;
  real g_llibertat_3;


  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN3,(gdouble)g_JMAX3);
      g_llibertat_3=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_ESQ_3,&g_llibertat_3);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"robot_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN3,(gdouble)g_JMAX3);
      g_llibertat_3=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_DRET_3,&g_llibertat_3);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"objecte"))
    {
      gtk_range_set_range(range,(gdouble)g_ZMIN,(gdouble)g_ZMAX);
      g_llibertat_3=(real)gtk_range_get_value(range);
      aplicar_translacio_z_solid(TRANSL_OBJECTE,&g_llibertat_3);
      avancar_simulacio();

      if(gl_prensio_on_line)
        {
          config=crear_configuracio();
          canviar_a_configuracio_on_line(&gl_dades_config,config);
          alliberar_configuracio(&config);
        }
    }
 if(!strcmp(gl_elem_moure,"prensio_esquerra"))
   {
     gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
     g_llibertat_3=(real)gtk_range_get_value(range);
     aplicar_translacio_z_solid(TRANSL_TRIEDRE_ESQ_OBJCT,&g_llibertat_3);
     avancar_simulacio();
   }
  if(!strcmp(gl_elem_moure,"prensio_dret"))
   {
     gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
     g_llibertat_3=(real)gtk_range_get_value(range);
     aplicar_translacio_z_solid(TRANSL_TRIEDRE_DRET_OBJCT,&g_llibertat_3);
     avancar_simulacio();
```

```
    }
}
void
on_g_llibertat_4_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data)
{
  CONFIG* config;
  real g_llibertat_4;


  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN4,(gdouble)g_JMAX4);
      g_llibertat_4=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_ESQ_4,&g_llibertat_4);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"robot_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN4,(gdouble)g_JMAX4);
      g_llibertat_4=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_DRET_4,&g_llibertat_4);
      avancar_simulacio();
    }
 if(!strcmp(gl_elem_moure,"objecte"))
    {
      gtk_range_set_range(range,(gdouble)g_ROT_XMIN,(gdouble)g_ROT_XMAX);
      g_llibertat_4=(real)gtk_range_get_value(range);
      canviar_angle_solid (ROT_X_OBJECTE,&g_llibertat_4);
      avancar_simulacio();

      if(gl_prensio_on_line)
        {
         config=crear_configuracio();
         canviar_a_configuracio_on_line(&gl_dades_config,config);
         alliberar_configuracio(&config);
        }
    }
 if(!strcmp(gl_elem_moure,"prensio_esquerra"))
   {
     gtk_range_set_range(range,(gdouble)g_ROT_XMIN,(gdouble)g_ROT_XMAX);
     g_llibertat_4=(real)gtk_range_get_value(range);
     canviar_angle_solid(ROT_X_TRIEDRE_ESQ_OBJCT,&g_llibertat_4);
     avancar_simulacio();
   }
  if(!strcmp(gl_elem_moure,"prensio_dret"))
   {
     gtk_range_set_range(range,(gdouble)g_ROT_XMIN,(gdouble)g_ROT_XMAX);
     g_llibertat_4=(real)gtk_range_get_value(range);
     canviar_angle_solid(ROT_X_TRIEDRE_DRET_OBJCT,&g_llibertat_4);
     avancar_simulacio();
   }
}

void
on_g_llibertat_5_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data)
{
  CONFIG* config;
  real g_llibertat_5;


  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN5,(gdouble)g_JMAX5);
      g_llibertat_5=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_ESQ_5,&g_llibertat_5);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"robot_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN5,(gdouble)g_JMAX5);
      g_llibertat_5=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_DRET_5,&g_llibertat_5);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"objecte"))
    {
      gtk_range_set_range(range,(gdouble)g_ROT_YMIN,(gdouble)g_ROT_YMAX);
      g_llibertat_5=(real)gtk_range_get_value(range);
      canviar_angle_solid (ROT_Y_OBJECTE,&g_llibertat_5);
      avancar_simulacio();

      if(gl_prensio_on_line)
        {
          config=crear_configuracio();
          canviar_a_configuracio_on_line(&gl_dades_config,config);
          alliberar_configuracio(&config);
        }
    }
  if(!strcmp(gl_elem_moure,"prensio_esquerra"))
    {
```

```
      gtk_range_set_range(range,(gdouble)g_ROT_YMIN,(gdouble)g_ROT_YMAX);
      g_llibertat_5=(real)gtk_range_get_value(range);
      canviar_angle_solid(ROT_Y_TRIEDRE_ESQ_OBJCT,&g_llibertat_5);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"prensio_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_ROT_YMIN,(gdouble)g_ROT_YMAX);
      g_llibertat_5=(real)gtk_range_get_value(range);
      canviar_angle_solid(ROT_Y_TRIEDRE_DRET_OBJCT,&g_llibertat_5);
      avancar_simulacio();
    }
}
void
on_g_llibertat_6_hscale_value_changed  (GtkRange        *range,
                                        gpointer         user_data)
{
  CONFIG* config;
  real g_llibertat_6;


  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN6,(gdouble)g_JMAX6);
      g_llibertat_6=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_ESQ_6,&g_llibertat_6);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"robot_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_JMIN6,(gdouble)g_JMAX6);
      g_llibertat_6=(real)gtk_range_get_value(range);
      canviar_angle_solid (ART_DRET_6,&g_llibertat_6);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"objecte"))
    {
      gtk_range_set_range(range,(gdouble)g_ROT_ZMIN,(gdouble)g_ROT_ZMAX);
      g_llibertat_6=(real)gtk_range_get_value(range);
      canviar_angle_solid (ROT_Z_OBJECTE,&g_llibertat_6);
      avancar_simulacio();

      if(gl_prensio_on_line)
        {
          config=crear_configuracio();
          canviar_a_configuracio_on_line(&gl_dades_config,config);
          alliberar_configuracio(&config);
        }
    }
  if(!strcmp(gl_elem_moure,"prensio_esquerra"))
    {
      gtk_range_set_range(range,(gdouble)g_ROT_ZMIN,(gdouble)g_ROT_ZMAX);
      g_llibertat_6=(real)gtk_range_get_value(range);
      canviar_angle_solid(ROT_Z_TRIEDRE_ESQ_OBJCT,&g_llibertat_6);
      avancar_simulacio();
    }
  if(!strcmp(gl_elem_moure,"prensio_dret"))
    {
      gtk_range_set_range(range,(gdouble)g_ROT_ZMIN,(gdouble)g_ROT_ZMAX);
      g_llibertat_6=(real)gtk_range_get_value(range);
      canviar_angle_solid(ROT_Z_TRIEDRE_DRET_OBJCT,&g_llibertat_6);
      avancar_simulacio();
    }
}

void
on_obertura_pinca_hscale_value_changed (GtkRange        *range,
                                        gpointer         user_data)
{
  printf("obrint_çpina...\n");
}
void
on_g_sis_hscale_refresc_by_left_rob_scene(GtkToggleButton *togglebutton,
                                          gpointer         user_data)
{
  SIS_WIDGETS* g_sis_hscale;
  GtkRange* range;
  gdouble g_AUXMIN=-10.0;
  gdouble g_AUXMAX=10.0;

  g_sis_hscale=(SIS_WIDGETS*)user_data;


  if(!strcmp(gl_elem_moure,"robot_esquerra"))
    {
      obtenir_angles_robot ("esquerra");
      avancar_simulacio();

      range=(GtkRange*)(g_sis_hscale->g_widget_1);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_esq[0]);
      gtk_range_set_range(range,(gdouble)g_JMIN1,(gdouble)g_JMAX1);
```

```
      range=(GtkRange*)(g_sis_hscale->g_widget_2);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_esq[1]);
      gtk_range_set_range(range,(gdouble)g_JMIN2,(gdouble)g_JMAX2);

      range=(GtkRange*)(g_sis_hscale->g_widget_3);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_esq[2]);
      gtk_range_set_range(range,(gdouble)g_JMIN3,(gdouble)g_JMAX3);


      range=(GtkRange*)(g_sis_hscale->g_widget_4);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_esq[3]);
      gtk_range_set_range(range,(gdouble)g_JMIN4,(gdouble)g_JMAX4);


      range=(GtkRange*)(g_sis_hscale->g_widget_5);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_esq[4]);
      gtk_range_set_range(range,(gdouble)g_JMIN5,(gdouble)g_JMAX5);


      range=(GtkRange*)(g_sis_hscale->g_widget_6);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_esq[5]);
      gtk_range_set_range(range,(gdouble)g_JMIN6,(gdouble)g_JMAX6);
    }

  else if (!strcmp(gl_elem_moure,"robot_dret"))
    {
      obtenir_angles_robot ("dret");
      avancar_simulacio();

      range=(GtkRange*)(g_sis_hscale->g_widget_1);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_dret[0]);
      gtk_range_set_range(range,(gdouble)g_JMIN1,(gdouble)g_JMAX1);

      range=(GtkRange*)(g_sis_hscale->g_widget_2);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_dret[1]);
      gtk_range_set_range(range,(gdouble)g_JMIN2,(gdouble)g_JMAX2);

      range=(GtkRange*)(g_sis_hscale->g_widget_3);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_dret[2]);
      gtk_range_set_range(range,(gdouble)g_JMIN3,(gdouble)g_JMAX3);


      range=(GtkRange*)(g_sis_hscale->g_widget_4);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_dret[3]);
      gtk_range_set_range(range,(gdouble)g_JMIN4,(gdouble)g_JMAX4);


      range=(GtkRange*)(g_sis_hscale->g_widget_5);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_dret[4]);
      gtk_range_set_range(range,(gdouble)g_JMIN5,(gdouble)g_JMAX5);


      range=(GtkRange*)(g_sis_hscale->g_widget_6);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)angle_rob_dret[5]);
      gtk_range_set_range(range,(gdouble)g_JMIN6,(gdouble)g_JMAX6);
    }

  else if (!strcmp(gl_elem_moure,"prensio_esquerra"))
    {
      obtenir_translacio_solid(TRANSL_TRIEDRE_ESQ_OBJCT,transl_grasp_esq);
      obtenir_angle_solid (ROT_X_TRIEDRE_ESQ_OBJCT,&(rot_grasp_esq[0]));
      obtenir_angle_solid (ROT_Y_TRIEDRE_ESQ_OBJCT,&(rot_grasp_esq[1]));
      obtenir_angle_solid (ROT_Z_TRIEDRE_ESQ_OBJCT,&(rot_grasp_esq[2]));
      avancar_simulacio();

      range=(GtkRange*)(g_sis_hscale->g_widget_1);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)transl_grasp_esq[0]);
      gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_2);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)transl_grasp_esq[1]);
      gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_3);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)transl_grasp_esq[2]);
      gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);
```

```
      range=(GtkRange*)(g_sis_hscale->g_widget_4);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)rot_grasp_esq[0]);
      gtk_range_set_range(range,(gdouble)g_ROT_XMIN,(gdouble)g_ROT_XMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_5);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)rot_grasp_esq[1]);
      gtk_range_set_range(range,(gdouble)g_ROT_YMIN,(gdouble)g_ROT_YMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_6);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)rot_grasp_esq[2]);
      gtk_range_set_range(range,(gdouble)g_ROT_ZMIN,(gdouble)g_ROT_ZMAX);
    }
  else if (!strcmp(gl_elem_moure,"prensio_dret"))
    {
      obtenir_translacio_solid(TRANSL_TRIEDRE_DRET_OBJCT,transl_grasp_dret);
      obtenir_angle_solid (ROT_X_TRIEDRE_DRET_OBJCT,&(rot_grasp_dret[0]));
      obtenir_angle_solid (ROT_Y_TRIEDRE_DRET_OBJCT,&(rot_grasp_dret[1]));
      obtenir_angle_solid (ROT_Z_TRIEDRE_DRET_OBJCT,&(rot_grasp_dret[2]));
      avancar_simulacio();

      range=(GtkRange*)(g_sis_hscale->g_widget_1);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)transl_grasp_dret[0]);
      gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_2);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)transl_grasp_dret[1]);
      gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_3);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)transl_grasp_dret[2]);
      gtk_range_set_range(range,(gdouble)gl_GMIN,(gdouble)gl_GMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_4);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)rot_grasp_dret[0]);
      gtk_range_set_range(range,(gdouble)g_ROT_XMIN,(gdouble)g_ROT_XMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_5);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)rot_grasp_dret[1]);
      gtk_range_set_range(range,(gdouble)g_ROT_YMIN,(gdouble)g_ROT_YMAX);

      range=(GtkRange*)(g_sis_hscale->g_widget_6);
      gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
      gtk_range_set_value(range,(gdouble)rot_grasp_dret[2]);
      gtk_range_set_range(range,(gdouble)g_ROT_ZMIN,(gdouble)g_ROT_ZMAX);
    }

   else if (!strcmp(gl_elem_moure,"pinca_esquerra"))
    {

    }
  else if (!strcmp(gl_elem_moure,"pinca_dret"))
    {

    }
}

void
on_g_sis_hscale_refresc_by_object_scene(GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
  SIS_WIDGETS* g_sis_hscale;
  GtkRange* range;
  gdouble g_AUXMIN=-10.0;
  gdouble g_AUXMAX=10.0;

  g_sis_hscale=(SIS_WIDGETS*)user_data;

  obtenir_posicio_objecte();
  avancar_simulacio();

  range=(GtkRange*)(g_sis_hscale->g_widget_1);
  gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
  gtk_range_set_value(range,(gdouble)transl_objecte[0]);
  gtk_range_set_range(range,(gdouble)g_XMIN,(gdouble)g_XMAX);

  range=(GtkRange*)(g_sis_hscale->g_widget_2);
  gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
  gtk_range_set_value(range,(gdouble)transl_objecte[1]);
  gtk_range_set_range(range,(gdouble)g_YMIN,(gdouble)g_YMAX);

  range=(GtkRange*)(g_sis_hscale->g_widget_3);
  gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
  gtk_range_set_value(range,(gdouble)transl_objecte[2]);
  gtk_range_set_range(range,(gdouble)g_ZMIN,(gdouble)g_ZMAX);
```

```
  range=(GtkRange*)(g_sis_hscale->g_widget_4);
  gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
  gtk_range_set_value(range,(gdouble)rot_objecte[0]);
  gtk_range_set_range(range,(gdouble)g_ROT_XMIN,(gdouble)g_ROT_XMAX);

  range=(GtkRange*)(g_sis_hscale->g_widget_5);
  gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
  gtk_range_set_value(range,(gdouble)rot_objecte[1]);
  gtk_range_set_range(range,(gdouble)g_ROT_YMIN,(gdouble)g_ROT_YMAX);

  range=(GtkRange*)(g_sis_hscale->g_widget_6);
  gtk_range_set_range(range,(gdouble)g_AUXMIN,(gdouble)g_AUXMAX);
  gtk_range_set_value(range,(gdouble)rot_objecte[2]);
  gtk_range_set_range(range,(gdouble)g_ROT_ZMIN,(gdouble)g_ROT_ZMAX);
}
void
on_esq_radiobutton_toggled     (GtkToggleButton *togglebutton,
                                gpointer         user_data)
{
  TRES_TOGGLE* g_tres_toggle;

  g_tres_toggle=(TRES_TOGGLE*)user_data;

  if (togglebutton->active)
    {
      sprintf(gl_tipus,"%s","esquerra");
      printf("%s\n",gl_tipus);

      /* Si el control llega aqui el boton se encuentra pulsado */

      if (((GtkToggleButton *)(g_tres_toggle->robot_togglebutton))->active)
        {
          sprintf(gl_elem_moure,"robot_esquerra");
          printf("L'element_a_moureé_s_el_%s\n",gl_elem_moure);
        }
      else if (((GtkToggleButton *)(g_tres_toggle->prensio_togglebutton))->active)
        {
          sprintf(gl_elem_moure,"prensio_esquerra");
          printf("L'element_a_moureé_s_el_triedre_que_estableix_la_%s\n",gl_elem_moure);
        }
      else if (((GtkToggleButton *)(g_tres_toggle->pinca_togglebutton))->active)
        {
          sprintf(gl_elem_moure,"pinca_esquerra");
          printf("L'element_a_obriré_s_la_%s\n",gl_elem_moure);
        }
      else
        {

        }
    }
  else
    {

    }
}

void
on_carregar_par_button_clicked (GtkButton       *button,
                                gpointer         user_data)
{
  FILE* fp;
  WIDGETS_PARAMETRES* widgets_parametres;
  GtkObject* object;
  GtkToggleButton *togglebutton;
  char directori[MAX_CADENA]=DIRECTORI;
  char nom_fitxer[MAX_CADENA];
  G_CONST_RETURN gchar text_entrat[MAX_CADENA];

  char buffer[MAX_CADENA];
  gchar text[MAX_CADENA];

  sprintf(nom_fitxer,"%s%s.par",directori,gl_nom_base);
  widgets_parametres=(WIDGETS_PARAMETRES*)user_data;

  if ((fp=fopen(nom_fitxer,"r"))==NULL)
    printf("no_podem_obrir_%s_per_llegir-lo\n",nom_fitxer);
  else
    {
      fscanf(fp,"Fase_d'Aprenentatge\n");
      fscanf(fp,"----------------------\n");
      fscanf(fp,"max._nodes:%d\n",&g_MAX_NODES);
      fscanf(fp,"%%_expansio:%f\n",&g_PERC_EXP);
      fscanf(fp,"min._nodes:%d\n",&g_MIN_NODES);
      fscanf(fp,"num._veins:%d\n",&g_NVEINS);
      fscanf(fp,"max._dist:%f\n",&g_MAX_DIST);
      fscanf(fp,"generacio_aleat:%s\n",gl_dades_config.elem_aleat);
      fscanf(fp,"Cami_local\n");
      fscanf(fp,"----------------------\n");
      fscanf(fp,"pas_cami_local:%f\n",&g_pas_cami_local);
      fscanf(fp,"elem._a_interpolar:%s\n",gl_dades_config.elem_interp);
      fscanf(fp,"Fase_Cerca/Expansio\n");
```

```
fscanf(fp,"----------\n");
fscanf(fp,"num. pasos:%d\n",&g_N_PASOS);
fscanf(fp,"fora dist. max.:%d\n",&g_N_FORA_DIST_MAX);
fscanf(fp,"max. no local:%d\n",&g_MAX_NO_CAMI_LOCAL);
fscanf(fp,"max. profunditat:%d\n",&g_max_profund);

//fscanf(fp,"nom. fitxer de depuracio:%d\n",&g_MAX_NO_CAMI_LOCAL);
fscanf(fp,"Robots\n");
fscanf(fp,"----------\n");
fscanf(fp,"angle 1 min:%f, angle 1 max:%f\n",&(g_JMIN1),&(g_JMAX1));
fscanf(fp,"angle 2 min:%f, angle 2 max:%f\n",&(g_JMIN2),&(g_JMAX2));
fscanf(fp,"angle 3 min:%f, angle 3 max:%f\n",&(g_JMIN3),&(g_JMAX3));
fscanf(fp,"angle 4 min:%f, angle 4 max:%f\n",&(g_JMIN4),&(g_JMAX4));
fscanf(fp,"angle 5 min:%f, angle 5 max:%f\n",&(g_JMIN5),&(g_JMAX5));
fscanf(fp,"angle 6 min:%f, angle 6 max:%f\n",&(g_JMIN6),&(g_JMAX6));

gl_dades_config.jr.jmax_c.th1=g_JMAX1;
gl_dades_config.jr.jmax_c.th2=g_JMAX2;
gl_dades_config.jr.jmax_c.th3=g_JMAX3;
gl_dades_config.jr.jmax_c.th4=g_JMAX4;
gl_dades_config.jr.jmax_c.th5=g_JMAX5;
gl_dades_config.jr.jmax_c.th6=g_JMAX6;


gl_dades_config.jr.jmin_c.th1=g_JMIN1;
gl_dades_config.jr.jmin_c.th2=g_JMIN2;
gl_dades_config.jr.jmin_c.th3=g_JMIN3;
gl_dades_config.jr.jmin_c.th4=g_JMIN4;
gl_dades_config.jr.jmin_c.th5=g_JMIN5;
gl_dades_config.jr.jmin_c.th6=g_JMIN6;



fscanf(fp,"config. robot esq:%s config. robot dret:%s\n",
       gl_dades_config.conf_rob_esq,gl_dades_config.conf_rob_dret);
fscanf(fp,"Objecte\n");
fscanf(fp,"----------\n");
fscanf(fp,"angle x min:%f, angle x max:%f\n",&g_XMIN,&g_XMAX);
fscanf(fp,"angle y min:%f, angle y max:%f\n",&g_YMIN,&g_YMAX);
fscanf(fp,"angle z min:%f, angle z max:%f\n",&g_ZMIN,&g_ZMAX);
fscanf(fp,"rot angle x min:%f, rot angle x max:%f\n",&g_ROT_XMIN,&g_ROT_XMAX);
fscanf(fp,"rot angle y min:%f, rot angle y max:%f\n",&g_ROT_YMIN,&g_ROT_YMAX);
fscanf(fp,"rot angle z min:%f, rot angle z max:%f\n",&g_ROT_ZMIN,&g_ROT_ZMAX);
fscanf(fp,"Prensio\n");
fscanf(fp,"----------\n");
fscanf(fp,"coord cart min:%f, coord cart max:%f\n",&gl_GMIN,&gl_GMAX);
fclose(fp);


object=(GtkObject*)(widgets_parametres->max_nodes_entry);

gcvt((double)g_MAX_NODES,3,buffer);
//printf("%s\n",buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->percen_expansio_entry);

gcvt((double)g_PERC_EXP,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->min_nodes_entry);

gcvt((double)g_MIN_NODES,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->num_veins_entry);

gcvt((double)g_NVEINS,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->max_dist_entry);

gcvt((double)g_MAX_DIST,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->pas_cami_local_entry);

gcvt((double)g_pas_cami_local,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->num_pasos_entry);

gcvt((double)g_N_PASOS,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);
```

```
object=(GtkObject*)(widgets_parametres->fora_dist_max_entry);

gcvt((double)g_N_FORA_DIST_MAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->max_no_local_entry);

gcvt((double)g_MAX_NO_CAMI_LOCAL,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->max_profunditat_entry);

gcvt((double)g_max_profund,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);



object=(GtkObject*)(widgets_parametres->angle_1_min_entry);

gcvt((double)(g_JMIN1),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->angle_2_min_entry);

gcvt((double)(g_JMIN2),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_3_min_entry);

gcvt((double)(g_JMIN3),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_4_min_entry);

gcvt((double)(g_JMIN4),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_5_min_entry);

gcvt((double)(g_JMIN5),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_6_min_entry);

gcvt((double)(g_JMIN6),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_1_max_entry);

gcvt((double)(g_JMAX1),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->angle_2_max_entry);

gcvt((double)(g_JMAX2),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_3_max_entry);

gcvt((double)(g_JMAX3),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_4_max_entry);

gcvt((double)(g_JMAX4),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_5_max_entry);

gcvt((double)(g_JMAX5),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->angle_6_max_entry);
```

```
gcvt((double)(g_JMAX6),3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->x_max_entry);

gcvt((double)g_XMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->y_max_entry);

gcvt((double)g_YMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->z_max_entry);

gcvt((double)g_ZMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->rot_x_max_entry);

gcvt((double)g_ROT_XMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->rot_y_max_entry);

gcvt((double)g_ROT_YMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->rot_z_max_entry);

gcvt((double)g_ROT_ZMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


 object=(GtkObject*)(widgets_parametres->x_min_entry);

gcvt((double)g_XMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->y_min_entry);

gcvt((double)g_YMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->z_min_entry);

gcvt((double)g_ZMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->rot_x_min_entry);

gcvt((double)g_ROT_XMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->rot_y_min_entry);

gcvt((double)g_ROT_YMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->rot_z_min_entry);

gcvt((double)g_ROT_ZMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);


object=(GtkObject*)(widgets_parametres->coord_cart_min_prensio_entry);

gcvt((double)gl_GMIN,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);

object=(GtkObject*)(widgets_parametres->coord_cart_max_prensio_entry);

gcvt((double)gl_GMAX,3,buffer);
sprintf(text,"%s",buffer);
gtk_entry_set_text (GTK_ENTRY(object),text);
```

```
      object=(GtkObject*)(widgets_parametres->config_rob_esq_entry);
      memcpy ((void*)text_entrat,(void*)gl_dades_config.conf_rob_esq,
              strlen(gl_dades_config.conf_rob_esq)+1);
      gtk_entry_set_text (GTK_ENTRY(object),text_entrat);

      object=(GtkObject*)(widgets_parametres->config_rob_dret_entry);
      memcpy ((void*)text_entrat,(void*)gl_dades_config.conf_rob_dret,
              strlen(gl_dades_config.conf_rob_dret)+1);
      gtk_entry_set_text (GTK_ENTRY(object),text_entrat);


      if (!strcmp(gl_dades_config.elem_aleat,"esquerra"))
        {
          togglebutton=(GtkToggleButton*)(widgets_parametres->rob_esq_aleat_radiobutton);
          gtk_toggle_button_set_active (togglebutton,TRUE);
        }
      else if (!strcmp(gl_dades_config.elem_aleat,"dret"))
        {
          togglebutton=(GtkToggleButton*)(widgets_parametres->rob_dret_aleat_radiobutton);
          gtk_toggle_button_set_active (togglebutton,TRUE);
        }
      else if (!strcmp(gl_dades_config.elem_aleat,"objecte"))
        {
          togglebutton=(GtkToggleButton*)(widgets_parametres->objecte_aleat_radiobutton);
          gtk_toggle_button_set_active (togglebutton,TRUE);
        }

       if (!strcmp(gl_dades_config.elem_interp,"esquerra"))
        {
          togglebutton=(GtkToggleButton*)(widgets_parametres->rob_esq_interpol_radiobutton);
          gtk_toggle_button_set_active (togglebutton,TRUE);
        }
      else if (!strcmp(gl_dades_config.elem_interp,"dret"))
        {
          togglebutton=(GtkToggleButton*)(widgets_parametres->rob_dret_interpol_radiobutton);
          gtk_toggle_button_set_active (togglebutton,TRUE);
        }
    }
}

void
on_dret_radiobutton_toggled    (GtkToggleButton *togglebutton,
                                gpointer         user_data)
{
  TRES_TOGGLE* g_tres_toggle;

  g_tres_toggle=(TRES_TOGGLE*)user_data;

  if (togglebutton->active)
    {
      sprintf(gl_tipus,"%s","dret");
      printf("%s\n",gl_tipus);

      /* Si el control llega aqui el boton se encuentra pulsado */

      if (((GtkToggleButton *)(g_tres_toggle->robot_togglebutton))->active)
        {
          sprintf(gl_elem_moure,"%s","robot_dret");
          printf("%s\n",gl_elem_moure);
        }
      else if (((GtkToggleButton *)(g_tres_toggle->prensio_togglebutton))->active)
        {
          sprintf(gl_elem_moure,"%s","prensio_dret");
          printf("%s\n",gl_elem_moure);
        }
      else if (((GtkToggleButton *)(g_tres_toggle->pinca_togglebutton))->active)
        {
          sprintf(gl_elem_moure,"%s","pinca_dret");
          printf("%s\n",gl_elem_moure);
        }
      else
        {


        }
    }
  else
    {

    }
}

void
on_convertir_labels_a_g_llibertat_robot_radiobutton_toggled(GtkToggleButton *togglebutton,
                                                            gpointer         user_data)
{
  GtkLabel* label;
  SIS_WIDGETS* g_sis_label;

  g_sis_label=(SIS_WIDGETS*)user_data;
```

```
  label=(GtkLabel*)(g_sis_label->g_widget_1);
  gtk_label_set_label (label,"angle_1");

  label=(GtkLabel*)(g_sis_label->g_widget_2);
  gtk_label_set_label (label,"angle_2");

  label=(GtkLabel*)(g_sis_label->g_widget_3);
  gtk_label_set_label (label,"angle_3");

  label=(GtkLabel*)(g_sis_label->g_widget_4);
  gtk_label_set_label (label,"angle_4");

  label=(GtkLabel*)(g_sis_label->g_widget_5);
  gtk_label_set_label (label,"angle_5");

  label=(GtkLabel*)(g_sis_label->g_widget_6);
  gtk_label_set_label (label,"angle_6");
}

void
on_convertir_labels_a_g_llibertat_solid_radiobutton_toggled(GtkToggleButton *togglebutton,
                                                            gpointer         user_data)
{
  GtkLabel* label;
  SIS_WIDGETS* g_sis_label;

  g_sis_label=(SIS_WIDGETS*)user_data;

  label=(GtkLabel*)(g_sis_label->g_widget_1);
  gtk_label_set_label (label,"trans_x");

  label=(GtkLabel*)(g_sis_label->g_widget_2);
  gtk_label_set_label (label,"trans_y");

  label=(GtkLabel*)(g_sis_label->g_widget_3);
  gtk_label_set_label (label,"trans_z");

  label=(GtkLabel*)(g_sis_label->g_widget_4);
  gtk_label_set_label (label,"rot_x");

  label=(GtkLabel*)(g_sis_label->g_widget_5);
  gtk_label_set_label (label,"rot_y");

  label=(GtkLabel*)(g_sis_label->g_widget_6);
  gtk_label_set_label (label,"rot_z");
}
//funcions associades a la finestra de parametres
void
on_borrar_graf_checkbutton_toggled    (GtkToggleButton *togglebutton,
                                       gpointer         user_data)
{

 if (togglebutton->active)
    {
      gl_borrar_graf=TRUE;

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      gl_borrar_graf=FALSE;

      /* El boton no esta pulsado (sobresale) */
    }

}
void
on_nom_base_entry_activate         (GtkEntry        *entry,
                                    gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  sprintf(gl_nom_base,"%s",(char*)text_entrat);
}
void
on_max_nodes_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
 G_CONST_RETURN gchar* text_entrat;
 text_entrat=gtk_entry_get_text (entry);
 g_MAX_NODES=(int)atoi((char*)text_entrat);
}
void
on_percen_expansio_entry_activate      (GtkEntry        *entry,
                                        gpointer         user_data)
{
 G_CONST_RETURN gchar* text_entrat;
 text_entrat=gtk_entry_get_text (entry);
 g_PERC_EXP=(real)atof((char*)text_entrat);
}
void
```

```
on_min_nodes_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_MIN_NODES=(int)atoi((char*)text_entrat);
}

void
on_num_veins_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_NVEINS=(int)atoi((char*)text_entrat);
}

void
on_max_dist_entry_activate             (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_MAX_DIST=(real)atof((char*)text_entrat);
}
void
on_rob_esq_aleat_radiobutton_toggled   (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
   if (togglebutton->active)
    {
      sprintf(gl_dades_config.elem_aleat,"%s","esquerra");
      printf("El robot %s às er el que generarem òaleatriament\n",gl_dades_config.elem_aleat);

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      /* El boton no esta pulsado (sobresale) */
    }
}

void
on_rob_dret_aleat_radiobutton_toggled  (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
  if (togglebutton->active)
    {

      sprintf(gl_dades_config.elem_aleat,"%s","dret");
      printf("El robot %s às er el que generarem òaleatriament\n",gl_dades_config.elem_aleat);

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      /* El boton no esta pulsado (sobresale) */
    }
}
void
on_objecte_aleat_radiobutton_toggled   (GtkToggleButton *togglebutton,
                                        gpointer         user_data)
{
  if (togglebutton->active)
    {

      sprintf(gl_dades_config.elem_aleat,"%s","objecte");
      printf("L'%s às er el que generarem òaleatriament\n",gl_dades_config.elem_aleat);

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      /* El boton no esta pulsado (sobresale) */
    }
}
void
on_pas_cami_local_entry_activate       (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_pas_cami_local=(real)atof((char*)text_entrat);
}
void
on_rob_esq_interpol_radiobutton_toggled  (GtkToggleButton *togglebutton,
                                          gpointer         user_data)
{
```

```
  if (togglebutton->active)
    {

      sprintf(gl_dades_config.elem_interp,"%s","esquerra");
      printf("El robot %s àser el que interpolarem\n",gl_dades_config.elem_interp);

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      /* El boton no esta pulsado (sobresale) */
    }

}

void
on_rob_dret_interpol_radiobutton_toggled (GtkToggleButton *togglebutton,
                                          gpointer         user_data)
{
  if (togglebutton->active)
    {

      sprintf(gl_dades_config.elem_interp,"%s","dret");
      printf("El robot %s àser el que interpolarem\n",gl_dades_config.elem_interp);

      /* Si el control llega aqui el boton se encuentra pulsado */

    } else {

      /* El boton no esta pulsado (sobresale) */
    }
}
void
on_num_pasos_entry_activate           (GtkEntry        *entry,
                                       gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_N_PASOS=(int)atoi((char*)text_entrat);
}
void
on_fora_dist_max_entry_activate           (GtkEntry        *entry,
                                          gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_N_FORA_DIST_MAX=(int)atoi((char*)text_entrat);
}

void
on_max_no_local_entry_activate        (GtkEntry        *entry,
                                       gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_MAX_NO_CAMI_LOCAL=(int)atoi((char*)text_entrat);
}

void
on_max_profunditat_entry_activate       (GtkEntry        *entry,
                                         gpointer         user_data)

{
  G_CONST_RETURN gchar* text_entrat;
  text_entrat=gtk_entry_get_text (entry);
  g_max_profund=(int)atoi((char*)text_entrat);
}
//void
/* on_max_conn_paths_entry_activate        (GtkEntry        *entry, */
/*                                          gpointer         user_data) */
/* { */
/*   G_CONST_RETURN gchar* text_entrat; */
/*   text_entrat=gtk_entry_get_text (entry); */
/*   g_NUM_COMPONENTS=(int)atoi((char*)text_entrat); */
/*   printf("g_NUM_COMPONENTS:%d\n",g_NUM_COMPONENTS); */
/* } */

void
on_nom_fitxer_debug_entry_activate       (GtkEntry        *entry,
                                          gpointer         user_data)
{

}

void
on_desar_par_button_clicked(GtkButton        *button,
                            gpointer         user_data)
{
  FILE* fp;
```

```
  char directori[MAX_CADENA]=DIRECTORI;
  char nom_fitxer[MAX_CADENA];

  sprintf(nom_fitxer,"%s%s.par",directori,gl_nom_base);

  if ((fp=fopen(nom_fitxer,"w"))==NULL)
    printf("No puc obrir %s per escriure'l\n",nom_fitxer);
  else
    {
      fprintf(fp,"Fase d'Aprenentatge\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"max. nodes:%d\n",g_MAX_NODES);
      fprintf(fp,"%% expansio:%f\n",g_PERC_EXP);
      fprintf(fp,"min. nodes:%d\n",g_MIN_NODES);
      fprintf(fp,"num. veins:%d\n",g_NVEINS);
      fprintf(fp,"max. dist:%f\n",g_MAX_DIST);
      fprintf(fp,"generacio aleat:%s\n",gl_dades_config.elem_aleat);
      fprintf(fp,"Cami local\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"pas cami local:%f\n",g_pas_cami_local);
      fprintf(fp,"elem. a interpolar:%s\n",gl_dades_config.elem_interp);
      fprintf(fp,"Fase Cerca/Expansio\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"num. pasos:%d\n",g_N_PASOS);
      fprintf(fp,"fora dist. max.:%d\n",g_N_FORA_DIST_MAX);
      fprintf(fp,"max. no local:%d\n",g_MAX_NO_CAMI_LOCAL);
      fprintf(fp,"max. profunditat:%d\n",g_max_profund);
      //fprintf(fp,"nom. fitxer de depuracio:%d\n",g_MAX_NO_CAMI_LOCAL);
      fprintf(fp,"Robots\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"angle 1 min:%f, angle 1 max:%f\n",(real)(g_JMIN1),(real)(g_JMAX1));
      fprintf(fp,"angle 2 min:%f, angle 2 max:%f\n",(real)(g_JMIN2),(real)(g_JMAX2));
      fprintf(fp,"angle 3 min:%f, angle 3 max:%f\n",(real)(g_JMIN3),(real)(g_JMAX3));
      fprintf(fp,"angle 4 min:%f, angle 4 max:%f\n",(real)(g_JMIN4),(real)(g_JMAX4));
      fprintf(fp,"angle 5 min:%f, angle 5 max:%f\n",(real)(g_JMIN5),(real)(g_JMAX5));
      fprintf(fp,"angle 6 min:%f, angle 6 max:%f\n",(real)(g_JMIN6),(real)(g_JMAX6));
      fprintf(fp,"config. robot esq:%s config. robot dret:%s\n",
              gl_dades_config.conf_rob_esq,gl_dades_config.conf_rob_dret);
      fprintf(fp,"Objecte\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"angle x min:%f, angle x max:%f\n",g_XMIN,g_XMAX);
      fprintf(fp,"angle y min:%f, angle y max:%f\n",g_YMIN,g_YMAX);
      fprintf(fp,"angle z min:%f, angle z max:%f\n",g_ZMIN,g_ZMAX);
      fprintf(fp,"rot angle x min:%f, rot angle x max:%f\n",g_ROT_XMIN,g_ROT_XMAX);
      fprintf(fp,"rot angle y min:%f, rot angle y max:%f\n",g_ROT_YMIN,g_ROT_YMAX);
      fprintf(fp,"rot angle z min:%f, rot angle z max:%f\n",g_ROT_ZMIN,g_ROT_ZMAX);
      fprintf(fp,"Prensio\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"coord cart min:%f, coord cart max:%f\n",gl_GMIN,gl_GMAX);
      fclose(fp);
    }
}

void
on_angle_1_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMIN1=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmin_c.th1=g_JMIN1;
}

void
on_angle_1_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMAX1=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmax_c.th1=g_JMAX1;
}
void
on_angle_2_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMIN2=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmin_c.th2=g_JMIN2;
}
void
on_angle_2_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMAX2=(real)atof((char*)text_entrat);
```

```
  gl_dades_config.jr.jmax_c.th2=g_JMAX2;

}
void
on_angle_3_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMIN3=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmin_c.th3=g_JMIN3;
}

void
on_angle_3_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMAX3=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmax_c.th3=g_JMAX3;

}

void
on_angle_4_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMIN4=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmin_c.th4=g_JMIN4;
}
void
on_angle_4_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMAX4=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmax_c.th4=g_JMAX4;
}
void
on_angle_5_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMIN5=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmin_c.th5=g_JMIN5;
}
void
on_angle_5_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMAX5=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmax_c.th5=g_JMAX5;
}
void
on_angle_6_min_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMIN6=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmin_c.th6=g_JMIN6;
}
void
on_angle_6_max_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_JMAX6=(real)atof((char*)text_entrat);
  gl_dades_config.jr.jmax_c.th6=g_JMAX6;
}

void
on_config_rob_esq_entry_activate       (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
```

```
  text_entrat=gtk_entry_get_text (entry);
  sprintf(gl_dades_config.conf_rob_esq,"%s",(char*)text_entrat);
}
void
on_config_rob_dret_entry_activate      (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  sprintf(gl_dades_config.conf_rob_dret,"%s",(char*)text_entrat);
}

void
on_x_max_entry_activate                (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_XMAX=(real)atof((char*)text_entrat);
}
void
on_x_min_entry_activate                (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_XMIN=(real)atof((char*)text_entrat);
}
void
on_y_max_entry_activate                (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_YMAX=(real)atof((char*)text_entrat);
}
void
on_y_min_entry_activate                (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_YMIN=(real)atof((char*)text_entrat);
}
void
on_z_max_entry_activate                (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_ZMAX=(real)atof((char*)text_entrat);
}
void
on_z_min_entry_activate                (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_ZMIN=(real)atof((char*)text_entrat);
}
void
on_rot_x_max_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
 G_CONST_RETURN gchar* text_entrat;

 text_entrat=gtk_entry_get_text (entry);
 g_ROT_XMAX=(real)atof((char*)text_entrat);
}
void
on_rot_x_min_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_ROT_XMIN=(real)atof((char*)text_entrat);
}
void
on_rot_y_max_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
```

```
  text_entrat=gtk_entry_get_text (entry);
  g_ROT_YMAX=(real)atof((char*)text_entrat);
}
void
on_rot_y_min_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
 G_CONST_RETURN gchar* text_entrat;

 text_entrat=gtk_entry_get_text (entry);
 g_ROT_YMIN=(real)atof((char*)text_entrat);
}
void
on_rot_z_max_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_ROT_ZMAX=(real)atof((char*)text_entrat);
}
void
on_rot_z_min_entry_activate            (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  g_ROT_ZMIN=(real)atof((char*)text_entrat);
}
void
on_coord_cart_min_prensio_entry_activate
                                       (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  gl_GMIN=(real)atof((char*)text_entrat);
}

void
on_coord_cart_max_prensio_entry_activate
                                        (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  gl_GMAX=(real)atof((char*)text_entrat);
}

//funcions de callback associades a la tercera finestra, a la de tasques.
//void

void
on_inicialitzar_button_clicked         (GtkButton       *button,
                                        gpointer         user_data)
{
  gl_dades_config.pdh.a2=A2;
  gl_dades_config.pdh.a3=A3;
  gl_dades_config.pdh.d3=D3;
  gl_dades_config.pdh.d4=D4;
  gl_dades_config.pdh.d6=D6;

  gl_dades_config.jr.jofst_c.th1=JOFST1;
  gl_dades_config.jr.jofst_c.th2=JOFST2;
  gl_dades_config.jr.jofst_c.th3=JOFST3;
  gl_dades_config.jr.jofst_c.th4=JOFST4;
  gl_dades_config.jr.jofst_c.th5=JOFST5;
  gl_dades_config.jr.jofst_c.th6=JOFST6;

  gl_dades_config.jr.jmax_c.th1=g_JMAX1;
  gl_dades_config.jr.jmax_c.th2=g_JMAX2;
  gl_dades_config.jr.jmax_c.th3=g_JMAX3;
  gl_dades_config.jr.jmax_c.th4=g_JMAX4;
  gl_dades_config.jr.jmax_c.th5=g_JMAX5;
  gl_dades_config.jr.jmax_c.th6=g_JMAX6;

  gl_dades_config.jr.jmin_c.th1=g_JMIN1;
  gl_dades_config.jr.jmin_c.th2=g_JMIN2;
  gl_dades_config.jr.jmin_c.th3=g_JMIN3;
  gl_dades_config.jr.jmin_c.th4=g_JMIN4;
  gl_dades_config.jr.jmin_c.th5=g_JMIN5;
  gl_dades_config.jr.jmin_c.th6=g_JMIN6;

  gl_dades_config.jr.jrng_c.th1=JRNG1;
  gl_dades_config.jr.jrng_c.th2=JRNG2;
  gl_dades_config.jr.jrng_c.th3=JRNG3;
  gl_dades_config.jr.jrng_c.th4=JRNG4;
  gl_dades_config.jr.jrng_c.th5=JRNG5;
  gl_dades_config.jr.jrng_c.th6=JRNG6;
```

```
  obtenir_transf_ctnt(&(gl_dades_config.T));

  printf("problema inicialitzat\n");
  escriure_text_a_scenee_tree("problema inicialitzat");
}
void
on_construir_button_clicked            (GtkButton       *button,
                                        gpointer         user_data)
{

  if (!(gl_borrar_graf))
    construccio_graf_config(&(gl_dades_config),&gl_G);
  else
    {
      destroy_graph(&gl_G);
      printf("\nesborrat graf anterior\n");
      escriure_text_a_scenee_tree("esborrat graf anterior.");
    }
}

void
on_extendre_button_clicked             (GtkButton       *button,
                                        gpointer         user_data)
{
  expansio_graf_config(&gl_dades_config,&gl_G);
}

void
on_carregar_map_button_clicked         (GtkButton       *button,
                                        gpointer         user_data)
{
  char fitxer_map[MAX_CADENA];
  char directori [MAX_CADENA]=DIRECTORI;

  sprintf(fitxer_map,"%s%s.map",directori,gl_nom_base);

  if (!empty_list(gl_G))
    destroy_graph(&gl_G);

  gl_G=carregar_mapa(fitxer_map);
}

void
on_desar_map_button_clicked            (GtkButton       *button,
                                        gpointer         user_data)
{
  char fitxer_map[MAX_CADENA];
  char fitxer_par_map[MAX_CADENA];
  char fitxer_wbt_map[MAX_CADENA];
  char directori[MAX_CADENA]=DIRECTORI;

  sprintf(fitxer_map,"%s%s.map",directori,gl_nom_base);
  sprintf(fitxer_par_map,"%s%s.par",directori,gl_nom_base);
  sprintf(fitxer_wbt_map,"%s%s.map.wbt",directori,gl_nom_base);

  desar_mapa(gl_G,fitxer_map);
  desar_mapa_en_wbt(gl_G,fitxer_wbt_map);
}

void
on_marcar_ini_button_clicked           (GtkButton       *button,
                                        gpointer         user_data)
{

  escriure_text_a_scenee_tree("configuracio inicial marcada");
  if (gl_config_ini!=NULL)
    alliberar_configuracio(&gl_config_ini);

  obtenir_configuracio_actual();
  avancar_simulacio();

  gl_config_ini=crear_configuracio();
  copiar_obtenir_config_a_configuracio(gl_config_ini);

  if (!strcmp(gl_dades_config.elem_aleat,"esquerra"))
    {
    sprintf(SOLUCE(gl_config_ini),"tot");
    sprintf(SOLUCD(gl_config_ini),gl_dades_config.conf_rob_dret);
    }
  if (!strcmp(gl_dades_config.elem_aleat,"dret"))
    {
    sprintf(SOLUCD(gl_config_ini),"tot");
    sprintf(SOLUCE(gl_config_ini),gl_dades_config.conf_rob_esq);
    }
}

void
on_marcar_fi_button_clicked            (GtkButton       *button,
                                        gpointer         user_data)
{
  escriure_text_a_scenee_tree("configuracio final marcada");
```

```
  if (gl_config_fi!=NULL)
    alliberar_configuracio(&gl_config_fi);

  obtenir_configuracio_actual();
  avancar_simulacio();

  gl_config_fi=crear_configuracio();
  copiar_obtenir_config_a_configuracio(gl_config_fi);


  if (!strcmp(gl_dades_config.elem_aleat,"esquerra"))
    {
      sprintf(SOLUCE(gl_config_fi),"tot");
      sprintf(SOLUCD(gl_config_fi),gl_dades_config.conf_rob_dret);
    }
  if (!strcmp(gl_dades_config.elem_aleat,"dret"))
    {
      sprintf(SOLUCD(gl_config_fi),"tot");
      sprintf(SOLUCE(gl_config_fi),gl_dades_config.conf_rob_esq);
    }
}

void
on_veure_ini_fi_button_clicked          (GtkButton       *button,
                                         gpointer         user_data)
{
  static int cont;

  if((gl_config_ini!=NULL)&&(gl_config_fi)!=NULL)
    {
      if (cont==0)
        {
          canviar_a_configuracio_predeterminada(gl_config_ini);
          avancar_simulacio();
          escriure_text_a_scenee_tree("configuracio_inicial");
          cont=cont+1;
        }
      else
        {
          canviar_a_configuracio_predeterminada(gl_config_fi);
          avancar_simulacio();
          escriure_text_a_scenee_tree("configuracio_final");
          cont=cont-1;
        }
    }
  else
    {
      printf("no_tenim_configuracions_inicial/final_per_veure\n");
      escriure_text_a_scenee_tree("no_tenim_configuracions_inicial/final_per_veure");
    }
}

void
on_carregar_infi_button_clicked         (GtkButton       *button,
                                         gpointer         user_data)
{
  char fitxer_config_ini_fi[MAX_CADENA];
  char directori[MAX_CADENA]=DIRECTORI;

  sprintf(fitxer_config_ini_fi,"%s%s%s.infi",directori,gl_nom_base,gl_extensio_infi);

  carregar_infi(fitxer_config_ini_fi,&gl_config_ini,&gl_config_fi);
}

void
on_desar_infi_button_clicked            (GtkButton       *button,
                                         gpointer         user_data)

{
  char fitxer_config_ini_fi[MAX_CADENA];
  char directori[MAX_CADENA]=DIRECTORI;

  sprintf(fitxer_config_ini_fi,"%s%s%s.infi",directori,gl_nom_base,gl_extensio_infi);
  desar_infi(gl_config_ini,gl_config_fi,fitxer_config_ini_fi);
}

void
on_fitxer_infi_entry_activate           (GtkEntry        *entry,
                                         gpointer         user_data)
{

  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  sprintf(gl_extensio_infi,"%s",(char*)text_entrat);
  printf("nom_arxiu:%s\n",gl_extensio_infi);
}

void
on_cercar_button_clicked                (GtkButton       *button,
                                         gpointer         user_data)
{
```

```
  interrogacio_graf_config(&gl_G, gl_config_ini, gl_config_fi,&gl_dades_config,&gl_traject);
}

void
on_suavitzar_button_clicked            (GtkButton       *button,
                                        gpointer         user_data)
{
  //suavitzacio_trajectoria(&gl_dades_config,&gl_traject);
  cercar_cami_minim_lliure(&gl_dades_config,&gl_traject);
}

void
on_carregar_trj_button_clicked         (GtkButton       *button,
                                        gpointer         user_data)
{
  char fitxer_trj[MAX_CADENA];
  char directori[MAX_CADENA]=DIRECTORI;

  sprintf(fitxer_trj,"%s%s%s%s.trj",directori,gl_nom_base,gl_extensio_infi,gl_extensio_trj);
  if (gl_traject!=NULL)
    destroy_list(&gl_traject,p_func_f);
  gl_traject=carregar_trajectoria(fitxer_trj);

}

void
on_desar_trj_button_clicked            (GtkButton       *button,
                                        gpointer         user_data)
{
  char fitxer_trj[MAX_CADENA];
  char fitxer_wbt_trj[MAX_CADENA];
  char directori[MAX_CADENA]=DIRECTORI;
  real distancia_total;

  distancia_total=calcular_distancia_total_traject(&gl_dades_config,gl_traject);
  sprintf(fitxer_trj,"%s%s%s%s.trj",directori,gl_nom_base,gl_extensio_infi,gl_extensio_trj);
  sprintf(fitxer_wbt_trj,"%s%s%s%s.trj.wbt",directori,gl_nom_base,gl_extensio_infi,gl_extensio_trj);

  desar_trajectoria(gl_traject,distancia_total,fitxer_trj);
  //desar_cami_en_wbt(gl_traject,fitxer_wbt_trj);
}

void
on_fitxer_trj_entry_activate           (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;

  text_entrat=gtk_entry_get_text (entry);
  sprintf(gl_extensio_trj,"%s",(char*)text_entrat);
  printf("nom_arxiu:%s\n",gl_extensio_trj);
}

void
on_visualitzar_button_clicked          (GtkButton       *button,
                                        gpointer         user_data)
{
  visualitzar_trajectoria (&gl_dades_config,gl_traject);
}

void
on_memoritzar_config_button_clicked    (GtkButton       *button,
                                        gpointer         user_data)
{
  bool conf_valida;
  char buffer[MAX_CADENA];


  inicialitzar_colisio();
  gl_config=crear_configuracio();
  conf_valida=canviar_a_configuracio_on_line(&gl_dades_config,gl_config);

  if (conf_valida)

    {
      // Configuracio no col.lisonant. Memoritzem.
      if(append(&gl_traject,(generic_ptr)gl_config)==ERROR)
        cerror("Error en memoritza_configuracio_actual()::config. valida no memoritzada");

      sprintf(buffer,"configuracio memoritzada com num. %d",(length(gl_traject)));
      escriure_text_a_scenee_tree(buffer);
    }
  else
    {
      // Configuracio col.lisionant. No memoritzem.
      escriure_text_a_scenee_tree("configuracio no memoritzada");
      alliberar_configuracio(&gl_config);
    }
}

void
on_borrar_config_button_clicked        (GtkButton       *button,
```

```
                                        gpointer         user_data)
{
  char buffer[MAX_CADENA];

  if (length(gl_traject)!=0)
    {
      if(delete_node(&gl_traject, last(gl_traject), p_func_f)==ERROR)
        cerror("Error en borra_config_previa_button_clickedl()");
      sprintf(buffer,"esborrada configuracio num. %d",(length(gl_traject)+1));
      escriure_text_a_scenee_tree(buffer);
    }
  else
    escriure_text_a_scenee_tree("la trajectoria es buida");
}

void
on_executar_button_clicked              (GtkButton       *button,
                                         gpointer         user_data)
{
  //Treball futur...
}

void p_func_f(generic_ptr data)
{
  CONFIG* config;
  config=(CONFIG*)(data);
  alliberar_configuracio(&config);
}

//Funcions asscociades a la finestra d'analisi del graf
void
on_vn_entry_activate                    (GtkEntry        *entry,
                                         gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  list pvertex=NULL;
  CONFIG* config;
  bool colisionen;

  text_entrat=gtk_entry_get_text (entry);
  gl_num_vertex=(int)atoi((char*)text_entrat);

  if ((gl_num_vertex<=0)||(gl_num_vertex>VERTEX_NUMBER(last(gl_G))))
    printf("Estem fora dels èvrtexs del graf\n");

  else
    {
      if((pvertex=find_vertex(gl_G,gl_num_vertex))!=NULL)
        {

          inicialitzar_colisio();
          config= CONFIG_VERTEX(pvertex);
          canviar_a_configuracio_predeterminada(config);
          avancar_simulacio();
          colisionen=obtenir_colisio();
          if (!colisionen)
            escriure_text_a_scenee_tree("configuracio lliure de col.lisio");
          else
            escriure_text_a_scenee_tree("configuracio amb col.lisio");
          printf("vertex:%d\n",gl_num_vertex);
        }
      else
        printf("El èvrtex s'ha eliminat\n");
    }
}

void
on_vertex_i_entry_activate              (GtkEntry        *entry,
                                         gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  text_entrat=gtk_entry_get_text (entry);

  gl_vertex_i=(vertex)atoi((char*)text_entrat);
  printf("El èvrtex ié s:%d\n",gl_vertex_i);
}


void
on_vertex_j_entry_activate              (GtkEntry        *entry,
                                         gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  text_entrat=gtk_entry_get_text (entry);

  gl_vertex_j=(vertex)atoi((char*)text_entrat);
  printf("vertex jé s:%d\n",gl_vertex_j);
}


void
on_n_components_entry_activate          (GtkEntry        *entry,
```

```
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  text_entrat=gtk_entry_get_text (entry);
  g_NUM_COMPONENTS=(int)atoi((char*)text_entrat);
  printf("El_nombre_màxim_de_componets_a_visualitzaré_s:%d\n",g_NUM_COMPONENTS);
}

void
on_min_analisi_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  text_entrat=gtk_entry_get_text (entry);
  gl_filtrar_min=(int)atoi((char*)text_entrat);
}


void
on_max_analisi_entry_activate          (GtkEntry        *entry,
                                        gpointer         user_data)
{
  G_CONST_RETURN gchar* text_entrat;
  text_entrat=gtk_entry_get_text (entry);
  gl_filtrar_max=(int)atoi((char*)text_entrat);
}


void
on_filtrar_button_clicked              (GtkButton        *button,
                                        gpointer         user_data)
{
  if (gl_filtrar_min==0)
    gl_filtrar_min=g_MIN_NODES;
  eliminar_components_graf_config(&gl_G,gl_filtrar_min,gl_filtrar_max);
}


void
on_n_components_button_clicked         (GtkButton        *button,
                                        gpointer         user_data)
{
  gl_taula_components=crear_taula_components();
  buscar_components_connexos_del_graf_config(gl_G,gl_taula_components);
}


void
on_distancia_analisi_button_clicked    (GtkButton        *button,
                                        gpointer         user_data)
{
  real distancia_total;
  char buffer[10];
  char buffer_fi[MAX_CADENA];

  if (!empty_list(gl_traject))
    {
      distancia_total=calcular_distancia_total_traject(&gl_dades_config,gl_traject);
      gcvt(distancia_total,4,buffer);
      printf("distancia_total:_%f\n",distancia_total);
      sprintf(buffer_fi,"La_òtrajectria_te_una_distancia_total_de:_%s",buffer);
      escriure_text_a_scenee_tree(buffer_fi);
    }
  else
    escriure_text_a_scenee_tree("no_existeix_cap_trajectoria_per_calcular_la_àdistncia");
}

void
on_cercar_analisi_button_clicked       (GtkButton        *button,
                                        gpointer         user_data)
{
  interrogacio_graf_config(&gl_G,gl_config_ini,gl_config_fi,&gl_dades_config,&gl_traject);
}
void
on_vertex_seg_button_clicked           (GtkButton        *button,
                                        gpointer         user_data)
{
  list pvertex=NULL;
  CONFIG* config;
  bool colisionen;


  gl_num_vertex=gl_num_vertex+1;

  if ((gl_num_vertex<=0)||(gl_num_vertex>VERTEX_NUMBER(last(gl_G))))
    printf("Estem_fora_dels_èvrtexs_del_graf\n");

  else
    {
      while((pvertex=find_vertex(gl_G,gl_num_vertex))==NULL)
        {
          if (gl_num_vertex>VERTEX_NUMBER(last(gl_G)))
```

```
            break;
          else
            gl_num_vertex=gl_num_vertex+1;
        }

      if (pvertex!=NULL)
        {
          inicialitzar_colisio();
          config= CONFIG_VERTEX(pvertex);
          canviar_a_configuracio_predeterminada(config);
          avancar_simulacio();
          colisionen=obtenir_colisio();
          if (!colisionen)
            escriure_text_a_scenee_tree("configuracio_lliure_de_col.lisio");
          else
            escriure_text_a_scenee_tree("configuracio_amb_col.lisio");
        }
      else
        printf("El_èvrtex_s'ha_eliminat\n");
      printf("La_óconfiguraci_actualé_s_del_èvrtex:%d\n",gl_num_vertex);
    }
}

void
on_vertex_ant_button_clicked            (GtkButton       *button,
                                         gpointer         user_data)
{
list pvertex=NULL;
  CONFIG* config;
  bool colisionen;

  gl_num_vertex=gl_num_vertex-1;


  if ((gl_num_vertex<=0)||(gl_num_vertex>VERTEX_NUMBER(last(gl_G))))
    printf("Estem_fora_dels_èvrtexs_del_graf\n");

  else
    {
      while((pvertex=find_vertex(gl_G,gl_num_vertex))==NULL)
        {
          if (gl_num_vertex<=0)
            break;
          else
            gl_num_vertex=gl_num_vertex-1;
        }
       if (pvertex!=NULL)
         {
           inicialitzar_colisio();
           config= CONFIG_VERTEX(pvertex);
           canviar_a_configuracio_predeterminada(config);
           avancar_simulacio();
           colisionen=obtenir_colisio();
           if (!colisionen)
             escriure_text_a_scenee_tree("configuracio_lliure_de_col.lisio");
           else
             escriure_text_a_scenee_tree("configuracio_amb_col.lisio");
           printf("vertex:%d\n",gl_num_vertex);
         }
       else
         printf("El_èvrtex_s'ha_eliminat\n");
    }
}

void set_value_entry_by_button (GtkButton       *button,
                                gpointer         user_data)
{
  char buffer[MAX_CADENA];
  gchar text[MAX_CADENA];
  GtkEntry* entry;

  entry=(GtkEntry*)user_data;
  gcvt(gl_num_vertex,6,buffer);
  sprintf(text,"%s",buffer);
  gtk_entry_set_text (entry,text);
}
```

H.11 graf_config.h

```
#ifndef GRAF_CONFIG_H
#define GRAF_CONFIG_H

/////////////////////////////////////////////////////
/// @file graf_config.h
/// @brief éCont les rutines que treballen amb un graf
/// de configuracions.
/// @author  Gorka Bonals
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       En aquest òmdul creem un graf de configuracions
```

```
///      amb el èmtode que anomenem mapa íprobabilstic.
///      Les principals funcions úpbliques constitueixen
///      les diferents fases de que consta el èmtode.
//////////////////////////////////////////////////////

#include "vei.h"
#include "vertexs_exp.h"
#include "component.h"
#include "globals.h"
#include "configuracio.h"
#include "traject.h"
#include "graf.h"
#include "wgraph.h"
#include "tau_list.h"

#define CONFIG_VERTEX(V) ((CONFIG*)(((vertex_data*)DATA(V))->specific_data))

//Operacions constructores.

//////////////////////////////////////////////////////
///  @brief Realitza l'etapa de óconstrucci.
///
///  Aquesta rutina realitza l'etapa de óconstrucci del
///  mapa íprobabilstic. Consisteix en la
/// ó generaci de configuracions lliures que s'emmagatzemen
///  com nodes d'un graf. Les arestes entre aquests nodes
///  indiquen que hi ha una òtrajectria lliure entre elles.
///
///  @param dades_config Par. ent. Dades constants per
///                      la ógeneraci i óconnexi de les
///                      configuracions.
///
///  @param p_G Par. ent/sort. Llista doblement çenllaada que
///                            representa el graf que creem.
//////////////////////////////////////////////////////

void construccio_graf_config(DADES_CONFIG* dades_config,graph* p_G);

//////////////////////////////////////////////////////
///  @brief Realiza la fase d'óexpansi.
///
///  Realitza l'etapa d'óexpansi del èmtode íprobabilstic,
///  que expandeix aquells nodes que
///  estan en regions de gran dificultat.
///
///  @param dades_config Par. ent. Dades constants per
///                      la ógeneraci i óconnexi de les
///                      configuracions.
///
///  @param p_G Par. ent/sort. Graf de configuracions que
///                            creem.
//////////////////////////////////////////////////////

void expansio_graf_config(DADES_CONFIG* dades_config,graph* p_G);

//////////////////////////////////////////////////////
///  @brief Filtra aquells components residuals.
///
///  Filtra del graf aquells components amb un
/// ú nmero de nodes que estan per sota i per
///  sobre d'un llindar. Quan els llindars ósn èidntics
///  elimina  els/el comonent que ét/tenen aquell únmero
///  de nodes.
///
///  @param p_G Par. ent/sort. Graf de configuracions.
///
///  @param filtrar_min Par. ent. Nombre enter pel llindar ímnim.
///
///  @param filtrar_max Par. ent. Nombre enter pel llindar àmxim.
//////////////////////////////////////////////////////

void eliminar_components_graf_config(graph* p_G,int filtrar_min,int filtrar_max);

//////////////////////////////////////////////////////
///  @brief Realitza la fase d'óinterrogaci.
///
/// í Aqu es realitza l'etapa d'óinterrogaci, la qual
///  connecta dues configuracions donades al graf.
///  Si existeix una üèseqncia de nodes que uneix
///  les configuracions donades, aquesta s'emmagatzema
///  en una llista (list).
///
///  @param p_G Par. ent. Graf de configuracions
///
///  @param config_ini Par. ent. La óconfiguraci inicial.
///
///  @param config_fi Par. ent. La óconfiguraci final.
///
///  @param dades_config Par. ent. Dades constants per
///                      la ógeneraci i óconnexi de les
///                      configuracions.
///
```

```
///  @param ptraject Par. ent/sort. éCont la òtrajectria
///                  com una üèseqncia de configuracions.
/////////////////////////////////////////////////////

void interrogacio_graf_config(graph* p_G,CONFIG* config_ini,
                              CONFIG* config_fi,DADES_CONFIG* dades_config,
                              list* ptraject);

/////////////////////////////////////////////////////
///  @brief Realitza la ósuavitzaci d'una trajectoria.
///
/// ó Suavitzaci de la òtrajectria pel èmtode de l'optimitzador
///  de la poligonal.
///
///  @param dades_config Par. ent. Dades constants per
///                      la ógeneraci i óconnexi de les configuracions.
///
///  @param p_traject Par. ent/sort. éCont la òtrajectria
///                   com una üèseqncia de configuracions.
/////////////////////////////////////////////////////

void suavitzacio_trajectoria(DADES_CONFIG* dades_config,list* p_traject);

/////////////////////////////////////////////////////
///  @brief Visualitzem una òtrajectria emmagatzemada.
///
///  Visualitza una òtrajectria com la óconcatenaci de
///  les òtrajectries entre nodes adjacents. Aquestes
/// ú ltimes es calculen çmitjanant el planificador local.
///
///  @param dades_config Par. ent. Dades associades a les
///                      configuracions.
///  @param traject Par. ent. éCont la òtrajectria
///                 com una üèseqncia de configuracions.
/////////////////////////////////////////////////////

void visualitzar_trajectoria (DADES_CONFIG* dades_config,list traject);

/////////////////////////////////////////////////////
///  @brief Cerca el ícam ímnim d'una òtrajectria.
///
///  Crea un graf de la òtrajectria, i d'aquest
///  busca el ícam ímnim entre el node inicial i final
///  de la òtrajectria. La òtrajectria ímnima resultant
/// à haur de ser lliure de col.lisions.
///
///  @param dades_config Par. ent. Dades constants per
///                      la ógeneraci i óconnexi de
///                      les configuracions.
///
///  @param ptraject Par. ent/sort. éCont la òtrajectria
///                  com una üèseqncia de configuracions.
/////////////////////////////////////////////////////

void cercar_cami_minim_lliure(DADES_CONFIG* dades_config, list* ptraject);

//Operacions consultores.

/////////////////////////////////////////////////////
///  @brief Busca el nombre de components connexos del
///         graf.
///
///  Busca el nombre de components connexos del graf.
///
///  @param G  Par. ent. Graf de configuracions.
///
///  @param taula_components Par. ent/sort. Taula que ét la
/// ó                 informaci que volem escriure.
///  @return Un enter queé s el nombre de components
///          connexos.
/////////////////////////////////////////////////////

int buscar_components_connexos_del_graf_config(graph G,COMPONENT* taula_components);

#endif
```

H.12 graf_config.c

```
//Per calcular el temps d'óexecuci de les fases.
#include <unistd.h>
#include <sys/times.h>

#include "graf_config.h"

/////////////////////////////////////////////////////
// Variables globals "locals"
/////////////////////////////////////////////////////

// Comptador de nombre de èvrtexs d'un component connex
// Utilitzat a p_action_vertex_before()
int gl_cont_vertex_connexos_local;   //comptador de nombre de èvrtexs
```

```
                                        //d'un component connex que utilitza
                                        //p_action_vertex_before()

/////////////////////////////////////////////////////
//Rutines privades
/////////////////////////////////////////////////////
static int connectar_vertex_c_a_graf_config(DADES_CONFIG* dades_config,
                                            vertex num_vertex_c,graph* p_G);
static bool cercar_veins_vertex_c_graf_config(DADES_CONFIG* dades_config,
                                              list pvertex_c,graph G,
                                              TAULA_VERTEX_VEINS* vertex_veins);
static bool cercar_vertex_veins_a_vertex_c_dist_burda(graph G,list pvertex_c,
                                                      TAULA_VERTEX_VEINS* vertex_veins);
static void cercar_vertexs_expand_graf_config(graph G,
                                              TAULA_VERTEX_EXP* vertexs_expand);
static void calcular_pesos_graf_config(graph* p_G);
static void eliminar_seguents_marca_graf_config(graph* p_G,
                                                list pvertex_marca);
static real distancia_entre_vertexs_graf_config(list pvertex_c,list pvertex_n);
static bool comparar_solucions_vertexs_graf_config(list pvertex_c,list pvertex_n);
static INFO_PLANIF_LOCAL* planificador_local_graf_config(bool detectar_colisio,
                                                         DADES_CONFIG* dades_config,
                                                         list pvertex_c,list pvertex_n);
static void incrementar_crides_i_fallades_entre_vertexs(bool colisionen,
                                                        list pvertex_c,
                                                        list pvertex_n);
static int cercar_primera_connexio_graf_config (DADES_CONFIG* dades_config,
                                                list pvertex_c,
                                                TAULA_VERTEX_VEINS* vertex_veins,
                                                graph G);
static void marcar_connexes_a_vertex_graf_config(list pvertex_n);
static bool connexio_vertex_c_vertex_n (list pvertex_c,list pvertex_n);
static void llista_vertexs_intermedis_entre_dos_graf_config(list pvertex_ini,
                                                            list pvertex_fi,
                                                            list* ptraject);
static status  p_action_vertex_before(list pvertex);
static void p_func_f1(generic_ptr data_pointer);
static void desar_graf_config_fitxer_depuracio(graph G);
static void desar_visitats_fitxer_depuracio_graf_config(graph G);
static real distancia_mitja_entre_obstacles(DADES_CONFIG* dades_config,graph* p_G);
void cercar_cami_minim_lliure(DADES_CONFIG* dades_config, list* ptraject);
static void crear_wgraph_de_traject(wgraph* g, DADES_CONFIG* dades_config,
                                    TAU_LIST* t);
static bool visualitzar_cami_minim(wgraph* g,DADES_CONFIG* dades_config,TAU_LIST* t);
static int trajectoria_aleatoria_de_salts_en_graf_config(char* nom_fase,
                                                         DADES_CONFIG* dades_config,
                                                         list pvertex_exp,graph* p_G);

extern CAMI cami;        //Guarda els nodes del ícam ímnim de wgraph.
int parent[MAXV];        //S'empra per obtenir el ícam ímnim de wgraph.

//-----------------------------------------------------
void construccio_graf_config(DADES_CONFIG* dades_config,graph* p_G)

{
  //Variables per calcular el temps de òcmput.
  struct tms t_init, t_end;
  double tm,ts;

  CONFIG* config;
  vertex vertex_c; í            //ndex del èvrtex a emmagatzemar.
  int connexions_vertex_c,connexions_totals,cont_comp_connexos;
  int num_vertexs=0;

  times(&t_init);

  num_vertexs=length(*p_G);

  //g_mem_set_vtable (glib_mem_profiler_table);

  //Per generar un mapa des del principi.
  if (empty_list(*p_G))
    {
      cont_comp_connexos=0;
      connexions_totals=0;

      vertex_c=1;

      printf("\nIniciant etapa de óconstrucci...\n");
      escriure_text_a_scenee_tree("Iniciant etapa de construccio...");

      //Obrim el fitxer de ódepuraci com escritura.
      if((g_fd=fopen("depuracio.res","w"))==NULL)
        printf("No puc obrir %s per escriure'l\n","depuracio.res");
    }
  else
    {
      //éNoms ét valides si el mapa no s'ha filtrat.
      cont_comp_connexos=buscar_components_connexos_del_graf_config(*p_G,NULL);
      connexions_totals=num_vertexs-cont_comp_connexos;

      //Comptem lú'ltim èvrtex i no la longitud del graf.
```

```
        vertex_c=VERTEX_NUMBER(last(*p_G))+1;

        printf("\nAmpliant etapa de óconstrucci..\n");
        escriure_text_a_scenee_tree("Ampliant etapa de construccio...");

        //Obrim novament el fitxer de ódepuraci.
        if((g_fd=fopen("depuracio.res","a"))==NULL)
                        printf("No puc obrir %s per escriure'l\n","depuracio.res");
      }
    while (vertex_c<=g_MAX_NODES)
      {
            //Cada nou èvrtex, necessita una óconfiguraci diferent.
        config=crear_configuracio();

        canviar_a_configuracio_lliure(TRUE,dades_config,config);

        //Guardem la óconfiguraci generada en un èvrtex del graf.
        if(add_vertex(p_G, vertex_c,(generic_ptr)config)==ERROR)
                        cerror("Error en on aprenentatge_button_clicked()");

        //Fem la óconnexi del èvrtex afegit a altres èvrtexs del graf.
        connexions_vertex_c=connectar_vertex_c_a_graf_config(dades_config,


        //Contem les connexions àvlides que s'han realitzat.
        connexions_totals=connexions_totals+connexions_vertex_c;


        //óInformaci de sortida
        printf("\n%% èvrtexs ïconstruts: %.2f\n",((real)vertex_c/(real)(g_MAX_NODES))*100.0);
        printf("n.èvrtexs generats: %d\n",vertex_c);
        printf("n.èvrtexs a generar: %d\n",g_MAX_NODES);
        printf("n.components connexos: %d\n",(num_vertexs-connexions_totals));

        //éNoms incrementem el contador quan hem emmagatzemat un èvrtex al graf.
        vertex_c=vertex_c+1;

        //Nombre de èvrtexs del graf. Si no s'ha filtrat el graf coincideix amb vertex_c.
        num_vertexs=num_vertexs+1;
      }

    //Calculem els pesos de tots els èvrtexs.
    calcular_pesos_graf_config(p_G);
    fclose(g_fd);

    times(&t_end);

    ts=(double)sysconf(_SC_CLK_TCK); //tics per second
    tm=(double)((t_end.tms_utime-t_init.tms_utime)+
                (t_end.tms_stime-t_init.tms_stime)+
                (t_end.tms_cutime-t_init.tms_cutime)+
                (t_end.tms_cstime-t_init.tms_cstime))/ts;

    printf("Temps d'óexecuci: %f\n",tm);
    printf("\nMapa ïconstrut\n");
    escriure_text_a_scenee_tree("Mapa construit");
}
//-------------------------------------------------------------
void expansio_graf_config(DADES_CONFIG* dades_config,graph* p_G)
{

  TAULA_VERTEX_EXP* vertexs_expand;
  int i,vertex_connexio,cont_connexions,cont_comp_connexos,cont_extensions;
  int cont_comp_connexos_incials;


  //Variables per calcular el temps de òcmput.
  struct tms t_init, t_end;
  double tm,ts;

  times(&t_init);

  //éNoms extenem si tenim un graf en òmemria.
  if (!empty_list(*p_G))
    {
      printf("\nIniciant etapa d'óextensi...\n");
      escriure_text_a_scenee_tree("Extenent mapa...");

      vertexs_expand=crear_taula_vertexs_expand();

      //Extenem els èvrtexs de la taula.
      cercar_vertexs_expand_graf_config(*p_G,vertexs_expand);
      escriure_info_vertexs_expand(vertexs_expand);

      //Comptem el nombre de components connexos del graf.
      cont_comp_connexos=buscar_components_connexos_del_graf_config(*p_G,NULL);
      cont_comp_connexos_incials=cont_comp_connexos;


      cont_extensions=0;
      cont_connexions=0;
      i=0;
```

```
      //Mentre no haguem explorat tots els èvrtexs a extendre fem.
      while ((i<(vertexs_expand->n))&&(cont_comp_connexos>1))
          {

            vertex_connexio=trajectoria_aleatoria_de_salts_en_graf_config("expansio",
                                                                          dades_config,
                                                                          (vertexs_expand->t)[i].pvertex,p_G);

            //En aquest cas hi ha óconnexi
            if (vertex_connexio!=-1)
              {
                cont_connexions=cont_connexions+1;
                cont_comp_connexos=cont_comp_connexos-1;
              }
            printf("\n%%_èvrtexs_extesos:_%.2f\n",
                ((real)(i+1)/(real)(vertexs_expand->n))*100.0);
            printf("%%_connexions_exitoses:_%.2f\n",
                ((real)(cont_connexions))/((real)(i+1))*100.0);
            printf("n.èvrtexs_extesos:_%d\n",(i+1));
            printf("n.èvrtexs_extesos_ambè_xit:_%d\n",cont_connexions);
            printf("n.èvrtexs_a_extendre:_%d\n",(vertexs_expand->n));
            printf("n.components_connexos_inicials:_%d\n",cont_comp_connexos_incials);
            printf("n.components_connexos:_%d\n",cont_comp_connexos);

            i=i+1;
          }

      times(&t_end);

      ts=(double)sysconf(_SC_CLK_TCK); //tics per second
      tm=(double)((t_end.tms_utime-t_init.tms_utime)+
                  (t_end.tms_stime-t_init.tms_stime)+
                  (t_end.tms_cutime-t_init.tms_cutime)+
                  (t_end.tms_cstime-t_init.tms_cstime))/ts;

      printf("Temps_d'óexecuci:_%f\n",tm);

      printf("Mapa_èexts\n");
      escriure_text_a_scenee_tree("Mapa_extes");
    }
  else
    {
      printf("No_tenim_cap_mapa_per_extendre\n");
      escriure_text_a_scenee_tree("No_tenim_cap_mapa_per_extendre");
    }
}
//-----------------------------------------------------------
void interrogacio_graf_config(graph* p_G,CONFIG* config_ini,
                              CONFIG* config_fi,DADES_CONFIG* dades_config,list* ptraject)
{
  vertex vertex_act;
  list node_act=NULL;
  list pvertex_ini=NULL;
  list pvertex_fi=NULL;
  list pvertex_marca=NULL;

  int vertex_connexio_ini;
   int vertex_connexio_fi;
  //Variables per calcular el temps de òcmput.
  struct tms t_init, t_end;
  double tm,ts;

  times(&t_init);


  //Cal disposar d'un mapa i les configuracions d'inici i final.
  if (empty_list(*p_G))
    {
      printf("\nNo_disposem_de_cap_mapa\n");
      escriure_text_a_scenee_tree("No_disposem_de_cap_mapa");
    }
  else if (config_ini==NULL)
    {
      printf("\ónConfiguraci_inicial_no_marcada\n");
      escriure_text_a_scenee_tree("Configuracio_inicial_no_marcada");
    }
  else if (config_fi==NULL)
    {
      printf("\ónConfiguraci_final_no_marcada\n");
      escriure_text_a_scenee_tree("Configuracio_final_no_marcada");
    }
  else
    {
      printf("\nIniciant_fase_d'óinterrogaci...");
      escriure_text_a_scenee_tree("Iniciant_fase_d'interrogacio..");


      //Borrem la llista on guardavem la òtrajectria anterior.
      if (!empty_list(*ptraject))
        {
          node_act=NULL;
```

```
    while ((node_act=list_iterator(*ptraject,node_act))!=NULL)
      DATA(node_act)=NULL;
    destroy_list(ptraject,NULL);
  }

//Fem una marca a lú'ltim èvrtex del graf.
pvertex_marca=last(*p_G);
vertex_act=VERTEX_NUMBER(pvertex_marca);


//Afegim la óconfiguraci inicial al final del mapa.
vertex_act=vertex_act+1;
if(add_vertex(p_G,vertex_act,(generic_ptr) config_ini)==ERROR)
  cerror("Error en interrogacio_graf_config()()1");
pvertex_ini=find_vertex(*p_G,vertex_act);

//Afegim la óconfiguraci final al final del mapa.
vertex_act=vertex_act+1;
if(add_vertex(p_G,vertex_act,(generic_ptr) config_fi)==ERROR)
  cerror("Error en interrogacio_graf_config()()2");
pvertex_fi=find_vertex(*p_G,vertex_act);

//Intentem connectar la óconfiguraci inicial al graf.
vertex_connexio_ini=trajectoria_aleatoria_de_salts_en_graf_config ("interrogacio",
                                                                   dades_config,
                                                                   pvertex_ini,p_G);

//Intentem connectar la óconfiguraci final al graf.
vertex_connexio_fi=trajectoria_aleatoria_de_salts_en_graf_config ("interrogacio",
                                                                  dades_config,
                                                                  pvertex_fi,p_G);

if ((vertex_connexio_fi!=-1)&&(vertex_connexio_ini!=-1))
  {
    printf("\nLa configuracio d'inici i final s'han connectat\n");

    //Borrem tots els sentits de les arestesé, s molt important fer-ho.
    clear_all_senses(*p_G);

    //Desmarquem i pentinem un altre cop el mapa.
    if (mark_all_unvisited(*p_G)==ERROR)
      cerror("Error en interrogacio_graf_config()3");
    if((mark_senses_down(*p_G,pvertex_ini,NULL))==ERROR)
      cerror("Error en interrogacio_graf_config()4");

    //Mirem si existeix un ícam entre ambdos èvrtexs.
    if (connexio_vertex_c_vertex_n (pvertex_ini,pvertex_fi))
      {
        if(init_list(ptraject)==ERROR)
          cerror("Error en interrogacio_graf_config()5");

        //Busquem la llista de èvrtexs que connecten el vertex inicial i el final.
        llista_vertexs_intermedis_entre_dos_graf_config(pvertex_ini,


        printf("Hi ha un ícam entre la óconfiguraci inicial i final\n");
        escriure_text_a_scenee_tree("Hi ha un cami entre la configuracio inicial i final");

        times(&t_end);

        ts=(double)sysconf(_SC_CLK_TCK); //tics per second
        tm=(double)((t_end.tms_utime-t_init.tms_utime)+
                    (t_end.tms_stime-t_init.tms_stime)+
                    (t_end.tms_cutime-t_init.tms_cutime)+
                    (t_end.tms_cstime-t_init.tms_cstime))/ts;

        printf("Temps d'óexecuci: %f\n",tm);
      }
    else
      {
        printf("Les configuracions no pertanyen al mateix component\n");
        escriure_text_a_scenee_tree("Les configuracions no pertanyen al mateix component");
      }
  }
else
  {
    if ((vertex_connexio_fi==-1)&&(vertex_connexio_ini==-1))
      {
        printf("\nLa óconfiguraci d'inici i final no s'han connectat\n");
        escriure_text_a_scenee_tree("La configuracio d'inici i final no s'han connectat");
      }
    else if(vertex_connexio_ini==-1)
      {
        printf("\nLa óconfiguraci d'inici no s'ha connectat\n");
        escriure_text_a_scenee_tree("La configuracio d'inici no s'ha connectat");
      }
    else if(vertex_connexio_fi==-1)
      {
        printf("\nLa óconfiguraci final no s'ha connectat\n");
        escriure_text_a_scenee_tree("La configuracio final no s'ha connectat");
      }
  }
```

```
      //Eliminar èvrtexs üsegents édesprs de marca.
      if ((pvertex_marca)!=NULL)
        {
          eliminar_seguents_marca_graf_config(p_G,pvertex_marca);
          pvertex_marca=NULL;
        }
    }
}
//-----------------------------------------------------
void suavitzacio_trajectoria(DADES_CONFIG* dades_config,list* p_traject)
{
  TAULA_TRAJECT_AUX* taula_traject;
  list pnode_borrar;
  INFO_PLANIF_LOCAL* planificador_local;
  int n,i,i_node_borrar;

  //Variables per calcular el temps de òcmput.
  struct tms t_init, t_end;
  double tm,ts;

  times(&t_init);


  if (empty_list(*p_traject))
    {
      printf("No tenim òtrajectria per suavitzar\n");
      escriure_text_a_scenee_tree("No tenim trajectoria per suavitzar");
    }
  else
    {
      printf("\nSuavitzant òtrajectria...\n");
      escriure_text_a_scenee_tree("Suavitzant trajectoria...");

      //Longitud de la òtrajectria.
      n=length(*p_traject);

      printf("Longitud del ícam inicial: %d\n", n);
      //Copio la òtrajectria en una taula auxiliar.
      taula_traject=copiar_traject_en_taula_traject_auxiliar(p_traject);

      //Calculo la taula de àdistncies.
      calcular_distancies_taula_traject_auxiliar(dades_config,taula_traject);


      //Ordenem la taula auxiliar segons àdistncia creixent.
      ordenar_taula_traject_auxiliar_segons_reduc_distancia(taula_traject);

      i=0;
      while(i<=((taula_traject->n)-3))
        {

          //éNoms borrem  nodes que escurcin la trajectoria
         // i no s'hagi fet la crida en el planificador local.
          if ((((taula_traject->t)[i].reduc_distancia)>0)&&
              ((taula_traject->t)[i].planif_local_cridat==FALSE))
            {
              planificador_local=planificador_local_configi_configf(TRUE,dades_config,
                                                                     (taula_traject->t)[i].config_ant,
                                                                     (taula_traject->t)[i].config_seg);

              //Si hem fet una crida al planificador per una ·cella, ho indiquem.
              (taula_traject->t)[i].planif_local_cridat=TRUE;
              if (planificador_local->solucio)
                {

                  //printf("Borrarem de la taula el node %d\n\n",(taula_traject->t)[i].num_node);

                  //Ens apuntem el punter de l'elem que s'ha de borrar.
                  pnode_borrar=(taula_traject->t)[i].pnode;

                  ordenar_taula_traject_auxiliar_segons_num_node(taula_traject);

                  //Busquem la posicio de la taula del pnode a borrar.
                  i_node_borrar=trobar_posicio_taula_donat_pnode(taula_traject,pnode_borrar);

                  //Desplacament dels elements una óposici enrera a partir de l'element que volem borrar.
                  desplacament_taula_traject_auxiliar(taula_traject,i_node_borrar);

                  //Al canviar les àdistncies es pot tornar a cridar el planificador
                  (taula_traject->t)[i_node_borrar].planif_local_cridat=FALSE;
                  (taula_traject->t)[(i_node_borrar-1)].planif_local_cridat=FALSE;

                  //éDesprs de fer el çdesplaament ja podem borrar el node de la taula.
                  if(delete_node(p_traject,pnode_borrar,p_func_f1)==ERROR)
                    cerror("Error aplicar_smoothing_a_traject()");

                  //S'han de recalcular les àdistncies.
                  calcular_distancies_taula_traject_auxiliar(dades_config,taula_traject);

                  ordenar_taula_traject_auxiliar_segons_reduc_distancia(taula_traject);
```

```
                    //Fem un recorregut de la taula des del principi.
                    i=-1;
                  }
                alliberar_info_planificador_local(&planificador_local);
              }
            i=i+1;
          }

        times(&t_end);

        ts=(double)sysconf(_SC_CLK_TCK); //tics per second
        tm=(double)((t_end.tms_utime-t_init.tms_utime)+
                    (t_end.tms_stime-t_init.tms_stime)+
                    (t_end.tms_cutime-t_init.tms_cutime)+
                    (t_end.tms_cstime-t_init.tms_cstime))/ts;

        printf("Temps d'óexecuci: %f\n",tm);

        printf("Longitud del ícam suavitzat: %d\n",length(*p_traject));
        escriure_text_a_scenee_tree("Trajectoria suavitzada");
      }
}
//-------------------------------------------------------------
void visualitzar_trajectoria (DADES_CONFIG* dades_config,list traject)

{
  CONFIG* config_act;
  CONFIG* config_seg;

  list pnode_act;
  list pnode_seg;
  char buffer[100];
  int cont=1;

  INFO_PLANIF_LOCAL* planificador_local;

  if (!empty_list(traject))

    {
      if (g_crear_ftraj==TRUE)
        if((g_ftraj=fopen("angles_traject.trj","w"))==NULL)
          printf("No puc obrir %s per escriure'l\n","angles_traject.trj");

      //Fem un recorregut del ícam de èvrtexs.
      pnode_act=traject;
      pnode_seg=NEXT(traject);

      //Mentre no estiguem al final de la llista de traject.
      while(pnode_seg!=NULL)
        {
          //De cada node recuperem la seva óconfiguraci
          config_act=(CONFIG*)DATA(pnode_act);
          config_seg=(CONFIG*)DATA(pnode_seg);

          if (g_crear_ftraj==TRUE)
            desar_angles_robots(g_ftraj,ROBOTE(config_act),ROBOTD(config_act));

          sprintf(buffer,"Realitzant trajectoria entre nodes: %d i %d",cont,(cont+1));
          escriure_text_a_scenee_tree(buffer);

          planificador_local=planificador_local_configi_configf(TRUE,dades_config,
                                                                 config_act,config_seg);

          if (g_crear_ftraj==TRUE)
            desar_angles_robots(g_ftraj,ROBOTE(config_seg),ROBOTD(config_seg));

          //éNoms si el planificador localé s certa continuem amb el recorregut.
          if (planificador_local->solucio)
            {

              pnode_act = pnode_seg;
              pnode_seg = NEXT(pnode_seg);

              cont=cont+1;
            }
          else
            {
              sprintf(buffer,"No existeix trajectoria entre nodes: %d i %d",cont,(cont+1));
              escriure_text_a_scenee_tree(buffer);
              break;
            }
          alliberar_info_planificador_local(&planificador_local);
        }
      //éNoms cal tancar el fitxer, si l'hem obert.
      if (g_crear_ftraj==TRUE)
        fclose(g_ftraj);
    }
  else
    {
      printf("La trajectoria esta buida\n");
      escriure_text_a_scenee_tree("La trajectoria esta buida");
    }
```

```
}
static void p_func_f1(generic_ptr data_pointer)
{
  CONFIG* config;

  config=(CONFIG*)data_pointer;
  alliberar_configuracio(&config);
}

////////////////////////////////////////////////////
///  @brief Realitza la connexio d'un node c amb altres.
///
///  Intenta connectar el node c amb altres
//   nodes del graf de configuracions. La óconnexi es fa de
//   manera que no es generin cicles en el graf, seguint les
//   pautes del metode probabilistic
///
///  @param dades_config Par. ent. Dades constants per
///                      la ógeneraci i óconnexi de les
///                      configuracions.
///  @param num_vertex_c Par. ent. Index del node a insertar.
///
///  @param p_G Par. sort. Llista doblement çenllaada que
///                        representa el graf que crearem.
///  @return Un enter que indica el nombre de connexions ok.
////////////////////////////////////////////////////


static int connectar_vertex_c_a_graf_config(DADES_CONFIG* dades_config,
                                            vertex num_vertex_c,graph* p_G)


{
  TAULA_VERTEX_VEINS* vertex_veins;
  list pvertex_c;
  bool hi_ha_veins;
  bool colisionen;
  INFO_PLANIF_LOCAL* planificador_local;
  int index_vei;
  int connexions_vertex_c=0;

  TRACE1((g_fd,"El èvrtex c inserté s el %d.\n",num_vertex_c));

  //Marco tots els èvrtexs com a no visitats.
  if (mark_all_unvisited(*p_G)==ERROR)
    cerror("Error en connectar_vertex_c_a_graf_config()");

  //Creo una taula que àcontindr els èvrtexs veins a c
  vertex_veins=crear_taula_vertex_veins();

  pvertex_c=find_vertex(*p_G,num_vertex_c);
  if (pvertex_c==NULL)
    cerror("Error en  connectar_vertex_c_a_graf_config()");


  //Busquem els g_NVEINS éms propers al vertex c del graf.
  hi_ha_veins=cercar_veins_vertex_c_graf_config(dades_config,
                                                pvertex_c,*p_G,vertex_veins);

  //Desem la taula de veins al vertex c(ódepuraci completa).
  desar_info_vertex_veins_a_fitxer_depuracio(num_vertex_c,vertex_veins);

  index_vei=0;

  if (hi_ha_veins)
    {
      //Es busca el primer vei on el planificador local ha tingutè xit.
      index_vei=cercar_primera_connexio_graf_config (dades_config,
                                                     pvertex_c,vertex_veins,*p_G);


      //S'ha de comprovar que l'index_vei trobat, correspon a un íve àvlid de la taula.
      if (index_vei<vertex_veins->n)
        {
          TRACE1((g_fd,"La primera óconnexi ha estat entre el vertex %d i el èvrtex vei %d.\n",
                                num_vertex_c,
                  (vertex_veins->t)[index_vei].num_vertex));


          if (add_edge(*p_G,num_vertex_c,(vertex_veins->t)[index_vei].num_vertex,NULL)==ERROR)
            cerror("Error en connectar_vertex_c_a_graf_config()");

          //Marquem els èvrtexs que ósn connexes al vertex vei que hem connectat.
          marcar_connexes_a_vertex_graf_config((vertex_veins->t)[index_vei].pvertex);

          //Desem els èvrtexs que hem visitat a l'aplicar l'anterior ófunci(ódepuraci completa).
          TRACE2((g_fd,"Marcar èvrtexs connexes al vertex vei %d.\n",(vertex_veins->t)[index_vei].num_vertex));
          desar_visitats_fitxer_depuracio_graf_config(*p_G);

          //éNoms si hem connectat algun vertex íve el c ét sentit
          //analitzar si podem connectar els üsegents èvrtexs ïvens.
          index_vei=index_vei+1;
```

```
          connexions_vertex_c=connexions_vertex_c+1;

        }
      else
        TRACE1((g_fd,"No hem trobat cap crida entre el planificador local i els ïvens amb èxit.\n"));


      //Intentar connectar la resta de èvrtexs ïvens al èvrtex c sense crear cicles en el graf.
      while (index_vei<vertex_veins->n)
        {
          if (!(connexio_vertex_c_vertex_n(pvertex_c,(vertex_veins->t)[index_vei].pvertex)))

            {
              //Desem entre quins èvrtexs establim la crida al planificador local(ódepuraci completa).
              TRACE2((g_fd,"Crida del planificador local entre el èvrtex %d i el èvrtex ïve %d\n",
                      VERTEX_NUMBER(pvertex_c),VERTEX_NUMBER((vertex_veins->t)[index_vei].pvertex)));


              planificador_local=planificador_local_graf_config(TRUE,
                                                                 dades_config,pvertex_c,
                                                                 (vertex_veins->t)[index_vei].pvertex);

              //óActualitzaci dels camps de les cirdes i fracassos locals dels nodes intervinguts.
              //de les crides i els fracassos del planificador local dels èvrtexs que han intervingut.
              colisionen=obtenir_colisio();
              incrementar_crides_i_fallades_entre_vertexs(colisionen,pvertex_c,
                                                          (vertex_veins->t)[index_vei].pvertex);
              if (planificador_local->solucio)
                {
                  //Desem la óconnexi que s'ha establert entre el vertex c i un dels ïvens.
                  TRACE1((g_fd,"Hi ha connexio entre el vertex %d i el èvrtex vei %d.\n",num_vertex_c,
                          (vertex_veins->t)[index_vei].num_vertex));

                  //Marquem els èvrtexs que ósn connexes al èvrtex que hem afegit. Ho fem abans d'afegir
                  //l'aresta, ja que íaix ens estalviem de tornar a marcar èvrtexs que ja ho estaven.
                  marcar_connexes_a_vertex_graf_config((vertex_veins->t)[index_vei].pvertex);

                  //Desem en el fitxer de ódepuraci els èvrtexs que s'han visitat del graf
                  //a l'aplicar l'anterior ófunci.
                  desar_visitats_fitxer_depuracio_graf_config(*p_G);

                  if (add_edge(*p_G,num_vertex_c,(vertex_veins->t)[index_vei].num_vertex,NULL)==ERROR)
                    cerror("Error error en connectar aresta en construccio_graf()");

                  connexions_vertex_c=connexions_vertex_c+1;
                }

              alliberar_info_planificador_local(&planificador_local);
            }
          index_vei=index_vei+1;
        }
    }
  //Desem la órelaci entre arestes i èvrtexs, édesprs d'haver ïintrodut un èvrtex c.
  TRACE1((g_fd,"\nGraf de èvrtexs i arestes al finalitzar l'etapa de óconnexi del èvrtex %d amb èvrtex ïvens:\n",
          num_vertex_c));
  desar_graf_config_fitxer_depuracio(*p_G);
  TRACE1((g_fd,"-------------------------------------------------------------------------------\n\n"));
  return connexions_vertex_c;
}

/////////////////////////////////////////////////////////////
/// cercar_veins_vertex_c_graf_config()
///
/// En aquesta rutina s'omple una taula de veins, amb els nodes
/// veins g_NVEINS que tenen menys àdistncia respecte al node c
/// inserit. éDesprs s'ordena la taula  per àdistncia creixent
/// dels nodes ïvens al node c.
///
/// @param dades_config Dades constants associades a la generacio
///                     i connexio de configuracions.
///
///
/// @param  pvertex_c: El node c inserit.
///
///
/// @param G  Estructura que écont el graf de configuracions.èvrtex_veins:
///
/// @param vertex_veins Estructura que te els veins ordenats segons
///        distancia creixent al node c.
///
/// Retorna:
///    Un boolea que indica sihem trobat o no nodes ïvens.
/////////////////////////////////////////////////////////////

static bool cercar_veins_vertex_c_graf_config(DADES_CONFIG* dades_config,list pvertex_c,
                                              graph G,TAULA_VERTEX_VEINS* vertex_veins)
{
  list pvertex_act=NULL;
  real distancia;
  bool igual_tipus_config;
  bool hi_ha_veins;
  int i,imax=0;
```

```
  //En principi inicialitzem que no hi ha veins.
  hi_ha_veins=FALSE;
  //Inicialitzo a un valor molt gran el camp aresta de la taula de veins.
  inicialitzar_vertex_veins(vertex_veins);

  //Fem un recorregut per tot el graf.
  while((pvertex_act=list_iterator(G,pvertex_act))!=NULL)
    {
      //éNoms poden ser veins els que tinguin mateixa ósoluci àcinemtica.
      igual_tipus_config=comparar_solucions_vertexs_graf_config(pvertex_c,pvertex_act);


      distancia=distancia_entre_vertexs_graf_config(pvertex_c,pvertex_act);


      if  ((distancia<g_MAX_DIST)&&(igual_tipus_config)&&(!(VISITED(pvertex_act))))
        {

          imax=index_distancia_maxima(vertex_veins);

          //ïSubstitum el nou èvrtex vei pel de àdistncia major.
          if (((vertex_veins->t)[imax].aresta)>distancia)
            {
              (vertex_veins->t)[imax].aresta=distancia;
              (vertex_veins->t)[imax].pvertex=pvertex_act;
              (vertex_veins->t)[imax].num_vertex=VERTEX_NUMBER(pvertex_act);
            }
        }
    }
  //Ordenacio de la taula de nodes ïvens pel camp aresta(àdistncia) creixent.
  qsort(vertex_veins->t,g_NVEINS,sizeof(INFO_VEI),compare);

  //Recalculem les àdistncies de la taula, amb la àdistncia fina.
  i=0;
  while ((i<g_NVEINS)&&((vertex_veins->t)[i].aresta!=BIG))
    {
      distancia=distancia_fina_entre_configuracions(dades_config,
                                                    CONFIG_VERTEX(pvertex_c),
                                                    CONFIG_VERTEX((vertex_veins->t)[i].pvertex));
      if (distancia<g_MAX_DIST)
        {
          (vertex_veins->t)[i].aresta=distancia;

          //éNoms que trobem un íve, ja significa que hi ha veins.
          hi_ha_veins=TRUE;
        }
      else
        {
          (vertex_veins->t)[i].aresta=BIG;
          (vertex_veins->t)[i].pvertex=NULL;
          (vertex_veins->t)[i].num_vertex=BIG;
        }
      i=i+1;
    }

  qsort(vertex_veins->t,g_NVEINS,sizeof(INFO_VEI),compare);

  //El nombre de veins omplerts ódna lloc a la longitud de la taula.
  i=0;
  while (i<g_NVEINS)
    {
      if((vertex_veins->t)[i].aresta==BIG)
        break;
      i=i+1;
    }
  vertex_veins->n=i;


  return hi_ha_veins;
}
static bool cercar_vertex_veins_a_vertex_c_dist_burda(graph G,list pvertex_c,
                                                      TAULA_VERTEX_VEINS* vertex_veins)
{
  list pvertex_act=NULL;
  real distancia;
  bool igual_tipus_config;
  bool hi_ha_veins;
  int i,imax=0;

  //En principi inicialitzem que no hi ha veins.
  hi_ha_veins=FALSE;
  //Inicialitzo a un valor molt gran el camp aresta de la taula de veins.
  inicialitzar_vertex_veins(vertex_veins);

  //Fem un recorregut per tot el graf.
  while((pvertex_act=list_iterator(G,pvertex_act))!=NULL)
    {
      //éNoms poden ser veins els que tinguin mateixa ósoluci àcinemtica.
      igual_tipus_config=comparar_solucions_vertexs_graf_config(pvertex_c,pvertex_act);
```

```
      distancia=distancia_entre_vertexs_graf_config(pvertex_c,pvertex_act);

      if  ((distancia<g_MAX_DIST)&&(igual_tipus_config)&&(!(VISITED(pvertex_act))))
        {
          imax=index_distancia_maxima(vertex_veins);

          //ïSubstitum el nou èvrtex vei pel de àdistncia major.
          if (((vertex_veins->t)[imax].aresta)>distancia)
            {
              (vertex_veins->t)[imax].aresta=distancia;
              (vertex_veins->t)[imax].pvertex=pvertex_act;
              (vertex_veins->t)[imax].num_vertex=VERTEX_NUMBER(pvertex_act);

              //éNoms que trobem un íve, ja significa que hi ha veins.
              hi_ha_veins=TRUE;
            }
        }
    }


  //Ordenacio de la taula de èvrtexs ïvens pel camp aresta(àdistncia) creixent.
  qsort(vertex_veins->t,g_NVEINS,sizeof(INFO_VEI),compare);

  //El nombre de veins omplerts ódna lloc a la longitud de la taula.
  i=0;
  while (i<g_NVEINS)
    {
      if((vertex_veins->t)[i].aresta==BIG)
        break;
      i=i+1;
    }
  vertex_veins->n=i;


  return hi_ha_veins;
}

////////////////////////////////////////////////////////////////
// calcular_pesos_graf_config()
//
// Donat un graf de configuracions, calcula els pesos associats
// a cada èvrtex seguint el criteri de ratis descrit a la òmemria
// a la pag X. Els pesos éms alts indiquen  que els èvrtexs es
// troben en regions d'una gran dificultat.
//
// Arguments:
//    p_G:
//       Es el punter a l'estructura que écont el
//       graf de configuracions.
// Retorna:
//       void.
//
////////////////////////////////////////////////////////////////

static void calcular_pesos_graf_config(graph* p_G)
{
  list pvertex_act;
  CONFIG* config;
  real rati_act;
  real suma_ratis=0.0;

  //Recorregut per calcular els ratis de cada èvrtex.
  pvertex_act=NULL;
  while((pvertex_act=list_iterator(*p_G, pvertex_act))!=NULL)
    {
      config=CONFIG_VERTEX(pvertex_act);
      rati_act=(((real)F_PLAN_LOC(config))/(((real)N_PLAN_LOC(config))+1));
      PES(config)=rati_act;
      suma_ratis=suma_ratis+rati_act;
    }
  //Recorregut per normalitzar els ratis dels èvrtexs.
  pvertex_act=NULL;
  while((pvertex_act=list_iterator(*p_G, pvertex_act))!=NULL)
    {
      config=CONFIG_VERTEX(pvertex_act);
      PES(config)=((PES(config))/suma_ratis);
    }
}

////////////////////////////////////////////////////////////////
// cercar_vertexs_expand_graf_config()
//
// Calcula els (int)(((real)g_PERC_EXP*0.01)*g_MAX_NODES)
// de èvrtexs, amb els pesos éms grans de tots els èvrtexs del
// graf de configuracions.
//
// Arguments:
//    G:
//       Es l'estructura que écont el
//       graf de configuracions.
// Retorna:
//   vertexs_expand:
```

```
//       Es una taula que écont els èvrtexs que volem expandir en la
//       fase d'óexpansi.
//       void.
//
////////////////////////////////////////////////////////////////

static void cercar_vertexs_expand_graf_config(graph G,
                                              TAULA_VERTEX_EXP* vertexs_expand)
{
  list pvertex_act=NULL;
  CONFIG* config;
  real pes;
  int i,imin=0;
  int num_vertexs_expand;

  num_vertexs_expand=vertexs_expand->n;

   //Fem un recorregut per tots els èvrtexs del graf.
   while((pvertex_act=list_iterator(G,pvertex_act))!=NULL)
    {
      config=CONFIG_VERTEX(pvertex_act);
      pes=PES(config);

      imin=index_pes_minim(vertexs_expand);

      //ïSubstitum el pes éms petit de la taula pel nou.
      if ((pes!=0)&&(((vertexs_expand->t)[imin].pes)<pes))
        {
          (vertexs_expand->t)[imin].pes=pes;
          (vertexs_expand->t)[imin].pvertex=pvertex_act;
        }
    }
  //Ordenacio de la taula de veins per aresta creixent.
  qsort(vertexs_expand->t,num_vertexs_expand,sizeof(VERTEX_EXP),compare_pesos);

  //Busquem la longitud de la taula.
  i=0;
  while (i<num_vertexs_expand)
    {
      if((vertexs_expand->t)[i].pvertex==NULL)
        break;
      i=i+1;
    }
  (vertexs_expand->n)=i;
}

////////////////////////////////////////////////////////////////
// distancia_entre_vertexs_graf_config()
//
// Rutina que calcula la distancia entre dos èvrtexs del graf.
//
// Arguments:
//     pvertex_c:
//       Es un punter a un èvrtex c.
//     pvertex_n:
//       Es un punter a un èvrtex n.
// Retorna:
//     Un tipus real, ié s la àdistncia entre els dos èvrtexs.
//     Si els dos èvrtexs ósn iguals, o algun d'ellsé s NULL
//     la àdistncia retorna BIG.
//
////////////////////////////////////////////////////////////////

static real distancia_entre_vertexs_graf_config(list pvertex_c,list pvertex_n)

{
  CONFIG* config_c;
  CONFIG* config_n;

  //Si un dels dos punters als vertexs, es inutil trobar la distancia
  if ((pvertex_c==NULL)||(pvertex_n==NULL))
    return BIG;
  else
    {
      //Adquirim la informacio de les configuracions emmagatzemades en cada vertex.
      config_c= CONFIG_VERTEX(pvertex_c);
      config_n= CONFIG_VERTEX(pvertex_n);

      //Calculem la distancia entre les dues configuracions.
      return (distancia_entre_configuracions(config_c,config_n));
    }
}

////////////////////////////////////////////////////////////////
//  comparar_solucions_vertexs_graf_config()
//
// Rutina que donats dos èvrtexs del graf mira si les
// solucions d'ambdues configuracions associades a cada
// èvrtex, pertanyen al mateix tipus de ósoluci àcinemtica.
//
// Arguments:
//     pvertex_c:
```

```
//      Es un punter a un èvrtex c.
//    pvertex_n:
//      Es un punter a un èvrtex n.
// Retorna:
//    Un tipus àboole que indica si els èvrtexs ósn o no de
//    del mateix tipus de ósoluci.
//
////////////////////////////////////////////////////////////////

static bool comparar_solucions_vertexs_graf_config(list pvertex_c,
                                                   list pvertex_n)

{
  CONFIG* config_c;
  CONFIG* config_n;

  if ((pvertex_c==NULL)||(pvertex_n==NULL))
    return FALSE;
  else
    {
      config_c= CONFIG_VERTEX(pvertex_c);
      config_n= CONFIG_VERTEX(pvertex_n);
      return(comparar_tipus_solucions_configuracions(config_c,config_n));
    }
}

////////////////////////////////////////////////////////////////
// planificador_local_graf_config()
//
// Rutina que fa una crida del planificador local de les
// configuracions contingudes entre dos èvrtexs donats.
//
// Arguments:
//    pvertex_c:
//      Es un punter a un èvrtex c.
//    pvertex_n:
//      Es un punter a un èvrtex n.
// Retorna:
//    Un tipus àboole que indica si la crida ha
//    tingut o noè xit.
//
////////////////////////////////////////////////////////////////

static INFO_PLANIF_LOCAL* planificador_local_graf_config(bool detectar_colisio,
                                                         DADES_CONFIG* dades_config,
                                                         list pvertex_c,
                                                         list pvertex_n)
{
  CONFIG* config_c;
  CONFIG* config_n;
  INFO_PLANIF_LOCAL* planificador_local;


  if ((pvertex_c==NULL)||(pvertex_n==NULL))
    cerror("Error en planificador_local_graf_config()");
  else
    {
      config_c= CONFIG_VERTEX(pvertex_c);
      config_n= CONFIG_VERTEX(pvertex_n);

      planificador_local=planificador_local_configi_configf(detectar_colisio,
                                                             dades_config,config_c,
                                                             config_n);
    }
  return planificador_local;
}

////////////////////////////////////////////////////////////////
// incrementar_crides_i_fallades_entre_vertexs()
//
// Rutina que incrementa el únmero de crides del planificador local
// que s'estableix en un èvrtex, íaix com el nombre de fallades.
//
// Arguments:
//   colisionen:
//     Variable que indica si la ultima crida ha fracassat per
// ·ó       collisi o no.
//   pvertex_c:
//     Es el punter al vertex c.
//   pvertex_n:
// é    Cont la taula dels èvrtexs ïvens al èvrtex c.
// Retorna:
//    void.
//
////////////////////////////////////////////////////////////////

static void incrementar_crides_i_fallades_entre_vertexs(bool colisionen,
                                                        list pvertex_c,
                                                        list pvertex_n)
{
  CONFIG* config_c;
  CONFIG* config_n;
```

```
  config_c= CONFIG_VERTEX(pvertex_c);
  config_n= CONFIG_VERTEX(pvertex_n);

  //Incremento les crides entre la configuracio c i n.
  N_PLAN_LOC(config_c)=N_PLAN_LOC(config_c)+1;
  N_PLAN_LOC(config_n)=N_PLAN_LOC(config_n)+1;

  //Les fallades énoms les comptem per ócollisi.
  if (colisionen)
    {
      F_PLAN_LOC(config_c)=F_PLAN_LOC(config_c)+1;
      F_PLAN_LOC(config_n)=F_PLAN_LOC(config_n)+1;
    }
}

////////////////////////////////////////////////////////////
// cercar_primera_connexio_graf_config()
//
// Rutina que retorna lí'ndex del primer íve del conjunt
// de èvrtexs ïvens al vertex c,on la crida del planificador
// local entre el èvrtex c i aquest vei ha estat exitosa.
//
// Arguments:
//   elem_interp,pdh,jr,T,conf_rob_esq,conf_rob_dret:
// ó    Sn variables, que ósn constants per tota óconfiguraci
//      i que les necessitem per generar-les.
//   G:
//     Es l'estructura que écont el graf de configuracions.
//   pvertex_c:
//     Es el punter al vertex c.
//   vertex_veins
// é    Cont la taula dels èvrtexs ïvens al èvrtex c.
// Retorna:
//    void.
//
////////////////////////////////////////////////////////////

static int cercar_primera_connexio_graf_config (DADES_CONFIG* dades_config,
                                                list pvertex_c,
                                                TAULA_VERTEX_VEINS* vertex_veins,graph G)

{
  INFO_PLANIF_LOCAL* planificador_local;
  bool colisionen;

  //El primer index es zero, ja que hem de recorrer una taula.
  int index_vei=0;

  //Fins no arribar a lú'ltim vei.
  while (index_vei<vertex_veins->n)
    {
      //Fem el planificador local entre el èvrtex c i un èvrtex íve.
      planificador_local=planificador_local_graf_config(TRUE,dades_config,pvertex_c,
                                                         (vertex_veins->t)[index_vei].pvertex);

      //Actualitzem el contador de crides i fallades per cada óconfiguraci.
      colisionen=obtenir_colisio();
      incrementar_crides_i_fallades_entre_vertexs(colisionen,pvertex_c,
                                                  (vertex_veins->t)[index_vei].pvertex);

      //Si el planificador étè xit sortim.
      if (planificador_local->solucio)
        {
          alliberar_info_planificador_local(&planificador_local);
          break;
        }
      alliberar_info_planificador_local(&planificador_local);
      index_vei=index_vei+1;
    }
  return index_vei;
}

////////////////////////////////////////////////////////////
// marcar_connexes_a_vertex_graf_config()
//
// Simplement fem una crida de la funcio de cerca de
// profunditat amb els parametres desitjats.
//
// Arguments:
//   pvertex_n:
//     Es el punter al vertex n.
// Retorna:
//    void.
//
////////////////////////////////////////////////////////////

static void marcar_connexes_a_vertex_graf_config(list pvertex_n)
{

  //Cridem una cerca amb profunditat amb àparmetres adients.
  if (depth_first_vertices_edges(pvertex_n,pvertex_n,NULL,NULL,NULL,NULL)==ERROR)
```

```
    cerror("Error en marcar connexes a vertex graf config()");
}

////////////////////////////////////////////////////////////////
// connexio_vertex_c_vertex_n()
//
// Rutina que mira si existeix un ícam del graf entre els
// pvertex_c i el pvertex_n. Abans d'aquesta ófunci
// S'ha de fer una cerca amb profunditat partint d'un d'aquest
// èvrtexs.
//
// Arguments:
//   pvertex_c, pvertex_n :
// ó   Sn dos punters a dos èvrtexs donats.
// Retorna:
//    Un boolea que indica si existeix óconnexi o no entre
//    els nodes donats.
//
////////////////////////////////////////////////////////////////

static bool connexio_vertex_c_vertex_n (list pvertex_c,list pvertex_n)
{
  //Retorna cert si s'ha visitat el pvertex_n.
  return VISITED(pvertex_n);
}

////////////////////////////////////////////////////////////////
// escriure_vertexs_visitats_de_graf_config()
//
// Rutina que escriu els èvrtexs que tenen el camp
// visitat com a cert.
//
// Arguments:
//   G: El graf que écont els èvrtexs.
// Retorna:
//    void.
//
////////////////////////////////////////////////////////////////

void escriure_vertexs_visitats_de_graf_config(graph G)
{
  list pvertex_act=NULL;
  vertex num_vertex;

  //Fem un recorregut per tots els èvrtexs del graf.
  while((pvertex_act=list_iterator(G, pvertex_act))!=NULL)
    {
      if (VISITED(pvertex_act)==TRUE)
        {
          num_vertex=VERTEX_NUMBER(pvertex_act);
          printf("El èvrtex %d s'ha visitat\n",num_vertex);
        }
    }
}

////////////////////////////////////////////////////////////////
// llista_vertexs_intermedis_entre_dos_graf_config()
//
// Rutina recursiva que guarda en una llista els punters
// associats a cada un dels èvrtexs que configuren un ícam que
// uneix dos èvrtexs donats.
// Per ser aplicada, èprviament cal que s'hagi fet una crida
// de mark_sense_down() amb èvrtex inicial pvertex_ini.
// La llista ptraject que trobem al final, ét configuracions
// que no ósn punters al graf, i per tan podem borrar nodes
// sense cap problema.
//
// Arguments:
//   pvertex_ini
//     Punter al èvrtex inicial.
//   pvertex_act:
//     Punter al èvrtex final a la primera óiteraci. éDesprs
//     va esdevint el pvertex que va recorrent el ícam entre
//     el pvertex inicial i el final.
//   Retorna:
//      void.
//
////////////////////////////////////////////////////////////////

static void llista_vertexs_intermedis_entre_dos_graf_config(list pvertex_ini,
                                                            list pvertex_act,
                                                            list* ptraject)

{
  list pvertex_seg;
  list paresta_act=NULL;
  CONFIG* config;
  CONFIG* nconfig;

  //Quan la llista ptraject esta buida insertem el pvertex_act a la llista.
  if (*ptraject==NULL)
    {
```

```
        config=CONFIG_VERTEX(pvertex_act);

        printf("vertex:%d\n",VERTEX_NUMBER(pvertex_act));

        nconfig=crear_configuracio();

        //Copiem la óconfiguraci del èvrtex del graf, en un node d'una llista.
        copiar_config_a_nova_config(config,nconfig);

        if ((append(ptraject,(generic_ptr)(nconfig)))==ERROR)
          cerror("Error en llista_vertexs_intermitjos()");
      }

    //Busquem la primera aresta que surt d'un vertex donat que te sentit UP.
    while ((SENSE(paresta_act=list_iterator(EMANATING(pvertex_act),paresta_act)))!=UP);

    //Mirem el vertex amb que conecta l'aresta amb sentit UP.
    pvertex_seg=VERTEX_POINTER(paresta_act);

    //Encara no hem arribat al principi del ícam.
    if (pvertex_seg!=pvertex_ini)
      {
        //Insertem el pvertex_seg abans de l'ultim vertex introduit.
        config=CONFIG_VERTEX(pvertex_seg);

        printf("vertex:%d\n",VERTEX_NUMBER(pvertex_seg));

        nconfig=crear_configuracio();
        copiar_config_a_nova_config(config,nconfig);

        if((insert(ptraject,(generic_ptr)(nconfig)))==ERROR)
          cerror("Error_add_vertex()_failed_in_build_K4()");

        llista_vertexs_intermedis_entre_dos_graf_config(pvertex_ini,
                                                        pvertex_seg,ptraject);
      }
    //Si el seguent pvertexé s justament l'inicial, hem acabat.
    else
      {
        config=CONFIG_VERTEX(pvertex_seg);

        printf("vertex:%d\n",VERTEX_NUMBER(pvertex_seg));

        nconfig=crear_configuracio();
        copiar_config_a_nova_config(config,nconfig);

        if ((insert(ptraject,(generic_ptr)(nconfig)))==ERROR)
          cerror("Error_en_llista_vertexs_intermitjos()");
      }
}

//Srveix tant per la fase d'expansio com per la d'interrogacio.
//Per l'óexpansi no borra els nodes generats si no ha pogut
//connectar-los i si en canvi per la óinterrogaci.

////////////////////////////////////////////////////////////////
// trajectoria_aleatoria_de_salts_en_graf_config()
//
// Rutina que crea una òtrajectria aleatoria de salts, des de
// un vertex donat d'un graf. La òtrajectria òaleatria de salts,
// consisteix en un ícam de èvrtexs ié s emmagatzemat com a
// èvrtexs i arestes dins del graf. Lú'ltim èvrtex de cada ícam,
// s'intenta connectar amb altres èvrtexs del graf que no
// pertanyin al mateix component connex que els èvrtexs
// del ícam. Encara que no hi íhag óconnexi, els èvrtexs afegits
// no es suprimeixen.
//
// Arguments:
//   elem_interp,nom_robot,pdh,jr,T,conf_rob_esq,conf_rob_dre
// à    Parmetres d'una óconfiguraci que esdevenen constants i
//      dades per generar-les.
//   pvertex_exp:
//     Punter a un èvrtex del graf que volem expandir.
//   p_G:
//     Estructura on volem guardar el ícam generat.
//   Retorna:
//     Retorna un enter, queé s el èvrtex amb que s'ha connectat
//     lú'ltim èvrtex de la òtrajectria òaleatria de salts.
//
////////////////////////////////////////////////////////////////

int trajectoria_aleatoria_de_salts_en_graf_config (char* nom_fase,
                                                   DADES_CONFIG* dades_config,
                                                   list pvertex_exp,graph* p_G)
{
  CONFIG* config_en_bola;
  CONFIG* config_act;
  TAULA_VERTEX_VEINS* vertex_veins;
  list pvertex_ultim; É              //s per saber lú'ltim pvertex del graf abans de la òtrajectria.
  list pvertex_act;
```

```
vertex ult_vertex; É              //s lú'ltim èvrtex del graf p_G.
vertex vertex_act;
vertex vertex_seg;

int vertex_connexio,index_vei,cont;

bool hi_ha_connexio,hi_ha_veins;


//Inicialitzacions de diverses variables.
vertex_connexio=-1;
hi_ha_connexio=FALSE;
hi_ha_veins=FALSE;
vertex_veins=crear_taula_vertex_veins();

//Marquem tots els vertexs com a no visitats.
if (mark_all_unvisited(*p_G)==ERROR)
  cerror("Error en trajectoria aleatoria de salts en graf config()0");

//Diem que el pvertex_acté s el pvertex d'on volem generar la òtrajectria òaleatria.
pvertex_act=pvertex_exp;
vertex_act=VERTEX_NUMBER(pvertex_act);

//Marquem tos els pvertexs connexos a l'actual.
marcar_connexes_a_vertex_graf_config(pvertex_act);

//éNoms busquem com ïvens els connexos al node c.
hi_ha_veins=cercar_veins_vertex_c_graf_config(dades_config,
                                              pvertex_act,*p_G,vertex_veins);

//printf("\n Intentant connectar...\n");
if (hi_ha_veins)
  {
    //Busquem els g_NVEINS éms propers al vertex c del graf.
    index_vei=cercar_primera_connexio_graf_config (dades_config,
                                                   pvertex_act,vertex_veins,*p_G);

    //Hem de comprovar que estem dins del rang de la taula.
    if (index_vei<vertex_veins->n)
      {
        if (add_edge(*p_G,vertex_act,(vertex_veins->t)[index_vei].num_vertex,NULL)==ERROR)
          cerror("Error en trajectoria aleatoria de salts en graf config()2");
        hi_ha_connexio=TRUE;

        //Indiquem amb quin èvrtex hem connectat.
        vertex_connexio=(vertex_veins->t)[index_vei].num_vertex;
      }
  }
//printf("Acabada connexio.\n");

//Comptador dels èvrtexs de la òtrajectria òaleatria de salts.
cont=0;

//Miro lí'ndex de lú'ltim vertex del graf de configuracions.
pvertex_ultim=last(*p_G);
ult_vertex=VERTEX_NUMBER(pvertex_ultim);

//Els èvrtexs que afegirem a la òtrajectria òaleatria seran édesprs de lú'ltim.
vertex_seg=ult_vertex+1;

//Necessitem saber quinaé s la configuracio del èvrtex a extendre.
config_act=CONFIG_VERTEX(pvertex_act);

while ((cont<=g_max_profund)&&(!hi_ha_connexio))
  {


    if (!strcmp(nom_fase,"expansio"))
       config_en_bola=generar_configuracio_previa_colisio(dades_config,config_act);
    if (!strcmp(nom_fase,"interrogacio"))
       config_en_bola=generar_configuracio_previa_colisio(dades_config,config_act);

    if (config_en_bola!=NULL)
      {
        //Afegim la config_en_bola al graf, i l'aresta corresponent.
        if(add_vertex(p_G,vertex_seg, (generic_ptr) config_en_bola)==ERROR)
          cerror("Error:en trajectoria aleatoria de salts en graf config()3");
        if(add_edge(*p_G,vertex_act,vertex_seg,NULL)==ERROR)
          cerror("Error:en trajectoria aleatoria de salts en graf config()4");

        //El vertex que hem afegit, s'ha de marcar com a visitat ja que pertany al mateix
        //component connex que el pvertex a expandir.
        pvertex_act=find_vertex(*p_G,vertex_seg);
        VISITED(pvertex_act)=TRUE;

        hi_ha_veins=cercar_veins_vertex_c_graf_config(dades_config,

        //printf("\n Intentant connectar...\n");
        if (hi_ha_veins)
          {
            index_vei=cercar_primera_connexio_graf_config (dades_config,
                                                           pvertex_act,
```

```
                                                    vertex_veins,*p_G);

            //s'ha de comprovar que el valor de index_veié s un valor àvlid dintre del rang de la taula.
            if (index_vei<vertex_veins->n)
              {
                if (add_edge(*p_G,vertex_seg,(vertex_veins->t)[index_vei].num_vertex,NULL)==ERROR)
                  cerror("Error_en_trajectoria_aleatoria_de_salts_en_graf_config()5");
                hi_ha_connexio=TRUE;
                vertex_connexio=(vertex_veins->t)[index_vei].num_vertex;
              }
          }
        //printf("Acabada_óconnexi.\n");

        //Fem l'óactualitzaci de les variables de recorregut dels èvrtexs de la òtrajectria.
        config_act=config_en_bola;
        vertex_act=vertex_seg;
        vertex_seg=vertex_act+1;
        cont=cont+1;

        //printf("íCam_de_Longitud_%d\n",cont);

      }
    else
      {
        printf("No_s'ha_pogut_generar_una_configuracio_dins_la_bola_amb_els_parametres_establerts\n");
        break;
      }
  }
  printf("íCam_de_Longitud_%d\n",cont);
  return vertex_connexio;
}


////////////////////////////////////////////////////////////
// buscar_components_connexos_del_graf_config()
//
// Rutina que donat un graf, guarda en una taula el nombre n
// de components connexes éms gransé, s a dir, els que tenen éms
// èvrtexs continguts en cada un d'ells.
//
// Arguments:
//   G:
//     Estructura on es representa el graf de configuracions.
//   taula_components:
//     Taula que mostra la ódistribuci de n components connexes
// é    ms grans del graf, i que guarda per a cada un d'ells
//     un èvrtex representatiu.
//   Retorna:
//     Retorna un enter, queé s el nombre de components connexes
//     del graf donat.
//
////////////////////////////////////////////////////////////

int buscar_components_connexos_del_graf_config(graph G,COMPONENT* taula_components)
{

  list pvertex_act=NULL;
  int cont_comp_connexos=1;       //Com a ímmim hi àhaur un component connex.
  int imin;

  //Primer s'ha de marcar tots els èvrtexs com a no visitats.
  if (mark_all_unvisited(G)==ERROR)
    cerror("Error_en__buscar_components_connexos_del_graf_config()0");

  //Fem un recorregut lineal per tots els èvrtexs del graf.
  pvertex_act=G;
  while ((pvertex_act!=NULL)&&(!VISITED(pvertex_act)))
    {
      //Compta el nombre de èvrtexs de cada component connex.
      gl_cont_vertex_connexos_local=0;

      //Fem una óexploraci de tots els èvrtexs connexos del vertex actual.
      if (depth_first_vertices_edges(G,pvertex_act,NULL,p_action_vertex_before,NULL,NULL)==ERROR)
                cerror("Error_en_buscar_components_connexos_del_graf_config()1");

      //éNoms si tenim taula_components voldrem guardar els components connnexes.
      if (taula_components!=NULL)
        {
          //Busquem lí'ndex del ímnim de la taula.
          imin=index_num_vertexs_minim(taula_components);

          //Actualitzem el ímnim de la taula amb el comptador de èvrtexs trobat.
          if((taula_components[imin].num_vertexs)<(gl_cont_vertex_connexos_local))
            {
              taula_components[imin].num_vertexs=gl_cont_vertex_connexos_local;
              taula_components[imin].pvertex=pvertex_act;
            }

        }
      //Fem una cerca del primer èvrtex del graf que no estigui visitat.
      //íAix trobarem el primer èvrtex del üsegent component connex.
```

```
      while((pvertex_act=list_iterator(G,pvertex_act))!=NULL)
        {
          if(!(VISITED(pvertex_act)))
            {
              //Si trobem un èvrtex que no àest visitat, aleshores
              //segur que el graf ét un component connex éms dels
              //que tenia.
              cont_comp_connexos=cont_comp_connexos+1;
              g_cont_comp_connexes= g_cont_comp_connexes+1;
              break;
            }
        }
    }
  if (taula_components!=NULL)
    {
      //Ordenem la taula_components per components de éms grossos a éms petits.
      qsort(taula_components,g_NUM_COMPONENTS,sizeof(COMPONENT),compare_num_vertexs);
      escriure_taula_components(taula_components);
    }
  return cont_comp_connexos;
}

static status  p_action_vertex_before(list pvertex)
{
  gl_cont_vertex_connexos_local=gl_cont_vertex_connexos_local+1;
  //printf("%d\n",VERTEX_NUMBER(pvertex));
  //printf("%d\n",cont_vertex_connex);
  return OK;
}


void eliminar_components_graf_config(graph* p_G,int filtrar_min,int filtrar_max)
{

  list pvertex_act=NULL;
  list pvertex_aux=NULL;
  list paresta_act;
  bool condicio_de_filtratge;

  if (mark_all_unvisited(*p_G)==ERROR)
    cerror("Error en eliminar_components_connexos_petits_en_graf_config()");

  pvertex_act=*p_G;
  while ((pvertex_act!=NULL)&&(!VISITED(pvertex_act)))
    {
      gl_cont_vertex_connexos_local=0;

      if (depth_first_vertices_edges(*p_G,pvertex_act,NULL,p_action_vertex_before,NULL,NULL)==ERROR)
        cerror("Error en eliminar_components_connexos_petits_en_graf_config()");

      //Borrar un unic component.
      if (filtrar_min==filtrar_max)
        condicio_de_filtratge=((gl_cont_vertex_connexos_local==filtrar_min)&&
                                                              (gl_cont_vertex_connexos_local==filtrar_max));
      //Borrar interval de components.
      else
        condicio_de_filtratge=((gl_cont_vertex_connexos_local<=filtrar_min)||
                                                                (gl_cont_vertex_connexos_local>=filtrar_max));

      if (condicio_de_filtratge)
        {
          if (mark_all_unvisited(*p_G)==ERROR)
             cerror("Error en eliminar_components_connexos_petits_en_graf_config()");
           if (depth_first_vertices_edges(*p_G,pvertex_act,NULL,NULL,NULL,NULL)==ERROR)
             cerror("Error en eliminar_components_connexos_petits_en_graf_config()");

           //actualitzo el pvertex_act,com el primer vertex que no estigui marcat,seguint
           //endavant, ja que aixi no sera un dels vertexs borrats.
           while((pvertex_act=list_iterator(*p_G,pvertex_act))!=NULL)
             if(!(VISITED(pvertex_act)))
               break;

           //proces d eliminacio del graf de tots els vertex i de les arestes dels vertexs
           //marcats previament.
           pvertex_aux=NULL;
          while((pvertex_aux=list_iterator(*p_G,pvertex_aux))!=NULL)
            {
              if(VISITED(pvertex_aux))
                {
                  paresta_act=NULL;
                  while((paresta_act=list_iterator(EMANATING(pvertex_aux),paresta_act))!=NULL)
                    if((delete_edge(*p_G,SOURCE(paresta_act),DESTINATION(paresta_act)))==ERROR)
                      cerror("Error en eliminar_components_connexos_petits_en_graf_config()6");

                  if((delete_vertex_with_pointer(p_G,pvertex_aux))==ERROR)
                    cerror("Error en eliminar_components_connexos_petits_en_graf_config()7");
                }

            }
        }
      else
        {
```

```
        //actualitzo el nou pvertex_act com el primer que no
        //estigui visitat, i aixi indicara el primer vertex
        //del seguent component connex.
        while((pvertex_act=list_iterator(*p_G,pvertex_act))!=NULL)
          if(!(VISITED(pvertex_act)))
            break;
        }
    }

}

////////////////////////////////////////////////////////////////
// eliminar_seguents_marca_graf_config()
//
// Aquesta rutina elimina del graf de configuracions tots els
// èvrtexs i les seves respectives arestes, que estan per sota
// del vertex marca.
// Arguments:
//   pvertex_marca:
// É    s el punter al vertex marca.
//   p_G:
//     Es l'estructura on guardem el mapa de rutes generat.
// Retorna:
//   void
//
////////////////////////////////////////////////////////////////

static void eliminar_seguents_marca_graf_config(graph* p_G,list pvertex_marca)
{
  list pvertex_act;
  list paresta_act;

  //El punter al vertex marcaé, s lú'ltim vertex que s'havia
  //generat en la fase de óconstrucci o óexpansi i no el
  //volem borrar.
  pvertex_act=NEXT(pvertex_marca);
  while (pvertex_act!=NULL)
    {
      //Fem un recorregut de les arestes que surten de cada vertex, i les
      //anem borrant.
      paresta_act=NULL;
      while((paresta_act=list_iterator(EMANATING(pvertex_act),paresta_act))!=NULL)
        if((delete_edge(*p_G,SOURCE(paresta_act),DESTINATION(paresta_act)))==ERROR)
          cerror("Error en eliminar_seguents_marca_graf_config()6");

      //Per borrar el pvertex actual primer hem de guardar
      //el pvertex üsegent en una variable auxiliar, on podem utilitzar
      //el pvertex_marca per exemple.
      pvertex_marca=NEXT(pvertex_act);

      //Ara si que podem borrar el pvertex_act;
      if((delete_vertex_with_pointer(p_G,pvertex_act))==ERROR)
        cerror("Error en eliminar_seguents_marca_graf_config()7");
      //EL pvertex actual écont el pvertex üsegent.

      pvertex_act=pvertex_marca;
    }
}

////////////////////////////////////////////////////////////////
// desar_graf_config_fitxer_depuracio()
//
// Aquesta rutina guarda en un fitxer de depuracio, la órelaci
// d'arestes i èvrtexs donat un graf. Considerem que forma part
// de la ódepuraci completa.
//
// Arguments:
//    G: Es la variable que écont el graf.
// Retorna:
//    void.
//
////////////////////////////////////////////////////////////////

static void desar_graf_config_fitxer_depuracio(graph G)
{
  list V=NULL;;

  //escrivim els vertexs i el seu numero identificador.
  while((V=list_iterator(G, V))!=NULL)
    {
      TRACE1((g_fd,"[%u, %u]", VERTEX_NUMBER(V),VERTEX_INDEX(V)));
      TRACE1((g_fd,"->"));
      //escrivim les arestes que surten d un vertex.
      {
        list E=NULL;
        TRACE1((g_fd,"[vdest,sense]: "));
        while((E=list_iterator(EMANATING(V), E))!=NULL)
          TRACE1((g_fd,"[%u,%d] ",DESTINATION(E),SENSE(E)));
      }
      TRACE1((g_fd,"\n"));
    }
  TRACE1((g_fd,"\n"));
```

```
}

/////////////////////////////////////////////////////////////
// desar_visitats_fitxer_depuracio_graf_config()
//
// Rutina que guarda en un fitxer de depuracio, els èvrtexs
// que estan visitats donat un graf.
//
// Arguments:
//    G: Es la variable que écont el graf.
// Retorna:
//    void.
//
/////////////////////////////////////////////////////////////

static void desar_visitats_fitxer_depuracio_graf_config(graph G)
{
  list pvertex_act=NULL;
  vertex num_vertex;

  //Fem un recorregut del graf i guardem en un fitxer els èvrtexs
  //que estan visitats.
    while((pvertex_act=list_iterator(G, pvertex_act))!=NULL)
    {
      if (VISITED(pvertex_act)==TRUE)
        {
          num_vertex=VERTEX_NUMBER(pvertex_act);
          TRACE2((g_fd,"El vertex %u s'ha marcat\n",num_vertex));
        }
    }
    TRACE2((g_fd,"\n"));
}

/////////////////////////////////////////////////////////////
// distancia_mitja_entre_obstacles()
//
// Rutina que calcula la àdistncia mitja dels èvrtexs connectats
// als èvrtexs a expansionar, i el mateix pels èvrtexs que no
// expansionem
//
// Arguments:
//   elem_aleat,elem_interp,pdh,jr,T,conf_rob_esq,conf_rob_dret:
// ó     Sn variables, que ósn constants per tota óconfiguraci
//       i que les necessitem per generar-les.
//   p_G:
//       Es l'estructura on guardem el mapa de rutes generat.
// Retorna:
//   La àdistncia mitja dels èvrtexs a expandir.
//
/////////////////////////////////////////////////////////////


static real distancia_mitja_entre_obstacles(DADES_CONFIG* dades_config,graph* p_G)
{
  TAULA_VERTEX_EXP* vertexs_expand;
  int i,cont_arestes;
  real distancia,dist_mitja_vertex_exp,suma_total;
  real dist_mitja_exp,dist_mitja_no_exp;
  int cont_vertexs_expand,cont_vertexs_no_expand;
  list paresta_act,pvertex_desti;
  list pvertex;

  vertexs_expand=crear_taula_vertexs_expand();

  //Extenem els vertexs de la taula.
  cercar_vertexs_expand_graf_config(*p_G,vertexs_expand);

  //Calcul de la distancia mitja pels èvrtexs que expandim.
  cont_vertexs_expand=vertexs_expand->n;
  i=0;
  suma_total=0;
  cont_arestes=0;
  while (i<vertexs_expand->n)
    {
      dist_mitja_vertex_exp=0;
      cont_arestes=0;
      paresta_act=NULL;
      while((paresta_act=list_iterator(EMANATING((vertexs_expand->t)[i].pvertex),
                                       paresta_act))!=NULL)
        {
          pvertex_desti=VERTEX_POINTER(paresta_act);
          distancia=distancia_fina_entre_configuracions(dades_config,
                                                        CONFIG_VERTEX((vertexs_expand->t)[i].pvertex),
                                                        CONFIG_VERTEX(pvertex_desti));

          // dist_mitja_vertex_exp=dist_mitja_vertex_exp+distancia;
          //cont_arestes=cont_arestes+1;

          if (distancia>dist_mitja_vertex_exp)
            dist_mitja_vertex_exp=distancia;
        }
      //if (cont_arestes!=0)
```

```
      //dist_mitja_vertex_exp=((real)dist_mitja_vertex_exp/(real)cont_arestes);

      //Els èvrtexs aillats no tenen veins, i no els contem
      //per calcular la distancia mitja.
      if (dist_mitja_vertex_exp==0)
        cont_vertexs_expand=cont_vertexs_expand-1;
      suma_total=suma_total+dist_mitja_vertex_exp;
      i=i+1;
    }
  dist_mitja_exp=((real)suma_total/(real)cont_vertexs_expand);
  printf("suma_total: %f\n",suma_total);
  printf("distancia_mitja: %f\n",dist_mitja_exp);

  //Calcul de la distancia mitja pels èvrtexs que no expandim.
  suma_total=0;
  cont_vertexs_no_expand=0;
  pvertex=NULL;
  while((pvertex=list_iterator(*p_G,pvertex))!=NULL)
    {
      if (PES(CONFIG_VERTEX(pvertex))==0)
      {
        dist_mitja_vertex_exp=0;
        cont_arestes=0;
        paresta_act=NULL;
        while((paresta_act=list_iterator(EMANATING(pvertex),paresta_act))!=NULL)
          {
          pvertex_desti=VERTEX_POINTER(paresta_act);
          distancia=distancia_fina_entre_configuracions(dades_config,
                                                        CONFIG_VERTEX(pvertex),
                                                        CONFIG_VERTEX(pvertex_desti));

          //dist_mitja_vertex_exp=dist_mitja_vertex_exp+distancia;
          //cont_arestes=cont_arestes+1;
          if (distancia>dist_mitja_vertex_exp)
            dist_mitja_vertex_exp=distancia;
          }
        //if (cont_arestes!=0)
        //dist_mitja_vertex_exp=((real)dist_mitja_vertex_exp/(real)cont_arestes);

      cont_vertexs_no_expand=cont_vertexs_no_expand+1;
      suma_total=suma_total+dist_mitja_vertex_exp;
      }
    }
  dist_mitja_no_exp=((real)suma_total/(real)cont_vertexs_no_expand);
  printf("suma_total: %f\n",suma_total);
  printf("distancia_mitja: %f\n",dist_mitja_no_exp);

  return (dist_mitja_exp);
}

void cercar_cami_minim_lliure(DADES_CONFIG* dades_config, list* ptraject)
{
  wgraph g;
  TAU_LIST t;
  int i;
  bool cami_lliure;

  CONFIG* nconfig;
  CONFIG* config;

  list traject_curta;
  list traject_aux;

  list pnode;

  //Variables per calcular el temps de òcmput.
  struct tms t_init, t_end;
  double tm,ts;

  times(&t_init);

  initialize_wgraph(&g);
  traspassar_list_TAU_PLIST(*ptraject, &t) ;
  crear_wgraph_de_traject(&g,dades_config,&t);
  //print_wgraph(&g);

  do
    {
      dijkstra_wgraph(&g,1);

      cami.n=0;
      find_path_wgraph(1,g.nvertices,parent);
      cami_lliure=visualitzar_cami_minim(&g,dades_config,&t);
    }while (cami_lliure!=TRUE);


  if(init_list(&traject_curta)==ERROR)
    cerror("Error_en_cercar_cami_minim_lliure()1");

  for(i=0;i<cami.n;i++)
    {
      pnode=t.nodes[cami.v[i]];
```

```
        config=(CONFIG*)DATA (pnode);

        nconfig=crear_configuracio();
        copiar_config_a_nova_config(config,nconfig);

        if ((append(&traject_curta,(generic_ptr)(nconfig)))==ERROR)
          cerror("Error en cercar cami minim lliure()2");
      }
    traject_aux=*ptraject;

    *ptraject=traject_curta;

    destroy_list(&traject_aux,p_func_f1);;

    for (i=0; i<cami.n; i++)
      printf("x %d ", cami.v[i]);
    printf("\n");

    times(&t_end);

    ts=(double)sysconf(_SC_CLK_TCK); //tics per second
    tm=(double)((t_end.tms_utime-t_init.tms_utime)+
                (t_end.tms_stime-t_init.tms_stime)+
                (t_end.tms_cutime-t_init.tms_cutime)+
                (t_end.tms_cstime-t_init.tms_cstime))/ts;

    printf("Temps d'óexecuci: %f\n",tm);

}
static void crear_wgraph_de_traject(wgraph* g, DADES_CONFIG* dades_config,
                                    TAU_LIST* t)
{
    real distancia;
    int i,j;
    CONFIG* configi;
    CONFIG* configf;

    g->nvertices=t->n;

    for(i=1;i<=t->n;i++)
      {
        configi=(CONFIG*) DATA(t->nodes[i]);
        for(j=i+1;j<=t->n;j++)
          {
            configf=(CONFIG*) DATA(t->nodes[j]);
            distancia=distancia_fina_entre_configuracions(dades_config,

            if (distancia!=BIG)
              insert_edge_wgraph(g,i,j,FALSE,distancia);
          }

      }

}

static bool visualitzar_cami_minim(wgraph* g, DADES_CONFIG* dades_config,
                                   TAU_LIST* t)
{
    bool cami_lliure=TRUE;
    int i=1;

    list pnode=NULL;
    list pnode_seg=NULL;
    CONFIG* configi;
    CONFIG* configf;

    INFO_PLANIF_LOCAL* planificador_local;

    //Com a minim un ícam tindra dos nodes.
    i=0;
    while (i<(cami.n-1))
      if (llegir_lliure_aresta_wgraph(g,cami.v[i],cami.v[i+1])==TRUE)
        i=i+1;
      else
        {
          //Sera la primera aresta del ícam ímnim que no s'ha detectat la ócollisi.
          pnode=t->nodes[cami.v[i]];
          pnode_seg=t->nodes[cami.v[i+1]];
          break;
        }


    while (i<(cami.n-1))
      {
        configi=(CONFIG*) DATA(pnode);
        configf=(CONFIG*) DATA(pnode_seg);

        //if (llegir_lliure_aresta_wgraph(&g,cami.v[i],cami.v[i+1])==FALSE)
        planificador_local=planificador_local_configi_configf(TRUE,dades_config,


        if (planificador_local->solucio)
```

```
        {
          alliberar_info_planificador_local(&planificador_local);
          afegir_lliure_aresta_wgraph(g,cami.v[i],cami.v[i+1],FALSE,TRUE);

          i=i+1;
          while (i<(cami.n-1))
            if (llegir_lliure_aresta_wgraph(g,cami.v[i],cami.v[i+1])==TRUE)
              i=i+1;
            else
              {
                pnode=t->nodes[cami.v[i]];
                pnode_seg=t->nodes[cami.v[i+1]];
                break;
              }
        }
      else
        {
          alliberar_info_planificador_local(&planificador_local);
          delete_edge_wgraph(g,(cami.v[i]),(cami.v[i+1]),FALSE);
          cami_lliure=FALSE;
          break;
        }
    }
  return cami_lliure;
}
```

H.13 configuracio.h

```
#ifndef CONFIGURACIO_H
#define CONFIGURACIO_H

/////////////////////////////////////////////////////
/// @file configuracio.h
/// @brief Les rutines que treballen amb configuracions.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       Implementem les rutines que treballen amb el
///       tipus CONFIG, que constitueixen les configuracions
///       del sistema a planificar. Pel present projecte,
///       una óconfiguració s la cadena àcinemtica tancada
///       que formen els dos robots RX60 quan transporten
///       un objecte subjectat amb les seves pinces.
/////////////////////////////////////////////////////

#include <string.h>
#include "globals.h"
#include "transf_ctnt.h"
#include "vector.h"
#include "objecte.h"
#include "robots.h"
#include "colisio.h"

typedef struct configu
{
  CONF_ROBOT* robote;
  CONF_ROBOT* robotd;
  CONF_OBJECTE* obj;
  int f;
  int n;
  real w;
}CONFIG;

typedef struct dades_config
{
  char elem_aleat[MAX_CADENA];
  char elem_interp[MAX_CADENA];
  PDH pdh;
  JNTS_RANGE jr;
  TRANSF_CTNT T;
  char conf_rob_esq[4];
  char conf_rob_dret[4];
}DADES_CONFIG;

typedef struct struct_planif_local
{
  bool solucio;
  int n;
  CONFIG* config_ant;

}INFO_PLANIF_LOCAL;


#define ROBOTE(config) ((config)->robote)
#define ROBOTD(config) ((config)->robotd)
#define OBJECTE(config) ((config)->obj)
#define N_PLAN_LOC(config) ((config)->n)
#define F_PLAN_LOC(config) ((config)->f)
#define PES(config) ((config)->w)
```

```
#define ARTICE(config)  (ARTIC(ROBOTE(config)))
#define ARTICD(config)  (ARTIC(ROBOTD(config)))
#define ARTICE_i(config,i) (ARTIC_i((ROBOTE(config)),i))
#define ARTICD_i(config,i) (ARTIC_i((ROBOTD(config)),i))
#define OBJ_TRANSL_i(config,i) TRANSL_i((OBJECTE(config)),i)
#define OBJ_ROT_i(config,i) ROT_i((OBJECTE(config)),i)
#define SOLUCE(config) (SOLUC((ROBOTE(config))))
#define SOLUCD(config) (SOLUC((ROBOTD(config))))


//Operacions constructores

/////////////////////////////////////////////////////
///  @brief óCreaci d'una óconfiguraci.
///
///  Reserva òmemria per una óconfiguraci, que consta de
///  sis angles pel robot esquerra, sis éms pel
///  dret i el vector d'óorientaci i de ótranslaci de l'objecte.
///
///  @return  Una óconfiguraci.
/////////////////////////////////////////////////////

CONFIG* crear_configuracio();

/////////////////////////////////////////////////////
///  @brief óEliminaci d'una óconfiguraci
///
///  Allibera òmemria per una óconfiguraci, que consta de
///  sis angles pel robot esquerra, sis éms pel
///  dret i el vector d'óorientaci i de ótranslaci de l'objecte.
///
///  @param config Par. ent/sort. Una óconfiguraci.
///
/////////////////////////////////////////////////////

void alliberar_configuracio(CONFIG** config);

/////////////////////////////////////////////////////
///  @brief Visualitzar una óconfiguraci.
///
///  Visualitza en Webots
///  una óconfiguraci donada la cadena
///  tancada a tractar.
///
///  @param config Par. ent. Configuracio que volem visualitzar.
/////////////////////////////////////////////////////

void canviar_a_configuracio_predeterminada(CONFIG* config);

/////////////////////////////////////////////////////
///  @brief Guarda la óconfiguraci actual.
///
///  Guarda la óconfiguraci actual en les taules
///  angle_rob_esq, angle_rob_dret i transl_objecte
///  de treball de webots_interface.h. Es fa íaix èperqu Webots
/// é noms permet emmagatzemar la óinformaci dels elements
///  de l'escena de treball en la mateixa zona de òmemria per
///  tota una ósimulaci.
///
/////////////////////////////////////////////////////

void obtenir_configuracio_actual();

/////////////////////////////////////////////////////
///  @brief Copia la óconfiguraci a una altre variable.
///
///  Copia la óconfiguraci obtinguda en les variables
///  de Webots, en una nova variable tipus CONFIG.
///
///  @param config Par. ent/sort. óConfiguraci a on copiem
///                               la óconfiguraci obtinguda.
/////////////////////////////////////////////////////

void copiar_obtenir_config_a_configuracio(CONFIG* config);

/////////////////////////////////////////////////////
///  @brief Intenta tancar la cadena movent l'objecte.
///
///  Movent l'objecte de Webots amb el mouse es
///  recalcula la àcinemtica inversa per cada robot
///  i es visualitza en cas d'existir ósoluci.
/// é Tamb s'extreu per pantalla si hi ha ósoluci
///  en forma de cadena tancada, i en cas de que
///  n'hi hagi s'informa  de sié s lliure de col.lisions.
///
///  @param dades_config Par. ent.  Punter a la óinformaci
///                      constant associada a l'óobtenci
///                      d'una cadena àcinemtica tancada.
///
///  @param config Par. ent/sort. Configuracio atual on-line.
///
///  @return Un boolea que indica si a trobat una óconfiguraci lliure.
```

```
/////////////////////////////////////////////////////

bool canviar_a_configuracio_on_line(DADES_CONFIG* dades_config,CONFIG* config);

/////////////////////////////////////////////////////
///  @brief Troba òaleatriament una óconfiguraci lliure.
///
///  Mostreja òaleatriament els angles del robot actiu
///  fins que el robot passiu tanqui la cadena àcinemtica.
///  La óconfiguraci obtinguda àser lliure o no depenen
///  del valor de la variable detectar_colisions.
///
///  @param detectar_colisions Par. ent. Sié s certa, s'inspecciona si
///                            les configuracions obtingudes ósn lliures
///                            de col.lisions, altrament énoms si es
///                            produeix tancament.
///
///  @param dades_config Par. ent. Dades constants associades
///                      a l'óobtenci de la cadena àcinemtica
///                      tancada.
///
///  @param config  Par. ent/sort. óConfiguraci lliure
///                 que obtindrem.
///
/////////////////////////////////////////////////////

void canviar_a_configuracio_lliure(bool detectar_colisions,DADES_CONFIG* dades_config,CONFIG* config);

/////////////////////////////////////////////////////
///  @brief àTrasps d'una óconfiguraci a una altra.
///
///  Copia la óinformaci emmagatzemada en una óconfiguraci
///  a una altra variable.
///
///  @param config  Par. ent. óConfiguraci d'entrada.
///
///  @param nconfig  Par. ent/sort. óConfiguraci copiada.
///
/////////////////////////////////////////////////////

void copiar_config_a_nova_config(CONFIG* config,CONFIG* nconfig);

/////////////////////////////////////////////////////
///  @brief Troba una óconfiguraci propera a un obstacle.
///
///  Genera direccions òaleatries per una óconfiguraci
///  config_act  i troba laú ltima lliure de col.lisions
///  per aquella ódirecci. Per seguir una ódirecci
///  s'empra el planificador local.
///
///  @param dades_config  Par. ent. Dades constants associades
///                       a l'óobtenci de la cadena
/// à                      cinemtica tancada.
///  @param config_act  Par. ent/sort. óConfiguraci
///                     des d'on volem generar una ódirecci òaleatria.
///
///  @return  Laú ltima óconfiguraci lliure de col.lisions per una
/// ó          direcci òaleatria.
/////////////////////////////////////////////////////

CONFIG* generar_configuracio_previa_colisio(DADES_CONFIG* dades_config,CONFIG* config_act);

/////////////////////////////////////////////////////
///  @brief Genera una óconfiguraci lliure dintre d'una bola de àdistncia.
///
///  Genera una óconfiguraci òaleatria interna en una bola de
///  de àdistncia, centrada en la óconfiguraci config_act.
///  A éms, la óconfiguraci generadaé s lliure de col.lisions, íaix com
///  la crida del planificador local entre la de partida i la aquesta.
///
///  @param dades_config  Par. ent. Dades constants associades
///                       a l'óobtenci de la cadena àcinemtica tancada.
///  @param config_act  Par. ent. óConfiguraci des d'on d'on volem
///                     generar una nova óconfiguraci.
///
///  @return   Una óconfiguraci òaleatria interna en una bola de
///            de àdistncia.
/////////////////////////////////////////////////////

CONFIG* generar_configuracio_robot_aleatori_en_bola_DIST_MAX(DADES_CONFIG* dades_config,CONFIG* config_act);

/////////////////////////////////////////////////////
///  @brief Escriu per pantalla el contingut d'una óconfiguraci.
///
///  Escriu pel canal àestndar de sortida el contingut d'una
/// ó configuraci: sis angles pel robot esquerra, sis éms pel
///  dret i el vector d'óorientaci i de ótranslaci de l'objecte.
///
///  @param config  Par. ent óConfiguraci que escrivim.
/////////////////////////////////////////////////////

void escriure_configuracio(CONFIG* config);
```

```
/////////////////////////////////////////////////////
///  @brief Desa una óconfiguraci en un fitxer de text.
///
///  Desa una óconfiguraci en un fitxer de text.
///  El que desaé s el üsegent: sis angles pel robot esquerra,
///  sis éms pel dret i el vector d'óorientaci i de ótranslaci
///  de l'objecte.
///
///  @param fa Par. ent/sort. Fitxer on desem la óconfiguraci.
///
///  @param config  Par. ent. óConfiguraci donada.
/////////////////////////////////////////////////////

void desar_configuracio(FILE* fa,CONFIG* config);

/////////////////////////////////////////////////////
///  @brief Carrega en òmemria una óconfiguraci des d'un fitxer de text.
///
///  Carrega una óconfiguraci en òmemria des d'un fitxer de text.
///  El que carregaé s el üsegent: sis angles pel robot esquerra,
///  sis éms pel dret i el vector d'óorientaci i de ótranslaci
///  de l'objecte.
///
///  @param fa Par. ent. Fitxer d'on carreguem la
/// ó           configuraci.
///
///  @param config Par. ent/sort. óConfiguraci que carreguem de fitxer.
/////////////////////////////////////////////////////

void carregar_configuracio(FILE* fa,CONFIG* config);

/////////////////////////////////////////////////////
///  @brief Crea un punter al que retorna el planificador local.
///
///  Crea un punter a INFO_PLANIF_LOCAL, queé s l'estructura
///  que guarda el que ens interessa d'una crida del
///  planificador local: la óconfiguraci previa a un àfracs
///  per col.ólisi, la óiteraci on ha passat, i el resultat
///  de la crida.
///
///  @return  Unn punter a l'estructura INFO_PLANIF_LOCAL.
/////////////////////////////////////////////////////

INFO_PLANIF_LOCAL* crear_info_planificador_local();

/////////////////////////////////////////////////////
///  @brief Allibera un punter del que retorna el planificador local.
///
///  Allibera la òmemria del punter de tipus INFO_PLANIF_LOCAL
///  que retorna el planificador local.
///
///  @param planificador_local Par. ent/sort. Allibera la variable
///                                           planificador_local.
///
/////////////////////////////////////////////////////

void alliberar_info_planificador_local(INFO_PLANIF_LOCAL** planificador_local);

/////////////////////////////////////////////////////
///  @brief Crida del planificador local entre dues configuracions.
///
///  Mira si existeix una òtrajectria entre dues
///  configuracions. Depenen del valor de  detectar_colisions
///  es mira sié s lliure o no de col.lisions.
///
///  @param detectar_colisio Par. ent.  Sié s certa, s'inspecciona si
///                          les configuracions obtingudes en la òtrajectria
/// ó                         sn lliures de col.lisions, altrament énoms
///                          si es produeix tancament.
///
///  @param dades_config Par. ent. Dades constants associades
///                      a l'óobtenci de la cadena àcinemtica tancada.
///  @param configi  Par. ent. óConfiguraci  inicial.
///
///  @param configf  Par. ent. óConfiguraci  final.
///
///  @return  Una estructura que écont tota la óinformaci
///           que necessitem.
/////////////////////////////////////////////////////

INFO_PLANIF_LOCAL* planificador_local_configi_configf(bool detectar_colisio,
                                                      DADES_CONFIG* dades_config,
                                                      CONFIG* configi,CONFIG* configf);

/////////////////////////////////////////////////////
///  @brief àDistncia íeucldia angular entre dues configuracions.
///
///  Calcula la àdistncia entre dues configuracions, com
///  la àdistncia íeucldia entre els vectors angulars
///  de la óconfiguraci config1 i la óconfiguraci config2.
///
```

```
///  @param config1 Par. ent. óConfiguraci inicial.
///
///  @param config2 Par. ent. óConfiguraci final.
///
///  @return  Un real queé s la àdistncia íeucldia
///           entre les dues configuracions.
////////////////////////////////////////////////////////

real distancia_entre_configuracions(CONFIG* config1,CONFIG* config2);

////////////////////////////////////////////////////
///  @brief àDistncia, com a longitud recorreguda pel planificador local.
///
///  Calcula la àdistncia entre dues configuracions, com
///  la longitud de la òtrajectria realitzada en la crida
///  del planificador entre aquestes dues.
///
///  @param dades_config Par. ent. Dades constants associades
///                      a l'óobtenci de la cadena
/// à                     cinemtica tancada.
///  @param configi  Par. ent. óConfiguraci inicial.
///
///  @param configf  Par. ent. óConfiguraci final.
///
///  @return  Un real queé s la àdistncia íeucldia
///           entre les dues configuracions.
////////////////////////////////////////////////////////

real distancia_fina_entre_configuracions(DADES_CONFIG* dades_config,CONFIG* configi,CONFIG* configf);

////////////////////////////////////////////////////
///  @brief Compara les solucions del robot passiu.
///
///  Compara les solucions àcinemtiques inverses
///  del robot passiu per a dues configuracions.
///
///  @param config1 Par. ent. Primera óconfiguraci.
///
///  @param config2 Par. ent. Segona óconfiguraci.
///
///  @return  Si les dues solucions àcinemtiques
/// ó          sn iguals.
////////////////////////////////////////////////////////

bool comparar_tipus_solucions_configuracions(CONFIG* config1,CONFIG* config2);

#endif
```

H.14 configuracio.c

```
#include "configuracio.h"

////////////////////////////////////////////////////
//Rutines privades a configuracio.c
////////////////////////////////////////////////////
static bool canviar_a_configuracio_donat_objecte(bool detectar_colisions,DADES_CONFIG* dades_config,
                                                 CONF_OBJECTE* objecte,CONFIG* config);
static bool canviar_a_configuracio_objecte_mouse(DADES_CONFIG* dades_config,CONFIG* config);
static bool canviar_a_configuracio_donat_robot(bool detectar_colisions,
                                               char* robot_actiu,DADES_CONFIG* dades_config,
                                               CONF_ROBOT* robot,CONFIG* config);
static bool canviar_a_configuracio_donat_robot1(bool detectar_colisions,char* robot_actiu,
                                                DADES_CONFIG* dades_config,CONF_ROBOT* robot,
                                                CONFIG* config);
CONFIG* crear_configuracio()
 {
  CONFIG* config;

  if(NEW(config,1,CONFIG)==NULL)
    cerror("error de malloc en crear_configuracio()\n");
  ROBOTE(config)=crear_robot();
  ROBOTD(config)=crear_robot();
  OBJECTE(config)=crear_posicio_objecte();
  N_PLAN_LOC(config)=0;
  F_PLAN_LOC(config)=0;
  PES(config)=0;

  return config;
}
void alliberar_configuracio(CONFIG** config)
{
  alliberar_robot(&(ROBOTE(*config)));
  alliberar_robot(&(ROBOTD(*config)));
  alliberar_posicio_objecte(&(OBJECTE(*config)));
  free(*config);
  *config=NULL;
}
INFO_PLANIF_LOCAL* crear_info_planificador_local()
{
  INFO_PLANIF_LOCAL* planificador_local;
```

```
  if(NEW(planificador_local,1,INFO_PLANIF_LOCAL)==NULL)
    cerror("error de malloc en crear info planificador local()\n");
  planificador_local->config_ant=crear_configuracio();

  return planificador_local;
}
void alliberar_info_planificador_local(INFO_PLANIF_LOCAL** planificador_local)
{
  alliberar_configuracio(&((*planificador_local)->config_ant));
  free(*planificador_local);
  (*planificador_local)=NULL;
}
void canviar_a_configuracio_predeterminada(CONFIG* config)
{
  canviar_angles_robot("esquerra",ROBOTE(config));
  canviar_angles_robot("dret",ROBOTD(config));
  canviar_posicio_objecte(OBJECTE(config));
}
void obtenir_configuracio_actual()
{
  obtenir_angles_robot("esquerra");
  obtenir_angles_robot("dret");
  obtenir_posicio_objecte();
}
static bool canviar_a_configuracio_donat_objecte(bool detectar_colisions,
                                                 DADES_CONFIG* dades_config,
                                                 CONF_OBJECTE* objecte,CONFIG* config)
{
  bool conf_valida_cine,conf_valida_cind;
  TRANSF_HOMOG Tobj,P1,Trob;
  int colisionen=0;

  conf_valida_cine=FALSE;
  conf_valida_cind=FALSE;


  //Assigno la informacio de l'objecte a la configuracio.
  copiar_objecte_a_nou_objecte(objecte,OBJECTE(config));

  cinematica_directa_objecte(OBJECTE(config),Tobj);

  //Calculem la matriu de transformacio del robot esquerra.
  producte_transf_homogenies((dades_config->T.b0e_T_ba),Tobj,P1);
  producte_transf_homogenies(P1,(dades_config->T.graspe),Trob);

  conf_valida_cine=cinematica_inversa_RX60((dades_config->pdh),
                                           &(dades_config->jr),
                                           Trob,(dades_config->conf_rob_esq),
                                           ROBOTE(config));

  //Calculem la matriu de transformacio del robot dret.
  producte_transf_homogenies((dades_config->T.b0d_T_ba),Tobj,P1);
  producte_transf_homogenies(P1,(dades_config->T.graspd),Trob);

  conf_valida_cind=cinematica_inversa_RX60((dades_config->pdh),
                                           &(dades_config->jr),
                                           Trob,(dades_config->conf_rob_dret),
                                           ROBOTD(config));

  if(conf_valida_cine&&conf_valida_cind)
    {
      //Pas dels angles de la referencia del Paul a la del RX60.
      angles_robot_ref_paper_ref_RX60(ROBOTE(config));
      angles_robot_ref_paper_ref_RX60(ROBOTD(config));

      if (detectar_colisions)
        {
          //Inicialitzar el fitxer de colisions a 0.
          inicialitzar_colisio();

          //Visualitzar la nova configuracio en webots.
          canviar_a_configuracio_predeterminada(config);
          avancar_simulacio();

          //Llegir el fitxer de colisions.
          colisionen=obtenir_colisio();
        }
    }
  return (conf_valida_cine&&conf_valida_cind)&&(!colisionen);
}
bool canviar_a_configuracio_on_line(DADES_CONFIG* dades_config,CONFIG* config)
{
  bool conf_valida_cine,conf_valida_cind;
  TRANSF_HOMOG Tobj,P1,Trobe,Trobd;
  int colisionen=0;
  int i;

  char* taula_conf[8]={"run","ruf","rdn","rdf","lun","luf","ldn","ldf"};

  conf_valida_cine=FALSE;
  conf_valida_cind=FALSE;
```

```
  obtenir_posicio_objecte();
  avancar_simulacio();

  copiar_obtencio_posicio_a_objecte(OBJECTE(config));

  //Assigno la informacio de l'objecte a la configuracio.

  cinematica_directa_objecte(OBJECTE(config),Tobj);

  //Calculem la matriu de transformacio del robot esquerra.
  producte_transf_homogenies((dades_config->T.b0e_T_ba),Tobj,P1);
  producte_transf_homogenies(P1,(dades_config->T.graspe),Trobe);

  //Calculem la matriu de transformacio del robot dret.
  producte_transf_homogenies((dades_config->T.b0d_T_ba),Tobj,P1);
  producte_transf_homogenies(P1,(dades_config->T.graspd),Trobd);


  conf_valida_cine=cinematica_inversa_RX60((dades_config->pdh),
                                           &(dades_config->jr),
                                           Trobe,(dades_config->conf_rob_esq),
                                           ROBOTE(config));
  conf_valida_cind=cinematica_inversa_RX60((dades_config->pdh),
                                           &(dades_config->jr),
                                           Trobd,(dades_config->conf_rob_dret),
                                           ROBOTD(config));

  if(!(conf_valida_cine&&conf_valida_cind))

    {
      for(i=0;i<8;i++)
        {
          if (!strcmp((dades_config->elem_interp),"esquerra"))
            conf_valida_cine=cinematica_inversa_RX60((dades_config->pdh),
                                                     &(dades_config->jr),Trobe,
                                                     taula_conf[i],ROBOTE(config));

          if (!strcmp((dades_config->elem_interp),"dret"))
            conf_valida_cind=cinematica_inversa_RX60((dades_config->pdh),
                                                     &(dades_config->jr),
                                                     Trobd,taula_conf[i],
                                                     ROBOTD(config));

          if (conf_valida_cine&&conf_valida_cind)
            break;
        }
    }

  if(conf_valida_cine&&conf_valida_cind)
    {

      //Pas dels angles de la referencia del Paul a la del RX60.
      angles_robot_ref_paper_ref_RX60(ROBOTE(config));
      angles_robot_ref_paper_ref_RX60(ROBOTD(config));

      //Inicialitzar el fitxer de colisions a 0.
      inicialitzar_colisio();

      //Visualitzar la nova configuracio en webots.
      canviar_a_configuracio_predeterminada(config);
      avancar_simulacio();

      //Llegir el fitxer de colisions.
      colisionen=obtenir_colisio();

      if (!(colisionen))
        escriure_text_a_scenee_tree("configuracio_lliure_de_col.lisions");
      else
        escriure_text_a_scenee_tree("configuracio_col.lisionant");
    }
  else
    escriure_text_a_scenee_tree("configuracio_no_assolible_cinematicament");

  return ((conf_valida_cine&&conf_valida_cind)&!colisionen);
}
static bool canviar_a_configuracio_donat_robot(bool detectar_colisions,
                                               char* robot_actiu,
                                               DADES_CONFIG* dades_config,
                                               CONF_ROBOT* robot,
                                               CONFIG* config)
{
  bool conf_valida;
  bool conf_valida_cine,conf_valida_cind;
  int colisionen=0;

  TRANSF_HOMOG Trobe,Trobd,Tobj,P1;
  char tipus_conf[4];

  if (!strcmp(robot_actiu,"esquerra"))
    {
      //Assigno la informacio del robot a la configuracio.
      copiar_robot_a_nou_robot(robot,ROBOTE(config));
```

```
    conf_valida_cine=angles_robot_ref_RX60_en_rang (robot,&(dades_config->jr));
    if (conf_valida_cine)
      {
        //Pas dels angles de la referencia del RX60 a la del Paul.
        angles_robot_ref_RX60_ref_paper(ROBOTE(config));

        cinematica_directa_RX60 ((dades_config->pdh),ROBOTE(config),Trobe,tipus_conf);

        //Calculem la matriu de transformacio de l'objecte.
        producte_transf_homogenies((dades_config->T.ba_T_b0e),Trobe,P1);
        producte_transf_homogenies(P1,(dades_config->T.graspe_inv),Tobj);

        //Calculem la matriu de transformacio del robot dret.
        producte_transf_homogenies((dades_config->T.b0d_T_ba),Tobj,P1);
        producte_transf_homogenies(P1,(dades_config->T.graspd),Trobd);

        conf_valida_cind=cinematica_inversa_RX60((dades_config->pdh),
                                                 &(dades_config->jr),
                                                 Trobd,(dades_config->conf_rob_dret),
                                                 ROBOTD(config));
        if(conf_valida_cind)
          {
            cinematica_inversa_objecte (Tobj,OBJECTE(config));

            //Pas dels angles de la referencia del Paul a la del RX60.
            angles_robot_ref_paper_ref_RX60(ROBOTE(config));
            angles_robot_ref_paper_ref_RX60(ROBOTD(config));
            if (detectar_colisions)
              {
                //Inicialitzar el fitxer de colisions a 0.
                inicialitzar_colisio();

                //Visualitzar la nova configuracio en webots.
                canviar_a_configuracio_predeterminada(config);
                avancar_simulacio();

                //Llegir el fitxer de colisions.
                colisionen=obtenir_colisio();
              }
          }
        //Per guardar cada configuracio en un arxiu.
         if (g_crear_ftraj==TRUE)
           desar_angles_robots(g_ftraj,ROBOTE(config),ROBOTD(config));
      }
    conf_valida=((conf_valida_cine)&&(conf_valida_cind)&&(!colisionen));
  }
if (!strcmp(robot_actiu,"dret"))
  {
    copiar_robot_a_nou_robot(robot,ROBOTD(config));
    conf_valida_cind=angles_robot_ref_RX60_en_rang (ROBOTD(config),&(dades_config->jr));
    if (conf_valida_cind)
      {
        //Pas dels angles de la referencia del RX60 a la del Paul.
        angles_robot_ref_RX60_ref_paper(ROBOTD(config));

        cinematica_directa_RX60 ((dades_config->pdh),ROBOTD(config),Trobd,tipus_conf);

        //Calculem la matriu de transformacio de l'objecte.
        producte_transf_homogenies((dades_config->T.ba_T_b0d),Trobd,P1);
        producte_transf_homogenies(P1,(dades_config->T.graspd_inv),Tobj);

        //Calculem la matriu de transformacio del robot dret.
        producte_transf_homogenies((dades_config->T.b0e_T_ba),Tobj,P1);
        producte_transf_homogenies(P1,(dades_config->T.graspe),Trobe);

        conf_valida_cine=cinematica_inversa_RX60((dades_config->pdh),
                                                 &(dades_config->jr),
                                                 Trobe,(dades_config->conf_rob_esq),
                                                 ROBOTE(config));
        if(conf_valida_cine)
          {
            cinematica_inversa_objecte (Tobj,OBJECTE(config));

            //Pas dels angles de la referencia del Paul a la RX60.
            angles_robot_ref_paper_ref_RX60(ROBOTE(config));
            angles_robot_ref_paper_ref_RX60(ROBOTD(config));
            if (detectar_colisions)
              {
                //Inicialitzar el fitxer de colisions a 0.
                inicialitzar_colisio();

                //Visualitzar la nova configuracio en webots.
                canviar_a_configuracio_predeterminada(config);
                avancar_simulacio();

                //Llegir el fitxer de colisions.
                colisionen=obtenir_colisio();
              }
            //Per guardar cada configuracio en un arxiu.
            if (g_crear_ftraj==TRUE)
              desar_angles_robots(g_ftraj,ROBOTE(config),ROBOTD(config));
          }
```

```
        }
      conf_valida=((conf_valida_cind)&&(conf_valida_cine)&&(!colisionen));
    }
  //Retornem si la configuracio ha estat valida cinematicament, si no detectem les colisions,
  //o si a émsé s lliure de colisions si detectem les colisions.
  return conf_valida;
}
static bool canviar_a_configuracio_donat_robot1(bool detectar_colisions,char* robot_actiu,
                                                DADES_CONFIG* dades_config,CONF_ROBOT* robot,
                                                CONFIG* config)
{

  TRANSF_HOMOG Trobe,Tobj,P1;
  char tipus_conf[4];
  int colisionen=0;

  if (!strcmp(robot_actiu,"esquerra"))
    {
      copiar_robot_a_nou_robot(robot,ROBOTE(config));
      copiar_robot_a_nou_robot(robot,ROBOTD(config));

      angles_robot_ref_RX60_ref_paper(ROBOTE(config));
      cinematica_directa_RX60 ((dades_config->pdh),ROBOTE(config),Trobe,tipus_conf);

      producte_transf_homogenies((dades_config->T.ba_T_b0e),Trobe,P1);
      producte_transf_homogenies(P1,(dades_config->T.graspe_inv),Tobj);

      cinematica_inversa_objecte (Tobj,OBJECTE(config));

      copiar_robot_a_nou_robot(ROBOTD(config),ROBOTE(config));

      if (detectar_colisions)
        {
          //Inicialitzar el fitxer de colisions a 0.
          inicialitzar_colisio();

          canviar_a_configuracio_predeterminada(config);
          avancar_simulacio();

          //Llegir el fitxer de colisions.
          colisionen=obtenir_colisio();
        }
    }

  return (!colisionen);
}
void canviar_a_configuracio_lliure(bool detectar_colisions,
                                   DADES_CONFIG* dades_config,CONFIG* config)

{
  CONF_ROBOT* robot;
  CONF_OBJECTE* objecte;
  bool conf_valida;
  char robot_actiu[MAX_CADENA];

  if (!strcmp((dades_config->elem_aleat),"objecte"))
    {
      objecte=crear_posicio_objecte();

      do
        {
          //Calcular una localitzacio aleatoria de l'objecte.
          posicio_objecte_aleatoria(objecte);

          //Canviar a una configuracio donada la configuracio de l'objecte.
          conf_valida=canviar_a_configuracio_donat_objecte(detectar_colisions,
                                                           dades_config,objecte,config);

        }while (!conf_valida);

      alliberar_posicio_objecte(&objecte);
    }
  else
    {
      robot=crear_robot();
      sprintf(robot_actiu,"%s",(dades_config->elem_aleat));

      //Fins que no trobem una configuracio lliure anem fent el bucle.
      do
        {
          //Calcular angles aleatoris per un robot.
          angles_robot_aleatoris(robot);

          //Calcular la configuracio tancada, a partir de la configuracio del robot.
          conf_valida=canviar_a_configuracio_donat_robot(detectar_colisions,
                                                         robot_actiu,dades_config,robot,config);

        }while (!conf_valida);

      alliberar_robot(&robot);
    }
}
```

```
static bool canviar_a_configuracio_objecte_mouse(DADES_CONFIG* dades_config,CONFIG* config)
{
  bool conf_valida_cine,conf_valida_cind;
  TRANSF_HOMOG Tobj,P1,Trob;
  int colisionen=0;

  conf_valida_cine=FALSE;
  conf_valida_cind=FALSE;

  //Obtenir la translacio i orientacio de l'objecte actuals.
  obtenir_posicio_objecte();
  avancar_simulacio();

   //Assigno la informacio de l'objecte a la configuracio.
  copiar_obtencio_posicio_a_objecte(OBJECTE(config));

  cinematica_directa_objecte(OBJECTE(config),Tobj);

  //Calculem la matriu de transformacio del robot esquerra.
  producte_transf_homogenies(dades_config->T.b0e_T_ba,Tobj,P1);
  producte_transf_homogenies(P1,dades_config->T.graspe,Trob);

  conf_valida_cine=cinematica_inversa_RX60((dades_config->pdh),
                                           &(dades_config->jr),
                                           Trob,
                                           (dades_config->conf_rob_esq),ROBOTE(config));

  //Calculem la matriu de transformacio del robot dret.
  producte_transf_homogenies(dades_config->T.b0d_T_ba,Tobj,P1);
  producte_transf_homogenies(P1,dades_config->T.graspd,Trob);

  conf_valida_cind=cinematica_inversa_RX60((dades_config->pdh),
                                           &(dades_config->jr),
                                           Trob,
                                           (dades_config->conf_rob_dret),ROBOTD(config));

  if(conf_valida_cine&&conf_valida_cind)
    {
      //Pas dels angles de la referencia del Paul a la del RX60.
      angles_robot_ref_paper_ref_RX60(ROBOTE(config));
      angles_robot_ref_paper_ref_RX60(ROBOTD(config));

      //Inicialitzar el fitxer de colisions a 0.
      inicialitzar_colisio();

      //Visualitzar la nova configuracio en webots.
      canviar_a_configuracio_predeterminada(config);
      avancar_simulacio();

      //Llegir el fitxer de colisions.
      colisionen=obtenir_colisio();

    }
  return (conf_valida_cine&&conf_valida_cind)&&(!colisionen);

}
void copiar_obtenir_config_a_configuracio(CONFIG* config)
{
  copiar_obtencio_angles_a_robot("esquerra",ROBOTE(config));
  copiar_obtencio_angles_a_robot("dret",ROBOTD(config));
  copiar_obtencio_posicio_a_objecte(OBJECTE(config));
}
void copiar_config_a_nova_config(CONFIG* config,CONFIG* nconfig)
{
  int i;

  for(i=0;i<=5;i++)
    {
      ARTICE_i(nconfig,i)= ARTICE_i(config,i);
      ARTICD_i(nconfig,i)= ARTICD_i(config,i);
    }
  for(i=0;i<=2;i++)
    {
      OBJ_TRANSL_i(nconfig,i)= OBJ_TRANSL_i(config,i);
      OBJ_ROT_i(nconfig,i)=OBJ_ROT_i(config,i);
    }
  sprintf(SOLUCE(nconfig),SOLUCE(config));
  sprintf(SOLUCD(nconfig),SOLUCD(config));
}
void escriure_configuracio(CONFIG* config)
{
  printf("--------------------\n");
  escriure_angles_robot("Robot esquerra",ROBOTE(config));
  printf("solucio: %s\n",SOLUCE(config));
  escriure_angles_robot("Robot dret",ROBOTD(config));
  printf("solucio: %s\n",SOLUCD(config));
  escriure_posicio_objecte(OBJECTE(config));
  printf("--------------------\n");
}
void desar_configuracio(FILE* fa,CONFIG* config)
{
```

```
   desar_angles_robot(fa,ROBOTE(config));
   desar_angles_robot(fa,ROBOTD(config));
   desar_posicio_objecte(fa,OBJECTE(config));
   fprintf(fa,"n=%d_f=%d_w=%f\n",N_PLAN_LOC(config),F_PLAN_LOC(config),PES(config));
}
void carregar_configuracio(FILE* fa,CONFIG* config)
{
   carregar_angles_robot(fa,ROBOTE(config));
   carregar_angles_robot(fa,ROBOTD(config));
   carregar_posicio_objecte(fa,OBJECTE(config));
   fscanf(fa,"n=%d_f=%d_w=%f\n",&(N_PLAN_LOC(config)),&(F_PLAN_LOC(config)),&(PES(config)));
}

INFO_PLANIF_LOCAL* planificador_local_configi_configf(bool detectar_colisio,
                                                      DADES_CONFIG* dades_config,
                                                      CONFIG* configi,CONFIG* configf)
{
  VECTOR* vector_ini;
  VECTOR* vector;
  VECTOR* vector_landa;
  VECTOR* vector_resta;

  int n,nf;
  bool conf_valida;
  CONFIG* config_act;
  CONF_ROBOT* robot;
  char robot_actiu[MAX_CADENA];

  bool colisionen;

  INFO_PLANIF_LOCAL* planificador_local;


  planificador_local=crear_info_planificador_local();
  robot=crear_robot();
  config_act=crear_configuracio();

  sprintf(robot_actiu,"%s",dades_config->elem_interp);

  if (configi==configf)
    return FALSE;
  else
    {
      vector_ini=crear_vector(6);É
      //s  el vector final
      vector=crear_vector(6);

      if (!strcmp((dades_config->elem_interp),"esquerra"))
        {
          copiar_taula_ini_a_taula_fi(ARTICE(configi),ELEM(vector_ini));
          copiar_taula_ini_a_taula_fi(ARTICE(configf),ELEM(vector));
        }
      if (!strcmp((dades_config->elem_interp),"dret"))
        {
          copiar_taula_ini_a_taula_fi(ARTICD(configi),ELEM(vector_ini));
          copiar_taula_ini_a_taula_fi(ARTICD(configf),ELEM(vector));
        }

      vector_resta=restar_vectors(vector_ini,vector);

      //Vector que uneix el vector inicial i final esquerra
      //i que te longitud g_pas_cami_local
      vector_landa=vector_long_landa(vector_resta,g_pas_cami_local);

      //numero d'iteracions a executar
      nf=ultima_iteracio(vector_ini,vector,g_pas_cami_local);

      //alliberem memoria dels vectors que no utilizarem.
      alliberar_vector(&vector_resta);
      alliberar_vector(&vector);

      n=1;
      conf_valida =TRUE;

      while((n<=nf)&&(conf_valida))
        {
          vector=vector_enessim(n,vector_ini,vector_landa);

          assignar_angles_a_robot(ELEM(vector),robot);
          sprintf(SOLUC(robot),"%s","tot");

          //íAix conservem la óconfiguraci anterior cada vegada.
          copiar_config_a_nova_config(config_act,(planificador_local->config_ant));

          conf_valida=canviar_a_configuracio_donat_robot(detectar_colisio,
                                                         robot_actiu,
                                                         dades_config,
                                                         robot,config_act);

          //alliberem la memoria del vector
          alliberar_vector(&vector);
          n=n+1;
```

```
        }
      colisionen=obtenir_colisio();

      (planificador_local->solucio)=conf_valida;
      (planificador_local->n)=n;

      alliberar_configuracio(&config_act);
      alliberar_robot(&robot);
      return planificador_local;
    }
}
real distancia_entre_configuracions(CONFIG* config1,CONFIG* config2)
{
  real dist_robe;
  real dist_robd;
  int i;
  dist_robe=0.0;
  dist_robd=0.0;

  if (config1==config2)
    return BIG;
  else
    {
      for(i=0;i<=5;i++)
        {
          dist_robe =  dist_robe+quadrat_obertura(ARTICE_i(config1,i),ARTICE_i(config2,i));
          dist_robd =  dist_robd+quadrat_obertura(ARTICD_i(config1,i),ARTICD_i(config2,i));
        }
      return(sqrt(dist_robe+dist_robd));
    }
}
real distancia_fina_entre_configuracions(DADES_CONFIG* dades_config,
                                         CONFIG* configi,CONFIG* configf)
{
  VECTOR* vector_ini;
  VECTOR* vector;
  VECTOR* vector_landa;
  VECTOR* vector_resta;

  int n,nf;
  bool conf_valida_cin;
  real sum_quad=0.0;
  CONFIG* config_act;
  CONFIG* config_prev;
  CONF_ROBOT* robot;
  char robot_actiu[MAX_CADENA];

  robot=crear_robot();

  config_prev=crear_configuracio();
  config_act=crear_configuracio();
  copiar_config_a_nova_config(configi,config_prev);

  sprintf(robot_actiu,"%s",dades_config->elem_interp);

  if (configi==configf)
    sum_quad=BIG;
  else if ((configi==NULL)||(configf==NULL))
    sum_quad=BIG;
  else
    {
      vector_ini=crear_vector(6);
      //es  el vector final
      vector=crear_vector(6);

      copiar_taula_ini_a_taula_fi(ARTICE(configi),ELEM(vector_ini));
      copiar_taula_ini_a_taula_fi(ARTICE(configf),ELEM(vector));

      vector_resta=restar_vectors(vector_ini,vector);

      //vector que uneix el vector inicial i final esquerra
      //i que te longitud g_pas_cami_local
      vector_landa=vector_long_landa(vector_resta,g_pas_cami_local);

      //numero d'iteracions a executar
      nf=ultima_iteracio(vector_ini,vector,g_pas_cami_local);

      //alliberem memoria dels vectors que no
      // utilitzarem mes
      alliberar_vector(&vector_resta);
      alliberar_vector(&vector);

      n=1;
      conf_valida_cin =TRUE;

      while((n<=nf)&&(conf_valida_cin))
        {
          vector=vector_enessim(n,vector_ini,vector_landa);

          assignar_angles_a_robot(ELEM(vector),robot);
          sprintf(SOLUC(robot),"%s","tot");
          conf_valida_cin=canviar_a_configuracio_donat_robot(FALSE,
```

```
                                                                robot_actiu,
                                                                dades_config,
                                                                robot,config_act);

          if (conf_valida_cin)
            sum_quad=distancia_entre_configuracions(config_prev,config_act)+sum_quad;

          //zona de memoria diferent
          alliberar_vector(&vector);

          copiar_config_a_nova_config(config_act,config_prev);
          n=n+1;
        }
      alliberar_configuracio(&config_prev);
      alliberar_configuracio(&config_act);
      alliberar_robot(&robot);
      if (!conf_valida_cin)
        sum_quad=BIG;
    }
  return sum_quad;
}
bool comparar_tipus_solucions_configuracions(CONFIG* config1,CONFIG* config2)
{
  int long_sole1,long_sold1;
  int long_sole2,long_sold2;
  int ne,nd;

  if (config1==config2)
    return FALSE;
  else
    {
    long_sole1=strlen(SOLUCE(config1));
    long_sole2=strlen(SOLUCE(config2));
    long_sold1=strlen(SOLUCD(config1));
    long_sold2=strlen(SOLUCD(config2));

    //comparem si el tipus de solucio per cada una de les configuracio es la mateixa.
    ne=memcmp((void*)SOLUCE(config1),(void*)SOLUCE(config2),
              long_sole1>long_sole2?long_sole1:long_sole2);
    nd=memcmp((void*)SOLUCD(config1),
              (void*)SOLUCD(config2),long_sold1>long_sold2?long_sold1:long_sold2);
    return ((ne==0)&&(nd==0));
    }
}

CONFIG* generar_configuracio_previa_colisio(DADES_CONFIG* dades_config,
                                            CONFIG* config_act)
{
  CONFIG* config;
  bool colisionen;

  INFO_PLANIF_LOCAL* planificador_local;


  int iteracio;
  int cont_no_colisions=0;
  bool generada=FALSE;

  //assigno els valors maxims i minims dels angles del robot, a unes variables auxiliars.

  iteracio=0;
  generada=FALSE;
  colisionen=0;

  config=crear_configuracio();
  do
    {

      canviar_a_configuracio_lliure(FALSE,dades_config,config);

      planificador_local=planificador_local_configi_configf(FALSE,
                                                             dades_config,config_act,config);

      if (planificador_local->solucio)
        {
          alliberar_info_planificador_local(&planificador_local);

          planificador_local= planificador_local_configi_configf(TRUE,dades_config,
                                                                  config_act,config);

          //Si el planificador local fracassa, i la né s major que 2, aleshores
          //la óconfiguraci anterior a la que ha fallat es pot fer servir per ser la generada.
          if ((!(planificador_local->solucio))&&(planificador_local->n>2))
            {
              //printf("fallada_per_colisio\n");
              copiar_config_a_nova_config((planificador_local->config_ant),config);
              generada=TRUE;
            }
          else
            cont_no_colisions=cont_no_colisions+1;
          alliberar_info_planificador_local(&planificador_local);
        }
```

```
        else
          alliberar_info_planificador_local(&planificador_local);


      }while ((!generada)&&(cont_no_colisions<600));

    //printf("cont_no_colisions:%d\n",cont_no_colisions);
    //sum_no_colisions=sum_no_colisions+cont_no_colisions;
    //cont=cont+1;
    //printf("%f\n",(real)sum_no_colisions/(real)cont);


    if (generada)
      return config;
    else
      {
        alliberar_configuracio(&config);
        return NULL;
      }
}
CONFIG* generar_configuracio_robot_aleatori_en_bola_DIST_MAX(DADES_CONFIG* dades_config,
                                                             CONFIG* config_act)
{
  CONFIG* config;
  bool colisionen;

  INFO_PLANIF_LOCAL* planificador_local;

  real aux_g_JMAX1;
  real aux_g_JMIN1;
  real aux_g_JMAX2;
  real aux_g_JMIN2;
  real aux_g_JMAX3;
  real aux_g_JMIN3;
  real aux_g_JMAX4;
  real aux_g_JMIN4;
  real aux_g_JMAX5;
  real aux_g_JMIN5;
  real aux_g_JMAX6;
  real aux_g_JMIN6;

  int iteracio;
  int cont_fora_dist=0;
  int cont_no_local;
  bool generada=FALSE;

  //Assigno els rangs a variables auxiliars.
  aux_g_JMAX1=g_JMAX1;
  aux_g_JMIN1=g_JMIN1;
  aux_g_JMAX2=g_JMAX2;
  aux_g_JMIN2=g_JMIN2;
  aux_g_JMAX3=g_JMAX3;
  aux_g_JMIN3=g_JMIN3;
  aux_g_JMAX4=g_JMAX4;
  aux_g_JMIN4=g_JMIN4;
  aux_g_JMAX5=g_JMAX5;
  aux_g_JMIN5=g_JMIN5;
  aux_g_JMAX6=g_JMAX6;
  aux_g_JMIN6=g_JMIN6;


  iteracio=0;
  generada=FALSE;
  colisionen=0;
  cont_no_local=0;
  config=crear_configuracio();
  do
    {
      canviar_a_configuracio_lliure(FALSE,dades_config,config);

      if(distancia_fina_entre_configuracions(dades_config,config_act,config)<g_MAX_DIST)

        {
          planificador_local= planificador_local_configi_configf(TRUE,
                                                                  dades_config,
                                                                  config_act,config);

          if (!(planificador_local->solucio))
            cont_no_local=cont_no_local+1;
          else
            generada=(planificador_local->solucio);
          alliberar_info_planificador_local(&planificador_local);
        }
      else
        cont_fora_dist=cont_fora_dist+1;

      //quan cauen fora moltes configuracions de la bola DIST_MAX, actualitzo els valors
      //maxim i minim dels angles a un valor mes petit.
      if (cont_fora_dist>g_N_FORA_DIST_MAX)
        {
          cont_fora_dist=0;
```

```
            g_JMAX1=(((ARTICE_i(config_act,0)-aux_g_JMAX1))/((real)g_N_PASOS))+g_JMAX1;
            g_JMIN1=(((ARTICE_i(config_act,0)-aux_g_JMIN1))/((real)g_N_PASOS))+g_JMIN1;
            g_JMAX2=(((ARTICE_i(config_act,1)-aux_g_JMAX2))/((real)g_N_PASOS))+g_JMAX2;
            g_JMIN2=(((ARTICE_i(config_act,1)-aux_g_JMIN2))/((real)g_N_PASOS))+g_JMIN2;
            g_JMAX3=(((ARTICE_i(config_act,2)-aux_g_JMAX3))/((real)g_N_PASOS))+g_JMAX3;
            g_JMIN3=(((ARTICE_i(config_act,2)-aux_g_JMIN3))/((real)g_N_PASOS))+g_JMIN3;
            g_JMAX4=(((ARTICE_i(config_act,3)-aux_g_JMAX4))/((real)g_N_PASOS))+g_JMAX4;
            g_JMIN4=(((ARTICE_i(config_act,3)-aux_g_JMIN4))/((real)g_N_PASOS))+g_JMIN4;
            g_JMAX5=(((ARTICE_i(config_act,4)-aux_g_JMAX5))/((real)g_N_PASOS))+g_JMAX5;
            g_JMIN5=(((ARTICE_i(config_act,4)-aux_g_JMIN5))/((real)g_N_PASOS))+g_JMIN5;
            g_JMAX6=(((ARTICE_i(config_act,5)-aux_g_JMAX6))/((real)g_N_PASOS))+g_JMAX6;
            g_JMIN6=(((ARTICE_i(config_act,5)-aux_g_JMIN6))/((real)g_N_PASOS))+g_JMIN6;


            //subdividire el rang de cada articulacio en g_N_PASOS d'iteracions.
            iteracio=iteracio+1;
            printf("iteracio:%d\n",iteracio);
          }

      }while ((!generada)&&(iteracio<g_N_PASOS));

    g_JMAX1=aux_g_JMAX1;
    g_JMIN1=aux_g_JMIN1;
    g_JMAX2=aux_g_JMAX2;
    g_JMIN2=aux_g_JMIN2;
    g_JMAX3=aux_g_JMAX3;
    g_JMIN3=aux_g_JMIN3;
    g_JMAX4=aux_g_JMAX4;
    g_JMIN4=aux_g_JMIN4;
    g_JMAX5=aux_g_JMAX5;
    g_JMIN5=aux_g_JMIN5;
    g_JMAX6=aux_g_JMAX6;
    g_JMIN6=aux_g_JMIN6;

    //printf("planificador_local:%d\n",planificador_local);
    //printf("iteracio:%d\n",iteracio);
    if (generada)
      return config;
    else
      {
        alliberar_configuracio(&config);
        return NULL;
      }
}


CONFIG* generar_configuracio_objecte_aleatori_en_bola_DIST_MAX(char* elem_interp,
                                                               PDH pdh,JNTS_RANGE* jr,
                                                               TRANSF_CTNT* T,char* conf_rob_esq,
                                                               char* conf_rob_dret,CONFIG* config_act)
{
  CONFIG* config;
  config=crear_configuracio();
  return config;
}
```

H.15 robots.h

```
#ifndef ROBOTS_H
#define ROBOTS_H

/////////////////////////////////////////////////////
/// @file robots.h
/// @brief Les rutines que treballen amb els robots.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       Implementa les rutines que treballen amb el
///       tipus CONF_ROBOT, que écont una óconfiguraci d'un
///       robot especificada com els sis angles
///       que el descriuen (camp artic) i el seu tipus de ósoluci
/// à      cinemtica (camp sol).
/////////////////////////////////////////////////////

#include "funcions_generals.h"
#include "transf_homogenies.h"
#include "macros_generals.h"
#include "globals.h"
#include "webots_interface.h"

typedef real ARTIC;

typedef struct conf_robot
{
  ARTIC* artic;
  char* sol;
}CONF_ROBOT;

#define ARTIC(robot) ((robot)->artic)
#define ARTIC_i(robot,i) (((robot)->artic)[i])
```

```
#define SOLUC(robot) ((robot)->sol)

//operacions constructores

/////////////////////////////////////////////////////
///  @brief óCreaci d'una óconfiguraci pel robot.
///
///  Reserva òmemria per una robot, que consta de
///  sis valors angular i del tipus de ósoluci
/// à cinemtica.
///
///  @return  Una variable tipus ROBOT.
/////////////////////////////////////////////////////

CONF_ROBOT* crear_robot();

/////////////////////////////////////////////////////
///  @brief Allibera la óinformaci d'un robot.
///
///  Allibera la òmemria de la variable robot,
///  que consta de sis valors angulars pel robot esquerra, sis éms pel
///  dret i el vector d'óorientaci i de ótranslaci de l'objecte.
///
///  @param robot Par. ent/sort. óConfiguraci del robot
///               que volem alliberar.
/////////////////////////////////////////////////////

void alliberar_robot(CONF_ROBOT** robot);

/////////////////////////////////////////////////////
///  @brief Assignar els angles a un robot.
///
///  Afegeix uns angles donats a la variable
///  que écont una óconfiguraci d'un robot.
///
///  @param artic Par. ent. éCont un vector de sis angles.
///
///  @param robot Par. ent/sort. Variable del robot
///               que li assignen els angles donats.
///
/////////////////////////////////////////////////////

void assignar_angles_a_robot(ARTIC* artic,CONF_ROBOT* robot);

/////////////////////////////////////////////////////
///  @brief  Visualitzar una óconfiguraci del robot.
///
///  Visualitza en Webots pel robot que desitgem, esquerra o
///  dret, els angles continguts en la variable robot.
///
///  @param nom_robot Par. ent. Nom del robot que volem moure.
///
///  @param robot Par. ent. Variable que écont
///               una óconfiguraci d'un robot.
///
/////////////////////////////////////////////////////

void canviar_angles_robot (char* nom_robot,CONF_ROBOT* robot);

/////////////////////////////////////////////////////
///  @brief Canvi de èreferncia dels angles.
///
///  Transforma els angles de la èreferncia de
///  l'article desenvolupat per Paul a angles
///  aplicables al RX60.
///
///  @param robot Par. ent/sort.óConfiguraci d'un robot.
///
/////////////////////////////////////////////////////

void angles_robot_ref_paper_ref_RX60(CONF_ROBOT* robot);

/////////////////////////////////////////////////////
///  @brief Canvi de èreferncia dels angles.
///
///  Transforma els angles de la èreferncia
///  del RX60 als de l'article d'en Paul.
///
///  @param robot Par. ent/sort. óConfiguraci d'un robot.
/////////////////////////////////////////////////////

void angles_robot_ref_RX60_ref_paper(CONF_ROBOT* robot);

/////////////////////////////////////////////////////
///  @brief éObt una óconfiguraci òaleatria del robot.
///
///  Troba un vector de sis angles aleatoris que
///  que emmagatzema al camp artic del la variable robot.
///
///  @param robot Par. ent/sort. éCont la óconfiguraci
/// ò              aleatria d'un robot.
/////////////////////////////////////////////////////
```

```
void angles_robot_aleatoris(CONF_ROBOT* robot);

////////////////////////////////////////////////////
///  @brief Guarda els angles actuals d'un robot.
///
///  Guarda els angles actuals del robot que considerem en la
///  taula angle_rob_esq o angle_rob_dret
///  de  webots_interface.h. Es fa íaix èperqu Webots
/// é noms permet emmagatzemar la óinformaci dels elements
///  de l'escena de treball en la mateixa zona de òmemria per
///  tota una ósimulaci.
///
///  @param nom_robot Par. ent. Nom del robot que volem moure
/// ò                  aleatriament.
////////////////////////////////////////////////////

void obtenir_angles_robot (char* nom_robot);

////////////////////////////////////////////////////
///  @brief Copia els angles actuals del robot a una altre variable.
///
///  Copia la óconfiguraci obtinguda en les variables
///  de Webots, en una nova variable tipus CONFIG_ROBOT.
///
///  @param nom_robot Par. ent. Nom del robot que volem recuperar els angles.
///
///  @param robot Par. ent/sort. óConfiguraci copiada.
////////////////////////////////////////////////////

void copiar_obtencio_angles_a_robot(char* nom_robot,CONF_ROBOT* robot);

////////////////////////////////////////////////////
///  @brief àTrasps d'una óconfiguraci d'un robot a una altra.
///
///  Copia la óinformaci emmagatzemada en una óconfiguraci
///  d'un robot a una altra variable del tipus C0NF_ROBOT.
///
///  @param robot Par. ent. óConfiguraci donada del robot.
///
///  @param nrobot  Par. ent/sort. óConfiguraci on fem
///                 el àtrasps.
////////////////////////////////////////////////////

void copiar_robot_a_nou_robot(CONF_ROBOT* robot,CONF_ROBOT* nrobot);

////////////////////////////////////////////////////
///  @brief Resol el problema àcinemtic directe pel robot.
///
///  Resol el problema directe del robot RX60.
///
///  @param pdh Par. ent. éCont els àparmetres de Denavit-Hartenberg.
///
///  @param robot Par.ent. éCont els angles dada del robot.
///
///  @param Mr Par. ent/sortÉ. s la matriu de ótransformaci directe a trobar.
///
///  @param tipus_conf Par. ent/sortÉ. s el tipus de óconfiguraci
/// à                   cinemtica que
///                    trobem dels angles donats.
////////////////////////////////////////////////////

void cinematica_directa_RX60 (PDH pdh,CONF_ROBOT* robot,TRANSF_HOMOG Mr,char* tipus_conf);

////////////////////////////////////////////////////
///  @brief Resol el problema àcinemtic invers pel robot.
///
///  Resol el problema invers del robot RX60.
///
///  @param pdh Par. ent. éCont els àparmetres de Denavit-Hartenberg.
///
///  @param jr Par. ent. éCont els rangs de les articulacions.
///
///  @param Mr Par. ent. Matriu de ótransformaci donada.
///
///  @param tipus_conf Par. en. É s el tipus de óconfiguraci àcinemtica que
///                             volem determinar.
///  @param robot Par. ent/sort àContindr els angles que volem trobar.
////////////////////////////////////////////////////

bool cinematica_inversa_RX60 (PDH pdh,JNTS_RANGE* jr,TRANSF_HOMOG Mr,char* tipus_conf,CONF_ROBOT* robot);

////////////////////////////////////////////////////
///  @brief Comprova si els angles del robot estan en rang.
///
///  Comprova si els angles del robot estan dintre el rang
///  de les articulacions.
///
///  @param robot Par. ent. éCont la óconfiguraci d'un robot.
///
///  @param jr Par. ent. éCont els rangs de les articulacions.
///
```

```
///  @return Si els angles del robot estan en rang.
/////////////////////////////////////////////////////////

bool angles_robot_ref_RX60_en_rang (CONF_ROBOT* robot,JNTS_RANGE* jr);

//Operacions consultores

/////////////////////////////////////////////////////////
///  @brief Escriu els angles del robot per pantalla.
///
///  Escriu els angles del robot per pantalla.
///
///  @param nom_robot Par. ent. Nom del robot.
///
///  @param robot Par. ent. éCont els angles que escriurem.
/////////////////////////////////////////////////////////

void escriure_angles_robot(char* nom_robot,const CONF_ROBOT* robot);

/////////////////////////////////////////////////////////
///  @brief Desa els angles del robot en un fitxer.
///
///  Desa els angles del robot en un fitxer.
///
///  @param fa Par. ent.  Nom del fitxer a on desem la óconfiguraci.
///
///  @param robot Par. ent. éCont els angles que que volem desar.
/////////////////////////////////////////////////////////

void desar_angles_robot(FILE* fa,CONF_ROBOT* robot);

/////////////////////////////////////////////////////////
///  @brief Carrega els angles del robot des d' un fitxer.
///
///  Carrega els angles del robot des d'un fitxer.
///
///  @param fa Par. ent. Fitxer des d'on carreguem
///            la óconfiguraci del robot.
///
///  @param robot Par. ent/sort. éCont els angles del fitxer.
/////////////////////////////////////////////////////////

void carregar_angles_robot(FILE* fa,CONF_ROBOT* robot);

/////////////////////////////////////////////////////////
///  @brief Desa les configuracions de dos robots.
///
///  Desa les configuracions de dos robots.
///
///  @param ftraj Fitxer a on desem
///               les configuracions dels dos robots.
///
///  @param robote éCont la óconfiguraci d'un robot.
///
///  @param robotd éCont la óconfiguraci d'un robot.
/////////////////////////////////////////////////////////

void desar_angles_robots(FILE* ftraj,CONF_ROBOT* robote,CONF_ROBOT* robotd);

#endif
```

H.16 robots.c

```
#include "robots.h"

/////////////////////////////////////////////////////////
//Rutines privades a configuracio.c
/////////////////////////////////////////////////////////
static void convertir_JNTS_a_CONF_ROBOT(JNTS* j,CONF_ROBOT* robot);
static void convertir_CONF_ROBOT_a_JNTS(CONF_ROBOT* robot,JNTS* j);

CONF_ROBOT* crear_robot()
{
  CONF_ROBOT* robot;
  int i;

  if(NEW(robot,1,CONF_ROBOT)==NULL)
    cerror("error de malloc en crear_robot()\n");
  if(NEW(ARTIC(robot),6,ARTIC)==NULL)
    cerror("error de malloc en crear_robot()\n");
  if(NEW(SOLUC(robot),4,char)==NULL)
    cerror("error de malloc en crear_robot()\n");
  for(i=0;i<=5;i++)
    ARTIC_i(robot,i)=VAL_INICIAL;
  sprintf(SOLUC(robot),"xxx");

  return robot;
}
void assignar_angles_a_robot(ARTIC* artic,CONF_ROBOT* robot)
{
  int i;
```

```
  for(i=0;i<=5;i++)
    ARTIC_i(robot,i)=artic[i];
}
void alliberar_robot(CONF_ROBOT** robot)
{
  free(ARTIC(*robot));
  ARTIC(*robot)=NULL;
  free(SOLUC(*robot));
  SOLUC(*robot)=NULL;
  free(*robot);
  *robot=NULL;
}
void canviar_angles_robot (char* nom_robot,CONF_ROBOT* robot)
{
  if (!strcmp(nom_robot,"esquerra"))
    {
      canviar_angle_solid (ART_ESQ_1,&(ARTIC_i(robot,0)));
      canviar_angle_solid (ART_ESQ_2,&(ARTIC_i(robot,1)));
      canviar_angle_solid (ART_ESQ_3,&(ARTIC_i(robot,2)));
      canviar_angle_solid (ART_ESQ_4,&(ARTIC_i(robot,3)));
      canviar_angle_solid (ART_ESQ_5,&(ARTIC_i(robot,4)));
      canviar_angle_solid (ART_ESQ_6,&(ARTIC_i(robot,5)));
    }

  if (!strcmp(nom_robot,"dret"))
    {
      canviar_angle_solid (ART_DRET_1,&(ARTIC_i(robot,0)));
      canviar_angle_solid (ART_DRET_2,&(ARTIC_i(robot,1)));
      canviar_angle_solid (ART_DRET_3,&(ARTIC_i(robot,2)));
      canviar_angle_solid (ART_DRET_4,&(ARTIC_i(robot,3)));
      canviar_angle_solid (ART_DRET_5,&(ARTIC_i(robot,4)));
      canviar_angle_solid (ART_DRET_6,&(ARTIC_i(robot,5)));

    }
}
void angles_robot_ref_paper_ref_RX60(CONF_ROBOT* robot)
{
  int i;
  ARTIC_i(robot,0)=(ARTIC_i(robot,0)-JOFST1);
  ARTIC_i(robot,1)=(ARTIC_i(robot,1)-JOFST2);
  ARTIC_i(robot,2)=(ARTIC_i(robot,2)-JOFST3);

  /* canvi dels valors degut al sentit positiu d'alguns eixos del robot Staubli*/

  ARTIC_i(robot,1)=-(ARTIC_i(robot,1));
  ARTIC_i(robot,2)=-(ARTIC_i(robot,2));
  ARTIC_i(robot,4)=-(ARTIC_i(robot,4));

  /* passar els angles de la nova referencia de PI a -PI */

  for (i=0;i<=5;i++)
    ARTIC_i(robot,i)=convertir_angle_real_a_pi_pi(ARTIC_i(robot,i));
}
void angles_robot_ref_RX60_ref_paper(CONF_ROBOT* robot)
{
  ARTIC_i(robot,0)=(ARTIC_i(robot,0)+JOFST1);
  ARTIC_i(robot,1)=(-(ARTIC_i(robot,1))+JOFST2);
  ARTIC_i(robot,2)=(-(ARTIC_i(robot,2))+JOFST3);
  ARTIC_i(robot,4)=-(ARTIC_i(robot,4));
}


void angles_robot_aleatoris(CONF_ROBOT* robot)
{
  ARTIC_i(robot,0)=valor_aleatori((real)g_JMIN1,(real)g_JMAX1);
  ARTIC_i(robot,1)=valor_aleatori((real)g_JMIN2,(real)g_JMAX2);
  ARTIC_i(robot,2)=valor_aleatori((real)g_JMIN3,(real)g_JMAX3);
  ARTIC_i(robot,3)=valor_aleatori((real)g_JMIN4,(real)g_JMAX4);
  ARTIC_i(robot,4)=valor_aleatori((real)g_JMIN5,(real)g_JMAX5);
  ARTIC_i(robot,5)=valor_aleatori((real)g_JMIN6,(real)g_JMAX6);

  //Si mostrejo tots els rangs de treball  dels angles del robot
  //podem trobar qualsevol tipus de solucio de configuracio.
  sprintf(SOLUC(robot),"%s","tot");
}
void obtenir_angles_robot (char* nom_robot)
{
  if (!strcmp(nom_robot,"esquerra"))
    {
      obtenir_angle_solid (ART_ESQ_1,&(angle_rob_esq[0]));
      obtenir_angle_solid (ART_ESQ_2,&(angle_rob_esq[1]));
      obtenir_angle_solid (ART_ESQ_3,&(angle_rob_esq[2]));
      obtenir_angle_solid (ART_ESQ_4,&(angle_rob_esq[3]));
      obtenir_angle_solid (ART_ESQ_5,&(angle_rob_esq[4]));
      obtenir_angle_solid (ART_ESQ_6,&(angle_rob_esq[5]));
    }
  if (!strcmp(nom_robot,"dret"))
    {
      obtenir_angle_solid (ART_DRET_1,&(angle_rob_dret[0]));
      obtenir_angle_solid (ART_DRET_2,&(angle_rob_dret[1]));
      obtenir_angle_solid (ART_DRET_3,&(angle_rob_dret[2]));
```

```
      obtenir_angle_solid (ART_DRET_4,&(angle_rob_dret[3]));
      obtenir_angle_solid (ART_DRET_5,&(angle_rob_dret[4]));
      obtenir_angle_solid (ART_DRET_6,&(angle_rob_dret[5]));
    }
}
void copiar_obtencio_angles_a_robot(char* nom_robot,CONF_ROBOT* robot)
{
  int i;

  if (!strcmp(nom_robot,"esquerra"))
    for(i=0;i<=5;i++)
      ARTIC_i(robot,i)=angle_rob_esq[i];

  if (!strcmp(nom_robot,"dret"))
    for(i=0;i<=5;i++)
      ARTIC_i(robot,i)=angle_rob_dret[i];

}
void copiar_robot_a_nou_robot(CONF_ROBOT* robot,CONF_ROBOT* nrobot)
{
  int i;

  for(i=0;i<=5;i++)
    ARTIC_i(nrobot,i)=ARTIC_i(robot,i);
  sprintf(SOLUC(nrobot),"%s",SOLUC(robot));
}

void convertir_JNTS_a_CONF_ROBOT(JNTS* j,CONF_ROBOT* robot)
{
  ARTIC_i(robot,0)=(real)j->th1;
  ARTIC_i(robot,1)=(real)j->th2;
  ARTIC_i(robot,2)=(real)j->th3;
  ARTIC_i(robot,3)=(real)j->th4;
  ARTIC_i(robot,4)=(real)j->th5;
  ARTIC_i(robot,5)=(real)j->th6;
}
void convertir_CONF_ROBOT_a_JNTS(CONF_ROBOT* robot,JNTS* j)
{
  j->th1=(Greal)ARTIC_i(robot,0);
  j->th2=(Greal)ARTIC_i(robot,1);
  j->th3=(Greal)ARTIC_i(robot,2);
  j->th4=(Greal)ARTIC_i(robot,3);
  j->th5=(Greal)ARTIC_i(robot,4);
  j->th6=(Greal)ARTIC_i(robot,5);
}

void cinematica_directa_RX60 (PDH pdh,CONF_ROBOT* robot,TRANSF_HOMOG Mr,char* tipus_conf)
{
  JNTS j;
  TRSF tr;

  convertir_CONF_ROBOT_a_JNTS(robot,&j);
  jns_to_tr_n(pdh,&j,&tr,tipus_conf);
  convertir_TRSF_a_TRANSF_HOMOG(&tr,Mr);
}

bool cinematica_inversa_RX60 (PDH pdh,JNTS_RANGE* jr,TRANSF_HOMOG Mr,char* tipus_conf,CONF_ROBOT* robot)
{
  JNTS j;
  TRSF tr;
  int code;

  // Inicialitzem j
  j.th1 = -1000.;  j.th2 = -1000.; j.th3 = -1000.;
  j.th4 = -1000.;  j.th5 = -1000.; j.th6 = -1000.;

  //Pasar la transformacio homogenia del tipus TRANSF_HOMOG a TRSF
  convertir_TRANSF_HOMOG_a_TRSF(Mr,&tr);

  //Crida a la àcinemtica inversa
  code=tr_to_jns_n(&tr,pdh,tipus_conf,&j,jr);

  //Guardem al robot el tipus de solucio triada de la cinematica inversa.
  sprintf(SOLUC(robot),tipus_conf);

  //printf("------->Crida explain_code\n");
  //explain_code(code);
  //printf("code:%d\n",code);
  //printf("------->%f %f %f %f %f %f\n", j.th1, j.th2, j.th3, j.th4, j.th5, j.th6);

  //Pasar els angles del tipus JNTS al tipus robot.
  convertir_JNTS_a_CONF_ROBOT(&j,robot);
  return(code==0||code==256);
}
bool angles_robot_ref_RX60_en_rang (CONF_ROBOT* robot,JNTS_RANGE* jr)

{
  int code=0;

  if (!angle_en_rang_pi_pi((Greal)ARTIC_i(robot,0),(Greal)(jr->jmin_c.th1),(Greal)(jr->jmax_c.th1)))
    code |= 01;
  if (!angle_en_rang_pi_pi((Greal)ARTIC_i(robot,1),(Greal)(jr->jmin_c.th2),(Greal)(jr->jmax_c.th2)))
```

```
    code |= 02;
  if (!angle_en_rang_pi_pi((Greal)ARTIC_i(robot,2),(Greal)(jr->jmin_c.th3),(Greal)(jr->jmax_c.th3)))
    code |= 04;
  if (!angle_en_rang_pi_pi((Greal)ARTIC_i(robot,3),(Greal)(jr->jmin_c.th4),(Greal)(jr->jmax_c.th4)))
    code |= 010;
  if (!angle_en_rang_pi_pi((Greal)ARTIC_i(robot,4),(Greal)(jr->jmin_c.th5),(Greal)(jr->jmax_c.th5)))
    code |= 020;
  if (!angle_en_rang_pi_pi((Greal)ARTIC_i(robot,5),(Greal)(jr->jmin_c.th6),(Greal)(jr->jmax_c.th6)))
    code |= 040;
  //explain_code(code);

  //Si la configuracio àest en el rang
  return ((code==0)||(code==256));
}
void escriure_angles_robot(char* nom_robot,const CONF_ROBOT* robot)
{
  int i;
  printf("%s\n",nom_robot);
  for(i=0;i<=5;i++)
    printf("%f ",ARTIC_i(robot,i));
  printf("\n");
}
void desar_angles_robot(FILE* fa,CONF_ROBOT* robot)
{
  int i;

  fprintf(fa,"%s\n",SOLUC(robot));
  for(i=0;i<=5;i++)
    fprintf(fa,"%f ",ARTIC_i(robot,i));
  fprintf(fa,"\n");

}
void carregar_angles_robot(FILE* fa,CONF_ROBOT* robot)
{
  int i;

  fscanf(fa,"%s\n",SOLUC(robot));
  for(i=0;i<=5;i++)
    fscanf(fa,"%f ",&(ARTIC_i(robot,i)));
  fprintf(fa,"\n");
}
void desar_angles_robots(FILE* ftraj,CONF_ROBOT* robote,CONF_ROBOT* robotd)
{
  int i;
  for(i=0;i<=5;i++)
    fprintf(ftraj,"%f ",ARTIC_i(robote,i)*RADTODEG);
  fprintf(ftraj,"\n");
  for(i=0;i<=5;i++)
    fprintf(ftraj,"%f ",ARTIC_i(robotd,i)*RADTODEG);
  fprintf(ftraj,"\n");
}
```

H.17 objecte.h

```
#ifndef OBJECTE_H
#define OBJECTE_H

////////////////////////////////////////////////////
/// @file objecte.h
/// @brief Les rutines que treballen amb l'objecte.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       Implementem les rutines que treballen amb el
///       tipus CONF_OBJECTE, que écont una óconfiguraci d'un
///       objecte especificada com els sis angles
///       que el descriuen (camp artic) i el seu tipus de ósoluci
/// à      cinemtica (camp sol).
////////////////////////////////////////////////////

#include "globals.h"
#include "funcions_generals.h"
#include "transf_homogenies.h"
#include "macros_generals.h"
#include "webots_interface.h"

typedef real TRANSL;
typedef real ROT;

typedef struct conf_obj
{
  TRANSL* transl;
  ROT* rot;
}CONF_OBJECTE;

//les macros les podem considerar operacions
//consultores
#define TRANSL(objecte) ((objecte)->transl)
#define TRANSL_i(objecte,i) (((objecte)->transl)[i])
#define ROT(objecte) ((objecte)->rot)
```

```
#define ROT_i(objecte,i) (((objecte)->rot)[i])

//operacions constructores


/////////////////////////////////////////////////////
///  @brief óCreaci d'una óconfiguraci de l'objecte.
///
///  Reserva òmemria per una óconfiguraci del objecte,
///  que consta de un vector de ótranslaci i un
///  vector de órotaci.
///
///  @return  Un punter al tipus CONF_OBJECTE.
/////////////////////////////////////////////////////

CONF_OBJECTE* crear_posicio_objecte();

/////////////////////////////////////////////////////
///  @brief óAlliberaci d'una óconfiguraci de l'objecte.
///
///  @param objecte Par. ent/sort. óConfiguraci de
///                 l'objecte que volem alliberar.
///
/////////////////////////////////////////////////////

void alliberar_posicio_objecte(CONF_OBJECTE** objecte);

/////////////////////////////////////////////////////
///  @brief Copia la óinformaci d'un objecte a un altre.
///
///  Copia la óinformaci d'un objecte a un altre.
///
///  @param objecte Par. ent. óConfiguraci de l'objecte.
///
///  @param nobjecte Par. ent/sort. óConfiguraci de l'objecte.
///
/////////////////////////////////////////////////////

void copiar_objecte_a_nou_objecte(CONF_OBJECTE* objecte,CONF_OBJECTE* nobjecte);

/////////////////////////////////////////////////////
///  @brief  Visualitzar una óconfiguraci de l'objecte.
///
///  Visualitza en Webots la óposici i óorientaci de
///  l'objecte a partir de la variable objecte.
///
///  @param objecte Par. ent. óConfiguraci de l'objecte.
/////////////////////////////////////////////////////

void canviar_posicio_objecte(CONF_OBJECTE* objecte);

/////////////////////////////////////////////////////
///  @brief  Calcula una óconfiguraci òaleatria per l'objecte.
///
///  Mostreja aletoriament el vector de ótranslaci i
/// ó orientaci de l'objecte obtenint íaix una
/// ó configuraci òaleatria del mateix.
///
///  @param objecte Par. ent/sort. óConfiguraci òaleatria
///                 de l'objecte.
/////////////////////////////////////////////////////

void posicio_objecte_aleatoria (CONF_OBJECTE* objecte);

/////////////////////////////////////////////////////
///  @brief é Obt la óposici i óorientaci de l'objecte.
///
///  Guarda el vector de ótranslaci i óorientaci
///  actuals de l'objecte en les taules transl_objecte
///  i rot_objecte. Es fa íaix èperqu Webots énoms
///  permet emmagatzemar la óinformaci dels elements
///  de l'escena de treball en la mateixa zona de òmemria per
///  tota una ósimulaci.
///
/////////////////////////////////////////////////////

void obtenir_posicio_objecte ();

/////////////////////////////////////////////////////
///  @brief  Copia la óconfiguraci actual de l'objecte
///          a una altra variable.
///
///  Copia la óconfiguraci obtinguda en les variables
///  de Webots de l'objecte, en una nova variable
///  tipus CONFIG_ROBOT.
///
///  @param objecte Par. ent/sort. óConfiguraci de l'objecte
///                 copia.
///
/////////////////////////////////////////////////////

void copiar_obtencio_posicio_a_objecte(CONF_OBJECTE* objecte);
```

```
////////////////////////////////////////////////////
///  @brief Resol el problema àcinemtic directe per l'objecte.
///
///  Resol el problema àcinemtic directe per l'objecte.
/// É s a dir, a partir de la óposici i óorientaci de l'objecte
///  troba  la matriu de ótransformaci que el descriu.
///
///  @param objecte Par. ent. Punter a
///                 una óconfiguraci de l'objecte.
///
///  @param Mo Par. ent/sort. óTransformaci èhomognia que volem
///                 trobar.
///
////////////////////////////////////////////////////

void cinematica_directa_objecte (CONF_OBJECTE* objecte,TRANSF_HOMOG Mo);

////////////////////////////////////////////////////
///  @brief Resol el problema àcinemtic invers per l'objecte.
///
///  Resol el problema àcinemtic invers per l'objecte.
/// É s a dir, a partir de la matriu de ótransformaci
///  de l'objecte troba la óposici i óorientaci
///  del mateix.
///
///  @param Mo  Par. ent. óTransformaci èhomognia donada.
///
///  @param objecte Par.ent/sort. óConfiguraci de l'objecte.
///
////////////////////////////////////////////////////

void cinematica_inversa_objecte (TRANSF_HOMOG Mo,CONF_OBJECTE* objecte);

//Operacions consultores

////////////////////////////////////////////////////
///  @brief Escriure per pantalla la óposici i óorientaci
///         de l'objecte.
///
///  Escriu pel canal àestndard de sortida el vector
///  de óposici i óorientacic de l'objecte.
///
///  @param objecte Par. ent. óConfiguraci de l'objecte.
///
////////////////////////////////////////////////////

void escriure_posicio_objecte(const CONF_OBJECTE* objecte);

////////////////////////////////////////////////////
///  @brief Desa en un fitxer la óposici i óorientaci
///         de l'objecte.
///
///  Desa en un fitxer el vector
///  de óposici i óorientaci de la variable objecte.
///
///  @param fa Par. sort. Fitxer que llegim
///            per escritura.
///
///  @param objecte Par. ent. óConfiguraci de l'objecte
///                 que desarem en el fitxer.
///
////////////////////////////////////////////////////

void desar_posicio_objecte(FILE* fa,CONF_OBJECTE* objecte);

////////////////////////////////////////////////////
///  @brief Carrega d'un fitxer la óposici i óorientaci
///         de l'objecte.
///
///  Carrega en la variable objecte la óconfiguraci
///  d'un objecte que àest en el fitxer fa.
///
///  @param fa Par. ent. Fitxer que llegim
///            per lectura.
///
///  @param objecte Par. ent/sort. Punter a una óconfiguraci
///                              de l'objecte.
///
////////////////////////////////////////////////////

void carregar_posicio_objecte(FILE* fa,CONF_OBJECTE* objecte);

#endif
```

H.18 objecte.c

```
#include "objecte.h"


CONF_OBJECTE* crear_posicio_objecte()
```

```
{
  CONF_OBJECTE* objecte;
  int i;

  if(NEW(objecte,1,CONF_OBJECTE)==NULL)
    cerror("error de malloc en crear posicio objecte()\n");
  if(NEW(TRANSL(objecte),3,TRANSL)==NULL)
    cerror("error de malloc en crear posicio objecte()\n");
  if(NEW(ROT(objecte),3,ROT)==NULL)
    cerror("error de malloc en crear posicio objecte()\n");

  for(i=0;i<=2;i++)
    {
      TRANSL_i(objecte,i)=VAL_INICIAL;
      ROT_i(objecte,i)=VAL_INICIAL;
    }

  return objecte;

}
void alliberar_posicio_objecte(CONF_OBJECTE** objecte)
{
  free(TRANSL(*objecte));
  free(ROT(*objecte));
  TRANSL(*objecte)=NULL;
  ROT(*objecte)=NULL;
  free(*objecte);
  *objecte=NULL;
}
void copiar_objecte_a_nou_objecte(CONF_OBJECTE* objecte,CONF_OBJECTE* nobjecte)
{
  int i;

  for(i=0;i<=3;i++)
    {
      TRANSL_i(nobjecte,i)=TRANSL_i(objecte,i);
      ROT_i(nobjecte,i)=ROT_i(objecte,i);
    }
}
void canviar_posicio_objecte(CONF_OBJECTE* objecte)
{
  aplicar_translacio_solid(TRANSL_OBJECTE,TRANSL(objecte));
  canviar_angle_solid (ROT_X_OBJECTE,&(ROT_i(objecte,0)));
  canviar_angle_solid (ROT_Y_OBJECTE,&(ROT_i(objecte,1)));
  canviar_angle_solid (ROT_Z_OBJECTE,&(ROT_i(objecte,2)));
}
void posicio_objecte_aleatoria (CONF_OBJECTE* objecte)
{
  TRANSL_i(objecte,0)=(real)valor_aleatori(g_XMIN,g_XMAX);
  TRANSL_i(objecte,1)=(real)valor_aleatori(g_YMIN,g_YMAX);
  TRANSL_i(objecte,2)=(real)valor_aleatori(g_ZMIN,g_ZMAX);

  ROT_i(objecte,0)=(real)valor_aleatori(g_ROT_XMIN,g_ROT_XMAX);
  ROT_i(objecte,1)=(real)valor_aleatori(g_ROT_YMIN,g_ROT_YMAX);
  ROT_i(objecte,2)=(real)valor_aleatori(g_ROT_ZMIN,g_ROT_ZMAX);
}
void obtenir_posicio_objecte()
{
  obtenir_translacio_solid(TRANSL_OBJECTE,transl_objecte);
  obtenir_angle_solid (ROT_X_OBJECTE,&(rot_objecte[0]));
  obtenir_angle_solid (ROT_Y_OBJECTE,&(rot_objecte[1]));
  obtenir_angle_solid (ROT_Z_OBJECTE,&(rot_objecte[2]));
}
void copiar_obtencio_posicio_a_objecte(CONF_OBJECTE* objecte)
{
  int i;

  for(i=0;i<=2;i++)
    TRANSL_i(objecte,i)=transl_objecte[i];
  for(i=0;i<=2;i++)
    ROT_i(objecte,i)=rot_objecte[i];
}

void cinematica_directa_objecte (CONF_OBJECTE* objecte,TRANSF_HOMOG Mo)
{
  TRANSF_HOMOG M1,M2;
  /* real transl_objecte[3]; */

/*   transl_objecte[0]=(objecte->transl)[0]; */
/*   transl_objecte[1]=(objecte->transl)[1]; */
/*   transl_objecte[2]=(objecte->transl)[2]; */

  transf_homogenia_translacio(M1,(objecte->transl));
  transf_homogenia_rot_xyz(M2,((objecte->rot)[0]),((objecte->rot)[1]),((objecte->rot)[2]));
  producte_transf_homogenies(M1,M2,Mo);
}
void cinematica_inversa_objecte (TRANSF_HOMOG Mo,CONF_OBJECTE* objecte)
{
  real ex1,ex2,a;
  real sig1,sig2;

  (objecte->transl)[0]=Mo[0][3];
```

```
  (objecte->transl)[1]=Mo[1][3];
  (objecte->transl)[2]=Mo[2][3];

  if (fabs(Mo[0][2])==1)
    {
      (objecte->rot)[1]=asin(Mo[0][2]);
      sig1 =((Mo[0][0])>0.0) ? 1.: -1.;
      ex1=atan2((sig1*(Mo[1][0])),Mo[1][1]);
      ex2=atan2(Mo[2][1],(-sig1*(Mo[2][0])));
      /* si el sistema es compatible */
      if (ex1==ex2)
        {
          (objecte->rot)[2]=0;
          (objecte->rot)[0]=-sig1*(objecte->rot)[2]+ex1;


        }

    }
  else
    {
      (objecte->rot)[0]=PI-atan2(Mo[1][2],Mo[2][2]);
      (objecte->rot)[2]=PI-atan2(Mo[0][1],Mo[0][0]);
      sig1 =((Mo[0][0])>0.0) ? 1.: -1.;
      sig2=(cos((objecte->rot)[2])>0)? 1.:-1.;
      a=(pow(Mo[1][2],2.0)+pow(Mo[2][2],2.0));
      a=sig1*sig2*sqrt(a);
      (objecte->rot)[1]=atan2(Mo[0][2],a);
    }
  (objecte->rot)[0]=convertir_angle_real_a_pi_pi((objecte->rot)[0]);
  (objecte->rot)[1]=convertir_angle_real_a_pi_pi((objecte->rot)[1]);
  (objecte->rot)[2]=convertir_angle_real_a_pi_pi((objecte->rot)[2]);
}
void escriure_posicio_objecte(const CONF_OBJECTE* objecte)
{
  int i;
  printf("Translacio objecte\n");
  for(i=0;i<=2;i++)
    printf("%f ",TRANSL_i(objecte,i));
  printf("\n");
  printf("Rotacio objecte\n");
  for(i=0;i<=2;i++)
    printf("%f ",ROT_i(objecte,i));
  printf("\n");
}
void desar_posicio_objecte(FILE* fa,CONF_OBJECTE* objecte)
{
  int i;

  for(i=0;i<=2;i++)
    fprintf(fa,"%f ",TRANSL_i(objecte,i));
  for(i=0;i<=2;i++)
    fprintf(fa,"%f ",ROT_i(objecte,i));
  fprintf(fa,"\n");

}
void carregar_posicio_objecte(FILE* fa,CONF_OBJECTE* objecte)
{
  int i;

  for(i=0;i<=2;i++)
    fscanf(fa,"%f ",&(TRANSL_i(objecte,i)));
  for(i=0;i<=2;i++)
    fscanf(fa,"%f ",&(ROT_i(objecte,i)));
  fscanf(fa,"\n");

}
```

H.19 lib_colisions.c

```
#define _MAIN_COL_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ode/ode.h>
#include "tau_colisions.h"
#include <physics/physics.h>
#include "windows.c" /* necessary to make the shared library work under windows */

#define FITXER_NO_COL "/home/gbonals/Haydn/prc/exm/parelles_no_colisio.col"
#define FITXER_RES_COL "/home/gbonals/Haydn/prc/src/colisions.txt"

static int detectada_colisio;
static dGeomID id_solids[NSOL];
static dWorldID world=NULL;
static dSpaceID space=NULL;
static dJointGroupID contact_joint_group=NULL;

//------------------------------------------------------------------------------
void webots_physics_init(dWorldID w,dSpaceID s,dJointGroupID j)
```

```
{
  int i;

  // Inicialitzacions estandar per emprar ODE
  world = w;
  space = s;
  contact_joint_group = j;
  printf("ODE: Init:\n"
         "ODE: dWorldID = %p - dSpaceID = %p - dContactJointGroupID = %p\n",w,s,j);

  // adquisicio punters dels solids
  detectada_colisio=0;
  num_solids=0;
  printf("Llegint noms dels òslids...\n");
  inicialitzar_tau_noms(FITXER_NO_COL);
  for(i=0;i<num_solids;i++)
    id_solids[i] = dWebotsGetGeomFromDEF(taula_noms[i]);
  //for(i=0;i<num_solids;i++)
    //printf("ODE: Initialization: %s = %p\n",taula_noms[i],id_solids[i]);
  printf("Inicialitzant taula de colisions...\n");
  inicialitzar_tau_colisions(FITXER_NO_COL);
  //printf("ODE: Initialization detectada_colisio: %d\n",detectada_colisio);
}
//-------------------------------------------------------------------------------
void webots_physics_step()
{
  //printf("ODE: webots_physics_step\n");
  detectada_colisio=0;
}
//-------------------------------------------------------------------------------
int webots_physics_collide(dGeomID g1,dGeomID g2)
{
  char nom1[20],nom2[20];
  dContact contact[10];
  int n,id_g1,id_g2,i;
  FILE* fd;
  //int detectada_colisio=0;

  id_g1=-1;
  id_g2=-1;
  //printf("detectada_colisio= %d\n",detectada_colisio);
  if (!detectada_colisio)
    {
      sprintf(nom1,"UNKNOWN");
      sprintf(nom2,"UNKNOWN");

      // Traduccio dels punters a noms (a efectes de depuracio)
      for(i=0;i<num_solids;i++)
        {
          if(g1==id_solids[i]) sprintf(nom1,taula_noms[i]);
          if(g2==id_solids[i]) sprintf(nom2,taula_noms[i]);
        }

      // Obtenir indexs referents als solids
      for(i=0;i<num_solids;i++)
        {
          if(g1==id_solids[i])
              id_g1=i;
          if(g2==id_solids[i])
              id_g2=i;
        }
      //printf("id_g1:%d, id_g2:%d\n",id_g1,id_g2);

      // Verificar col.lisio nomes si es permet
      if (((id_g1!=-1)&&(id_g2!=-1))&&(cols_permeses[id_g1][id_g2]))
        {
          n = dCollide(g1,g2,10,&contact[0].geom,sizeof(dContact));

          // Si hi ha col.lisio...
          if(n>0)
            {
              detectada_colisio=1;
              //printf("ODE: Fine collision detected: %s %s\n", nom1, nom2);
              if((fd=fopen(FITXER_RES_COL,"w"))==NULL)
                printf("ODE: Error en obrir colisions.txt\n");
              else
                {
                  fprintf(fd,"1\n");
                  //printf("ODE: Imprimint 1 a fitxer...\n");
                  fclose(fd);
                }
            }
          //else
          //printf("ODE: Bounding collision detected: %s %s\n", nom1, nom2);
        }
    }
  return 1;
}
//-------------------------------------------------------------------------------
void webots_physics_cleanup()
{
  printf("ODE: Webots_physics_cleanup\n");
```

```
}
//--------------------------------------------------------------------------
```

H.20 tau_colisions.h

```
#ifndef TAU_COLISIONS_H
#define TAU_COLISIONS_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>

#define NSOL 100
#define MAX_CADENA 200

#ifdef _MAIN_COL_
#   define GLOBAL
#else
#   define GLOBAL extern
#endif

GLOBAL char taula_noms[NSOL][MAX_CADENA];
GLOBAL int cols_permeses[NSOL][NSOL];
GLOBAL int num_solids;

void inicialitzar_tau_noms(const char* nom_arxiu);
void inicialitzar_tau_colisions(const char* nom_arxiu);
int nom_index_taula(char*  nom);
void escriure_taula();
int cadenes_iguals(char* cad1, char* cad2);

#endif
```

H.21 tau_colisions.c

```
#ifndef TAU_COLISIONS_H
#define TAU_COLISIONS_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>

#define NSOL 100
#define MAX_CADENA 200

#ifdef _MAIN_COL_
#   define GLOBAL
#else
#   define GLOBAL extern
#endif

GLOBAL char taula_noms[NSOL][MAX_CADENA];
GLOBAL int cols_permeses[NSOL][NSOL];
GLOBAL int num_solids;

void inicialitzar_tau_noms(const char* nom_arxiu);
void inicialitzar_tau_colisions(const char* nom_arxiu);
int nom_index_taula(char*  nom);
void escriure_taula();
int cadenes_iguals(char* cad1, char* cad2);

#endif
```

H.22 colisio.h

```
#ifndef COLISIO_H
#define COLISIO_H

////////////////////////////////////////////////////
/// @file colisio.h
/// @brief Inicialitzem i llegim de fitxer la ·ócollisi.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       Inicialitzen i llegeixen si existeix ·ócollisi
///       en lú'ltim pas de ósimulaci de Webots.
////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>

void inicialitzar_colisio();
int obtenir_colisio();

#endif
```

H.23 colisio.c

```
#include "colisio.h"

#define FITXER_RES_COL "/home/gbonals/Haydn/prc/src/colisions.txt"

void inicialitzar_colisio()
{
  FILE* fd;

  if((fd=fopen(FITXER_RES_COL,"w"))==NULL)
    printf("Error en inicialitzar colisions.txt\n");
  else
    {
      fprintf(fd,"0\n");
      fclose(fd);
    }

}
int obtenir_colisio()
{

/*
 * Aqui recuperem el valor emmagatzemat per l'arxiu "colisio.txt", que conte un
 * 1, o 0, en funcio si hi ha hagut colisio o no de la cadena tancada dels robots
 * respecte algun solid de la cella de treball.
 */
  int colisio;
  FILE* fd;

  if((fd=fopen(FITXER_RES_COL,"r"))==NULL)
    printf("Error en llegir colisions.txt\n");
  else
    {
      fscanf(fd,"%d\n",&colisio);
      fclose(fd);
    }
  return colisio;
}
```

H.24 component.h

```
#ifndef TAULA_COMPONENTS_H
#define TAULA_COMPONENTS_H

////////////////////////////////////////////////////
/// @file component.h
/// @brief Treballen amb una taula de components del graf.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       En una estructura COMPONENT emmagatzemen el
///       nombre de nodes de cada component del graf i
///       node representatiu de cadascun d'ells.
////////////////////////////////////////////////////



#include "globals.h"
#include "macros_generals.h"
#include "graf.h"

typedef struct struct_component
{
  int num_vertexs;
  list pvertex;
}COMPONENT;

#define NO_ULTIM_COMPONENT(taula_components,i) ((i<g_NUM_COMPONENTS)&&(taula_components[i].num_vertexs!=-1))

COMPONENT* crear_taula_components();
void alliberar_taula_components(COMPONENT** taula_components);
int index_num_vertexs_minim(COMPONENT* taula_components);
int compare_num_vertexs (const void * a, const void * b);
void escriure_taula_components(COMPONENT* taula_components);
#endif
```

H.25 component.c

```
#include "component.h"

COMPONENT* crear_taula_components()
{
  COMPONENT* taula_components;
  int i;

  if(NEW(taula_components,g_NUM_COMPONENTS,COMPONENT)==NULL)
    cerror("error de malloc en crear_taula_vertexs_expan()\n");
  for(i=0;i<g_NUM_COMPONENTS;i++)
```

```
    {
      taula_components[i].num_vertexs=-1;
      taula_components[i].pvertex=NULL;
    }
  return taula_components;
}
void alliberar_taula_components(COMPONENT** taula_components)
{
  free((*taula_components));
  *taula_components=NULL;
}
int index_num_vertexs_minim(COMPONENT* taula_components)
{
  int i,imin;

  imin=0;
  for(i=1;i<g_NUM_COMPONENTS;i++)
    if (taula_components[i].num_vertexs<taula_components[imin].num_vertexs)
      imin=i;
  return imin;
}
int compare_num_vertexs (const void * a, const void * b)
{
 if (((const COMPONENT*)a)->num_vertexs<((const COMPONENT*)b)->num_vertexs)
    return(1);
  else if (((const COMPONENT*)a)->num_vertexs>((const COMPONENT*)b)->num_vertexs)
    return(-1);
  else
    return(0);
}
void escriure_taula_components(COMPONENT* taula_components)
{
  int i;
  int vertexs_totals=0;
  int num_components=0;

  for (i=0;i<g_NUM_COMPONENTS;i++)
    {
      if (taula_components[i].num_vertexs>0)
        {
          vertexs_totals=vertexs_totals+taula_components[i].num_vertexs;
          num_components=num_components+1;
        }

    }
  printf("\nNombre de components totals: %d\n", num_components);
  printf("Nombre de èvrtexs totals: %d\n", vertexs_totals);
  printf ("[#vertexs,vertex repres,%%sobre total]\n");
  for (i=0;i<g_NUM_COMPONENTS;i++)
    {
      if (taula_components[i].num_vertexs>0)
        {
          printf ("[%5d, %6d, %4.2f]\n ",taula_components[i].num_vertexs,
                  VERTEX_NUMBER(taula_components[i].pvertex),
                  ((real)taula_components[i].num_vertexs/(real)vertexs_totals)*100.0);
        }
      else
        break;
    }
}
```

H.26 Gtypes.h

```
/*************************************************************************
*       PUMA 560 Off-line Programming and Simulation Tool
*       (c) Institute of Industrial Robotics
*       Barcelona, Spain
*       Version 1.0
*
* Module:       Mtypes.h  (constants and data types)
* Date:         April 1989
* UPdated:      February 2001
*
*************************************************************************/

/*************************************************************************
*                               CONSTANTS
*************************************************************************/
#ifndef GTYPES_H
#define GTYPES_H

#include <stdio.h>
#include <math.h>
#include "Gdata.h"
#include "general.h"

#ifndef TRUE
#define TRUE   1
#endif

#ifndef FALSE
```

```
#define FALSE  0
#endif

/* #define NULL   0*/
#define PIB2              1.57079632679489660     /* pi / 2               */
#define PI                3.14159265358979320     /* pi                   */
#define PIT2              6.28318530717958650     /* pi * 2               */
#define RADTODEG          57.29577951308232100    /* 180 / pi             */
#define DEGTORAD          0.01745329251994330     /* pi / 180             */
#define SMALL             (1.e-5)                 /* considered as small  */
#define SMALL4            (1.e-3)
#define BIG               (1.e+5)                 /* considered as big    */
#define EQ                1                       /* lhs = rhs            */
#define TL                2                       /* tool =               */

#define KROT   16384.        /* VAL-II Programming Manual (Chapter 5-5) */
#define KTRAS  32.

#define NOMFICHERO  "Msim.dat"  /* Nombre del fichero donde se lee la    */
                                /* descripcion de las piezas             */

#define LEN          32 /* Se utiliza en la generacion de LUTs de colores */


/*************************************************************************
*                              MACROS
*************************************************************************/

#define SINCOS(s, c, a) {s = sin(a); c = cos(a);}

#define FABS(a)          (((a) < 0.) ? -(a) : (a))

#define ABS(a)           (((a) < 0) ? -(a) : (a))

#define ROUND(a)         ((a - (double)(int)a >= .5) ? (int)a + 1 : (int)a)

#define SIGN(a)          (((a) >= 0.0) ? 1.0 : -1.0)

#define SQR(a)           ((a) * (a))

#define CUBE(a)          ((a) * (a)  * (a))

#define Malloc(_ti)      ((_ti*)malloc((unsigned)sizeof(_ti)))

#define NEW(_var,_n,_type) ((_var)=(_type *)malloc(sizeof(_type)*(_n)))

#define Nalloc(_ta)      (malloc((unsigned)_ta))

#define Realloc(_p,_ti,_n)       ((_ti*)realloc((char *)_p,(unsigned)(_n*sizeof(_ti))))

#define Calloc(_ti,_n)   ((_ti*)calloc((unsigned)_n,(unsigned)sizeof(_ti)))

#define Free(_p)         free((char *)_p)

#define Min( a, b)       ( (a)<(b) ? (a) : (b) )

#define Max( a, b)       ( (a)>(b) ? (a) : (b) )


/*************************************************************************
*                        DEFINICION DE TIPOS
*************************************************************************/

typedef int boolean;

typedef double Greal;

typedef struct vector {
        Greal x, y, z;
} VECT;

typedef struct {
        VECT   n,o,a,p;
} TRSF;


typedef struct jns {
        Greal th1, th2, th3, th4, th5, th6;
} JNTS;

typedef struct peca {
  VECT d;
  VECT rot;
}CONFPECA;

typedef struct jnts_range{
  JNTS  jofst_c;
  JNTS  jrng_c;
  JNTS jmin_c;
  JNTS jmax_c;
}JNTS_RANGE;
```

```
typedef struct kindyn {
        Greal a2, a3, d3, d4, d6;
}PDH;


/*Gutil.c */

TRSF*  trmult(TRSF* r, TRSF* t, TRSF* u);
TRSF* trmultinp(TRSF* r, TRSF* m);
TRSF* trsl(TRSF* t, Greal x, Greal y, Greal z);
TRSF* trslm(TRSF* t, Greal x, Greal y, Greal z);

/* Gmotion.c */

void equal_inside_range(JNTS* j,JNTS jmin_c,JNTS jmax_c);
void explain_code(int code);
Greal range(Greal a);
void jns_to_tr_n(PDH p,JNTS* j,TRSF* t,char* c);
Greal range_j6(Greal a);
int tr_to_jns_n(TRSF* tr,PDH p,char c[4],JNTS* j,JNTS_RANGE* jr);
float angle_positiu(Greal a);
Greal dangle_positiu(Greal a);
Greal convertir_angle_a_pi_pi(Greal th);
bool angle_en_rang(int angle,Greal th,Greal thmin,Greal thmax);
bool angle_en_rang_pi_pi(Greal th,Greal thmin,Greal thmax);
int angles_en_rang_interpol_robot (float* join,JNTS_RANGE* jr);


#endif
```

H.27 fitxer.h

```
#ifndef FITXER_H
#define FITXER_H

////////////////////////////////////////////////////
/// @file fitxer.h
/// @brief Treballa amb els fitxer *.map, *.par, *.trj
///        i *.infi.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///     Rutines que treballen amb els fitxers per
///     desar i carregar un mapa (*.map), desar i
///     carregar uns àparmetres (*.par), desar i
///     carregar unes configuracions d'inici i fi
///     (*.infi), desar i carregar una
/// ò    trajectria i finalment desar i carregar
///     fitxers els fitxer visuals del mapa i la
/// ò    trajectria (*.map.wbt i *.map.wbt).
////////////////////////////////////////////////////

#include "graf_config.h"

graph carregar_mapa(const char* nom_fitxer);
void desar_mapa(graph G,const char *nom_fitxer);
void carregar_infi(const char* nom_fitxer,CONFIG** config_ini,CONFIG** config_fi);
void desar_infi(CONFIG* config_ini,CONFIG* config_fi,const char* nom_fitxer);
void desar_parametres_mapa(const char *nom_fitxer);
list carregar_trajectoria(const char *nom_fitxer);
void desar_trajectoria(list traject,real distancia_total,const char *nom_fitxer);

list borrar_ultim_node_trajectoria(list traject);
void desar_mapa_en_wbt(graph G,const char *nom_fitxer);
void desar_cami_en_wbt(list traject,const char *nom_fitxer);

#endif
```

H.28 fitxer.c

```
#include "fitxer.h"


////////////////////////////////////////////////////
//Rutines privades a configuracio.c
////////////////////////////////////////////////////
static void desar_linia(CONFIG* config1,CONFIG* config2,FILE* fmapa_triedres);
static void desar_triedre(CONFIG* config,int cont_triedre,FILE* fgrup_triedres);
static void desar_primer_triedre(CONFIG* config,FILE* fgrup_triedres);

////////////////////////////////////////////////////
// carregar_mapa()
//
// Crea un graf (llista doblement çenllaada) de configuracions
// a partir d'un arxiu.
//
// Paramatres:
//
// nom_fitxer: Es una cadena de caracters que indica el
```

```
//                                        nom del fitxer que conte el graf.
//                                        Es un parametre de entrada.
//
// valor de retorn: Una estructura graf(llista doblement çenllaada)
//                  que contindra el graf de configuracions.
////////////////////////////////////////////////////////

graph carregar_mapa(const char* nom_fitxer)


{
  FILE* fp;
  CONFIG* config;
  graph G;
  vertex n_nodes;
  vertex vertex_act;
  vertex aresta_act;
  vertex num_vertex;
  char buffer[MAX_CADENA];

   printf("\nCarregant mapa...\n");
   sprintf(buffer,"nom_fitxer: %s",nom_fitxer);
   printf("%s\n", buffer);
   escriure_text_a_scenee_tree("Carregant mapa..");

   if ((fp=fopen(nom_fitxer,"r"))==NULL)
     {
       printf("No podem obrir %s per llegir-lo\n",nom_fitxer);
       escriure_text_a_scenee_tree("no tenim el fitxer mapa per carregar");
       G=NULL;
     }
   else
     {
       if(init_graph(&G)==ERROR)
         cerror("Error:: en fase aprenentatge()");

       fscanf(fp,"#vertexs=%d\n\n",&n_nodes);

       //llegeixo els vertexs del arxiu i els guardo en un graf.
       for(vertex_act=1; vertex_act<=n_nodes;vertex_act++)
         {
           config=crear_configuracio();
           fscanf(fp,"VERTEX %d\n",&(num_vertex));
           carregar_configuracio(fp,config);
           fscanf(fp,"\n");
           if(add_vertex(&G,num_vertex,(generic_ptr)config)==ERROR)
             cerror("Error:: en carregar_mapa() ");
         }
       //llegeixo les arestes de cada vertex del arxiu i les guardo al graf.
       for(vertex_act=1; vertex_act<=n_nodes;vertex_act++)
         {
           fscanf(fp,"ARESTES DEL VERTEX %d\n\n",&(num_vertex));
           fscanf(fp,"%d ",&aresta_act);
           while (aresta_act!=-1)
             {
               if (aresta_act>num_vertex)
                 {
                   if ((add_edge(G,num_vertex,aresta_act,NULL))==ERROR)
                     cerror("Error:: en carregar_mapa()");
                 }
               fscanf(fp,"%d  ",&aresta_act);
             }
           fscanf(fp,"\n\n");
         }
       fclose(fp);
       printf("Mapa carregat\n");
       escriure_text_a_scenee_tree("Mapa carregat");
     }
   return G;
}

////////////////////////////////////////////////////////
// desar_mapa()
//
// Guarda en un arxiu la informacio de un graf de configuracions.
//
// Paramatres:
//
// G: Es una estructura graf, que es una llista doblement çenllada.
// Es un parametre de entrada.
//
// nom_fitxer: Es una cadena de caracters per guardar el nom del fitxer que
// contindra el graf. Es un parametre de sortida.
//
// valor de retorn: buit
////////////////////////////////////////////////////////

void desar_mapa(graph G,const char *nom_fitxer)

{
  FILE* fp;
  list pvertex_act=NULL;
```

```
  list paresta_act;
  CONFIG* config;
  vertex num_vertex;
  char buffer[MAX_CADENA];


  printf("\nDesant mapa...\n");
  sprintf(buffer,"nom fitxer: %s",nom_fitxer);
  printf("%s\n", buffer);

  if (empty_list(G))
    {
      printf("No tenim cap mapa per desar\n");
      escriure_text_a_scenee_tree("No tenim cap mapa per desar");
    }
  else
    {
      if ((fp=fopen(nom_fitxer,"w"))==NULL)
        {
          printf("No podem obrir %s per escriure'l\n",nom_fitxer);
          escriure_text_a_scenee_tree("No hem pogut desar el mapa");
        }
      else
        {
          //escric les configuracions de cada vertex
          fprintf(fp,"#vertexs=%d\n\n",length(G));
          while ((pvertex_act=list_iterator(G, pvertex_act))!=NULL)
            {
              num_vertex=VERTEX_NUMBER(pvertex_act);
              config=CONFIG_VERTEX(pvertex_act);
              fprintf(fp,"VERTEX %d\n",num_vertex);
              desar_configuracio(fp,config);
              fprintf(fp,"\n");
            }
          //escric les arestes de cada vertex
          pvertex_act=NULL;
          while ((pvertex_act=list_iterator(G,pvertex_act))!=NULL)
            {
              num_vertex=VERTEX_NUMBER(pvertex_act);
              fprintf(fp,"ARESTES DEL VERTEX %d\n\n",num_vertex);
              paresta_act=NULL;
              while((paresta_act=list_iterator(EMANATING(pvertex_act),paresta_act))!=NULL)
                fprintf(fp,"%d ",DESTINATION(paresta_act));
              fprintf(fp,"%d\n\n",-1);
            }
          fclose(fp);
          printf("Mapa desat\n");
          escriure_text_a_scenee_tree("Mapa desat");
        }
    }
}

void carregar_infi(const char* nom_fitxer,CONFIG** config_ini,CONFIG** config_fi)
{
  FILE* finfi;
  char buffer[MAX_CADENA];



  if ((finfi=fopen(nom_fitxer,"r"))==NULL)
    {
    printf("\nNo podem obrir %s per llegir-lo\n",nom_fitxer);
    escriure_text_a_scenee_tree("No tenim el fitxer infi per carregar");
    }
  else
    {
      //éNoms borrem les configuracions si sabem que existeix l'arxiu.
      if (((*config_ini)!=NULL)&&((*config_fi)!=NULL))
        {
          alliberar_configuracio(config_ini);
          alliberar_configuracio(config_fi);
        }

      (*config_ini)=crear_configuracio();
      (*config_fi)=crear_configuracio();

      fscanf(finfi,"óConfiguraci inicial\n\n");
      carregar_angles_robot(finfi,ROBOTE((*config_ini)));
      carregar_angles_robot(finfi,ROBOTD((*config_ini)));
      carregar_posicio_objecte(finfi,OBJECTE((*config_ini)));

      fscanf(finfi,"----------------------------------------\n");

      fscanf(finfi,"óConfiguraci final\n\n");
      carregar_angles_robot(finfi,ROBOTE((*config_fi)));
      carregar_angles_robot(finfi,ROBOTD((*config_fi)));
      carregar_posicio_objecte(finfi,OBJECTE((*config_fi)));
      fclose(finfi);

      printf("\nCarregades configuracio inicial i final\n");
      sprintf(buffer,"nom fitxer: %s",nom_fitxer);
      printf("%s\n", buffer);
```

```
      escriure_text_a_scenee_tree("Carregades configuracio inicial i final");
    }
}
void desar_infi(CONFIG* config_ini,CONFIG* config_fi,const char* nom_fitxer)
{
  FILE* finfi;
  char buffer[MAX_CADENA];

  if ((config_ini==NULL)||(config_fi==NULL))
    {
      printf("\nNo tenim config. d'inici i final per desar\n");
      escriure_text_a_scenee_tree("no tenim config. d'inici i final per desar");
    }
  else
    {
      if ((finfi=fopen(nom_fitxer,"w"))==NULL)
        {
          printf("No podem obrir %s per escriure'l\n",nom_fitxer);
          escriure_text_a_scenee_tree("no hem pogut desar les config. ini i final");
        }
      else
        {
          fprintf(finfi,"óConfiguraci inicial\n\n");
          desar_angles_robot(finfi,ROBOTE(config_ini));
          desar_angles_robot(finfi,ROBOTD(config_ini));
          desar_posicio_objecte(finfi,OBJECTE(config_ini));

          fprintf(finfi,"----------------------------------------\n");

          fprintf(finfi,"óConfiguraci final\n\n");
          desar_angles_robot(finfi,ROBOTE(config_fi));
          desar_angles_robot(finfi,ROBOTD(config_fi));
          desar_posicio_objecte(finfi,OBJECTE(config_fi));
          fclose(finfi);

          printf("\nDesades configuracions inicial i final\n");
          sprintf(buffer,"nom fitxer: %s",nom_fitxer);
          printf("%s\n", buffer);

          escriure_text_a_scenee_tree("Desades configuracions inicial i final");
        }
    }
}
void desar_parametres_mapa(const char *nom_fitxer)
{
  FILE* fp;

 if ((fp=fopen(nom_fitxer,"w"))==NULL)
    printf("No podem obrir %s per escriure'l\n",nom_fitxer);
  else
    {
      fprintf(fp,"PRM\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"max. nodes:%d\n",g_MAX_NODES);
      fprintf(fp,"min. nodes:%d\n",g_MIN_NODES);
      fprintf(fp,"num. veins:%d\n",g_NVEINS);
      fprintf(fp,"max. dist:%f\n\n",g_MAX_DIST);
      fprintf(fp,"Cami local\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"pas cami local:%f\n\n",g_pas_cami_local);
      fprintf(fp,"Fase Cerca/Expansio\n");
      fprintf(fp,"----------------------\n");
      fprintf(fp,"num. pasos:%d\n",g_N_PASOS);
      fprintf(fp,"fora dist. max.:%d\n",g_N_FORA_DIST_MAX);
      fprintf(fp,"max. intents:%d\n",g_max_intents);
      fprintf(fp,"max. profunditat:%d\n",g_max_profund);
      fprintf(fp,"num. components:%d\n",g_NUM_COMPONENTS);
      fclose(fp);
    }
}
/*-------------------------------------------------------------------------------------------------------------*/
void desar_trajectoria(list traject,real distancia_total,const char *nom_fitxer)

/**********************************************************************
* desar_trajectoria()
*
* Guarda en un fitxer de text una llista de vertexs, on cada vertex conte
* una configuracio, i on totes les configuracions corresponent a punts de
* pas valids duna trajectoria entre dos configuracions predeterminades.
*
* Paramatres:
*
* traject: Es una variable que es una llista çenllaada, on cada node
* conte una configuracio valida.
*
* nom_fitxer: Es una cadena de caracters per guardar el nom del fitxer que
* contindra la llista. Es un parametre de sortida.
*
* valor de retorn: Buit.
*
*
```

```
*****************************************************************************/
{
  FILE* fp;
  list pnode_act=NULL;
  CONFIG* config;
  char buffer[MAX_CADENA];

  printf("\nDesant òtrajectria...\n");
  sprintf(buffer,"nom_fitxer:%s",nom_fitxer);
  printf("%s\n", buffer);

  if (empty_list(traject))
    {
      printf("No tenim òtrajectria per desar\n");
      escriure_text_a_scenee_tree("No tenim cap trajectoria per desar");
    }
  else
    {
      if ((fp=fopen(nom_fitxer,"w"))==NULL)
        {
          printf("No podem obrir %s per escriure'l\n",nom_fitxer);
          escriure_text_a_scenee_tree("No hem pogut desar la trajectoria");
        }
      else
        {
          //vaig recorrent la llista i guardant les configuracions dels vertexs
          fprintf(fp,"#vertexs=%d\n",length(traject));
          fprintf(fp,"distancia_total:%f\n\n",distancia_total);

          while ((pnode_act=list_iterator(traject,pnode_act))!=NULL)
            {
              config=(CONFIG*)DATA(pnode_act);
              desar_configuracio(fp,config);
              fprintf(fp,"\n");
            }
          fclose(fp);
          printf("òTrajectria desada\n");
          escriure_text_a_scenee_tree("Trajectoria desada");
        }
    }
}

/*---------------------------------------------------------------------------------------------------------------------*/

list carregar_trajectoria(const char *nom_fitxer)

/*****************************************************************************
* carregar_trajectoria()
*
* Crear a partir de un fitxer de text una llista de vertexs, on cada vertex conte
* una configuracio, i on totes les configuracions corresponent a punts de
* pas valids duna trajectoria entre una configuracio inicial i final
* predeterminades.
*
* Paramatres:
*
* nom_fitxer: Es una cadena de caracters que indica el nom del fitxer que
* conte el graf. Es un parametre de entrada.
*
* valor de retorn: Retorna una llista de vertexs, on cada vertex conte
* una configuracio.
*
*
*****************************************************************************/

{
  FILE* fp;
  CONFIG* config;
  int n_nodes;
  list traject;
  int i;
  real distancia_total;
  char buffer[MAX_CADENA];

  printf("\nCarregant òtrajectria...\n");
  sprintf(buffer,"nom_fitxer:%s",nom_fitxer);
  printf("%s\n", buffer);

  if ((fp=fopen(nom_fitxer,"r"))==NULL)
    {
      printf("No podem obrir %s per llegir-lo\n",nom_fitxer);
      escriure_text_a_scenee_tree("No tenim el fitxer trj per carregar");
      traject=NULL;
    }
  else
    {
      if(init_list(&traject)==ERROR)
        cerror("Error: carregar_trajectoria():: trajectoria no inicialitzada");

      //llegeixo els vertexs de un fitxer i creo una llista de vertexs
      fscanf(fp,"#vertexs=%d\n",&n_nodes);
```

```
      fscanf(fp,"distancia_total:%f\n\n",&distancia_total);
      for(i=1;i<=n_nodes;i++)
        {
          config=crear_configuracio();
          carregar_configuracio(fp,config);
          append(&traject,(generic_ptr)config);
          fscanf(fp,"\n");
        }
      fclose(fp);

      printf("òTrajectria carregada\n");
      escriure_text_a_scenee_tree("Trajectoria carregada");
    }
  return(traject);
}

/*-------------------------------------------------------------------------------------------------------------------------*/

list borrar_ultim_node_trajectoria(list traject)

/*************************************************************************
* borrar_ultim_node_trajectoria()
*
* Aquest accio borra de la llista de nodes, l ultim node.
*
* Paramatres:
*
* traject: Conte la llista de nodes de la trajectoria. Es una variable
* d entrada.
*
* valor de retorn: Retorna una la llista anterior,pero sense l ultim node
* que s ha suprimit.
*
*
*************************************************************************/
{
  list pult_node;
  //busquem el punter a l ultim node de la llista
  pult_node=last(traject);
  //borrem l ultim node de la llista
  if(delete_node(&traject,pult_node,NULL)==ERROR)
    cerror("Error en borrar_ultim_node_trajectoria()");
  return traject;
}

/*-------------------------------------------------------------------------------------------------------------------------*/

void desar_primer_triedre(CONFIG* config,FILE* fgrup_triedres)
{
  char nom_triedre[100]="triedre_objecte";

  fprintf (fgrup_triedres,"DEF group_triedres Group {");
  fprintf (fgrup_triedres,"  children [\n");
  fprintf (fgrup_triedres,"    DEF %s%d Transform {\n",nom_triedre,1);
  fprintf (fgrup_triedres,"      translation %f %f %f\n",OBJ_TRANSL_i(config,0),
           OBJ_TRANSL_i(config,1),OBJ_TRANSL_i(config,2));
  fprintf (fgrup_triedres,"      rotation 0 0 0 0\n");
  fprintf (fgrup_triedres,"      children [\n");
  fprintf (fgrup_triedres,"        DEF rotacio_x Transform {\n");
  fprintf (fgrup_triedres,"          rotation 1 0 0 %f\n", OBJ_ROT_i(config,0));
  fprintf (fgrup_triedres,"          children [\n");
  fprintf (fgrup_triedres,"            DEF rotacio_y Transform {\n");
  fprintf (fgrup_triedres,"              rotation 0 1 0 %f\n",OBJ_ROT_i(config,1));
  fprintf (fgrup_triedres,"              children [\n");
  fprintf (fgrup_triedres,"                DEF rotacio_z Transform {\n");
  fprintf (fgrup_triedres,"                  rotation 0 0 1 %f\n",OBJ_ROT_i(config,2));
  fprintf (fgrup_triedres,"                  children [\n");
  fprintf (fgrup_triedres,"                    DEF triedre_group Group {\n");
  fprintf (fgrup_triedres,"                      children [\n");
  fprintf (fgrup_triedres,"                        DEF eix_x Shape {\n");
  fprintf (fgrup_triedres,"                          appearance Appearance {\n");
  fprintf (fgrup_triedres,"                            material Material {\n");
  fprintf (fgrup_triedres,"                              emissiveColor 1 0 0\n");
  fprintf (fgrup_triedres,"                            }\n");
  fprintf (fgrup_triedres,"                          }\n");
  fprintf (fgrup_triedres,"                          geometry IndexedLineSet {\n");
  fprintf (fgrup_triedres,"                            coord Coordinate {\n");
  fprintf (fgrup_triedres,"                              point [\n");
  fprintf (fgrup_triedres,"                                0 0 0\n");
  fprintf (fgrup_triedres,"                                0.02 0 0\n");
  fprintf (fgrup_triedres,"                              ]\n");
  fprintf (fgrup_triedres,"                            }\n");
  fprintf (fgrup_triedres,"                            coordIndex [\n");
  fprintf (fgrup_triedres,"                              0, 1, -1\n");
  fprintf (fgrup_triedres,"                            ]\n");
  fprintf (fgrup_triedres,"                          }\n");
  fprintf (fgrup_triedres,"                        }\n");
  fprintf (fgrup_triedres,"                        DEF eix_y Shape {\n");
  fprintf (fgrup_triedres,"                          appearance Appearance {\n");
  fprintf (fgrup_triedres,"                            material Material {\n");
  fprintf (fgrup_triedres,"                              emissiveColor 0 1 0\n");
  fprintf (fgrup_triedres,"                            }\n");
```

```
  fprintf (fgrup_triedres,"                          }\n");
  fprintf (fgrup_triedres,"                          geometry IndexedLineSet {\n");
  fprintf (fgrup_triedres,"                            coord Coordinate {\n");
  fprintf (fgrup_triedres,"                              point [\n");
  fprintf (fgrup_triedres,"                                0 0 0\n");
  fprintf (fgrup_triedres,"                                0 0.02 0 \n");
  fprintf (fgrup_triedres,"                              ]\n");
  fprintf (fgrup_triedres,"                            }\n");
  fprintf (fgrup_triedres,"                            coordIndex [\n");
  fprintf (fgrup_triedres,"                              0, 1, -1\n");
  fprintf (fgrup_triedres,"                            ]\n");
  fprintf (fgrup_triedres,"                          }\n");
  fprintf (fgrup_triedres,"                        }\n");
  fprintf (fgrup_triedres,"                        DEF eix z Shape {\n");
  fprintf (fgrup_triedres,"                          appearance Appearance {\n");
  fprintf (fgrup_triedres,"                            material Material {\n");
  fprintf (fgrup_triedres,"                              emissiveColor 0 0 1\n");
  fprintf (fgrup_triedres,"                            }\n");
  fprintf (fgrup_triedres,"                          }\n");
  fprintf (fgrup_triedres,"                          geometry IndexedLineSet {\n");
  fprintf (fgrup_triedres,"                            coord Coordinate {\n");
  fprintf (fgrup_triedres,"                              point [\n");
  fprintf (fgrup_triedres,"                                0 0 0\n");
  fprintf (fgrup_triedres,"                                0 0 0.02 \n");
  fprintf (fgrup_triedres,"                              ]\n");
  fprintf (fgrup_triedres,"                            }\n");
  fprintf (fgrup_triedres,"                            coordIndex [\n");
  fprintf (fgrup_triedres,"                              0, 1, -1\n");
  fprintf (fgrup_triedres,"                            ]\n");
  fprintf (fgrup_triedres,"                          }\n");
  fprintf (fgrup_triedres,"                        }\n");
  fprintf (fgrup_triedres,"                      ]\n");
  fprintf (fgrup_triedres,"                    }\n");
  fprintf (fgrup_triedres,"                  ]\n");
  fprintf (fgrup_triedres,"                }\n");
  fprintf (fgrup_triedres,"                DEF rotacio_z Transform {\n");
  fprintf (fgrup_triedres,"                  rotation 0 0 1 %f\n",OBJ_ROT_i(config,2));
  fprintf (fgrup_triedres,"                  children [\n");
  fprintf (fgrup_triedres,"                    USE triedre_group\n");
  fprintf (fgrup_triedres,"                  ]\n");
  fprintf (fgrup_triedres,"                }\n");
  fprintf (fgrup_triedres,"              ]\n");
  fprintf (fgrup_triedres,"            }\n");
  fprintf (fgrup_triedres,"          ]\n");
  fprintf (fgrup_triedres,"        }\n");
  fprintf (fgrup_triedres,"      ]\n");
  fprintf (fgrup_triedres,"    }\n");
}
void desar_triedre(CONFIG* config,int cont_triedre,FILE* fgrup_triedres)
{
  char nom_triedre[100]="triedre_objecte";

  fprintf (fgrup_triedres,"    DEF %s%d Transform {\n",nom_triedre,cont_triedre);
  fprintf (fgrup_triedres,"      translation %f %f %f\n",OBJ_TRANSL_i(config,0),
           OBJ_TRANSL_i(config,1),OBJ_TRANSL_i(config,2));
  fprintf (fgrup_triedres,"      rotation 0 0 0 0\n");
  fprintf (fgrup_triedres,"      children [\n");
  fprintf (fgrup_triedres,"        DEF rotacio_x Transform {\n");
  fprintf (fgrup_triedres,"          rotation 1 0 0 %f\n", OBJ_ROT_i(config,0));
  fprintf (fgrup_triedres,"          children [\n");
  fprintf (fgrup_triedres,"            DEF rotacio_y Transform {\n");
  fprintf (fgrup_triedres,"              rotation 0 1 0 %f\n",OBJ_ROT_i(config,1));
  fprintf (fgrup_triedres,"              children [\n");
  fprintf (fgrup_triedres,"                DEF rotacio_z Transform {\n");
  fprintf (fgrup_triedres,"                  rotation 0 0 1 %f\n",OBJ_ROT_i(config,2));
  fprintf (fgrup_triedres,"                  children [\n");
  fprintf (fgrup_triedres,"                    USE triedre_group\n");
  fprintf (fgrup_triedres,"                  ]\n");
  fprintf (fgrup_triedres,"                }\n");
  fprintf (fgrup_triedres,"              ]\n");
  fprintf (fgrup_triedres,"            }\n");
  fprintf (fgrup_triedres,"          ]\n");
  fprintf (fgrup_triedres,"        }\n");
  fprintf (fgrup_triedres,"      ]\n");
  fprintf (fgrup_triedres,"    }\n");
}

void desar_linia(CONFIG* config1,CONFIG* config2,FILE* fmapa_triedres)
{
  fprintf(fmapa_triedres,"    DEF linia Shape {\n");
  fprintf(fmapa_triedres,"      appearance Appearance {\n");
  fprintf(fmapa_triedres,"        material Material {\n");
  fprintf(fmapa_triedres,"        }\n");
  fprintf(fmapa_triedres,"      }\n");
  fprintf(fmapa_triedres,"      geometry IndexedLineSet {\n");
  fprintf(fmapa_triedres,"        coord Coordinate {\n");
  fprintf(fmapa_triedres,"          point [\n");
  fprintf(fmapa_triedres,"            %f %f %f\n",OBJ_TRANSL_i(config1,0),
          OBJ_TRANSL_i(config1,1),OBJ_TRANSL_i(config1,2));
  fprintf(fmapa_triedres,"            %f %f %f\n",OBJ_TRANSL_i(config2,0),
          OBJ_TRANSL_i(config2,1),OBJ_TRANSL_i(config2,2));
  fprintf(fmapa_triedres,"          ]\n");
  fprintf(fmapa_triedres,"        }\n");
  fprintf(fmapa_triedres,"        coordIndex [\n");
  fprintf(fmapa_triedres,"          0, 1, -1\n");
  fprintf(fmapa_triedres,"        ]\n");
  fprintf(fmapa_triedres,"      }\n");
  fprintf(fmapa_triedres,"    }\n");
```

```
}

void desar_mapa_en_wbt(graph G,const char *nom_fitxer)

{
  FILE* fmapa_triedres;
  list pvertex_act=NULL;
  list paresta_act;
  CONFIG* config1;
  CONFIG* config2;
  vertex num_vertex;
  bool primera_vegada=TRUE;

  if ((fmapa_triedres=fopen(nom_fitxer,"w"))==NULL)
    printf("No podem obrir %s per escriure'l\n",nom_fitxer);
  else
    {
      //escric les configuracions de cada vertex
      while ((pvertex_act=list_iterator(G, pvertex_act))!=NULL)
        {
          num_vertex=VERTEX_NUMBER(pvertex_act);
          config1=CONFIG_VERTEX(pvertex_act);
          if (primera_vegada)
            {
              desar_primer_triedre(config1,fmapa_triedres);
              primera_vegada=FALSE;
            }
          else
            desar_triedre(config1,(int)num_vertex,fmapa_triedres);
        }
      //escric les arestes de cada vertex
      pvertex_act=NULL;
      while ((pvertex_act=list_iterator(G,pvertex_act))!=NULL)
        {
          config1=CONFIG_VERTEX(pvertex_act);
          paresta_act=NULL;
          while((paresta_act=list_iterator(EMANATING(pvertex_act),paresta_act))!=NULL)
            {
              if (DESTINATION(paresta_act)>VERTEX_NUMBER(pvertex_act))
                {
                  config2=CONFIG_VERTEX(VERTEX_POINTER(paresta_act));
                  desar_linia(config1,config2,fmapa_triedres);
                }
            }
        }
      fprintf (fmapa_triedres,"  ]\n");
      fprintf (fmapa_triedres,"}\n");
      fclose(fmapa_triedres);
    }
}


void desar_cami_en_wbt(list traject,const char *nom_fitxer)
{

  FILE* fcami_triedres;
  list pnode_ant;
  list pnode_act=NULL;
  CONFIG* config1;
  CONFIG* config2;
  vertex num_vertex;
  bool primera_vegada=TRUE;

  if (empty_list(traject))
    printf("No podem desar la trajectoria en wbt èperqu no n'hi ha\n");
  else
    {
      if ((fcami_triedres=fopen(nom_fitxer,"w"))==NULL)
        printf("No podem obrir %s per escriure'l\n",nom_fitxer);
      else
        {
          //escric els triedres per cada configuracio
          while ((pnode_act=list_iterator(traject, pnode_act))!=NULL)
            {
              //num_vertex=VERTEX_NUMBER(pnode_act);
              config1=(CONFIG*)(DATA(pnode_act));

              if (primera_vegada)
                {
                  desar_primer_triedre(config1,fcami_triedres);
                  primera_vegada=FALSE;
                }
              else
                desar_triedre(config1,(int)num_vertex,fcami_triedres);
            }

          //per escriure les linies entre els triedres
          pnode_ant=traject;
          pnode_act=NEXT(pnode_ant);
          while (pnode_act!=NULL)
            {
              config1=(CONFIG*)(DATA(pnode_ant));
```

```
            config2=(CONFIG*)(DATA(pnode_act));
            desar_linia(config1,config2,fcami_triedres);

            pnode_ant=pnode_act;
            pnode_act=NEXT(pnode_ant);
          }
        fprintf (fcami_triedres,"]\n");
        fprintf (fcami_triedres,"}\n");
        fclose(fcami_triedres);
      }
  }
}
```

H.29 glr.h

```
#ifndef GLR_H
#define GLR_H
/*****************************************************************
 *  GLRß 0.1
 *  Graph Libraryß 0.1
 *
 *  graph_types.h
 *  Copyright: (c) Lluis Ros 1995, 1996
 *  Creation: April 1995
 *  Revision: May 1999
 *
 *  Declaration of graph types and prototypes
 *****************************************************************/


/*****************************************************************
 *  vertex_data type and its accessor macros
 *****************************************************************/
typedef list graph;
typedef unsigned int vertex;
typedef struct{
  generic_ptr specific_data;
  list emanating;
  vertex vertex_number;
  vertex vertex_index;
  bool visited;
  int label;    /* multipurpose label, e.g. to colour cycles */
  list labels; /* list of labels of fundamental cycles that contain this vertex */
} vertex_data;

#define EMANATING(V)      (((vertex_data*)DATA(V))->emanating)
#define VERTEX_NUMBER(V) (((vertex_data*)DATA(V))->vertex_number)
#define VERTEX_INDEX(V)  (((vertex_data*)DATA(V))->vertex_index)
#define VISITED(V)        (((vertex_data*)DATA(V))->visited)
#define LABEL(V)          (((vertex_data*)DATA(V))->label)
#define LABELS(V)         (((vertex_data*)DATA(V))->labels)
#define THISLABEL(L)      (*((int*)DATA(L)))

/*****************************************************************
 *  edge_data type and its accessor macros
 *****************************************************************/
typedef struct{
  generic_ptr specific_data;
  vertex vertex_ini;     /* vertex where this arc emanates from (from storage point of view) */
  vertex vertex_number;  /* vertex where this arc points to (from storage point of view) */
  list vertex_pointer;
  int copy;
  int sense; /* May be one of UP, DOWN, or NOSENSE (see below).
                Is UP if vertex vertex_number is upstream in the DFS order */
} edge_data;

#define SOURCE(E)     (((edge_data*)DATA(E))->vertex_ini)
#define DESTINATION(E)     (((edge_data*)DATA(E))->vertex_number)
#define VERTEX_POINTER(E) (((edge_data*)DATA(E))->vertex_pointer)
#define COPY(E)           (((edge_data*)DATA(E))->copy)
#define SENSE(E)          (((edge_data*)DATA(E))->sense)


/*****************************************************************
 *  cycle_data type and its accessor macros
 *****************************************************************/
typedef struct {
  list vertices; /* list of vertices of the cycle */
  list edges;    /* list of edges of the cycle */
  int label;     /* label of this cycle */
} cycle_data;

#define CYC_VERTICES(C) (((cycle_data*)DATA(C))->vertices)
#define CYC_EDGES(C)    (((cycle_data*)DATA(C))->edges)
#define CYC_LABEL(C)    (((cycle_data*)DATA(C))->label)

/*****************************************************************
 *  other macros
 *****************************************************************/

#define UP 1
```

```
#define DOWN 0
#define NOSENSE -1
#define NOLABEL -1

/* end condition for the up() function */
#define BY_NODE_ID 1
#define BY_LABEL 0

/* labeling mode for up() */
#define DOLABEL 1
#define DONTLABEL 0


/*****************************************************************
 *  Primitive operations of the "graph" abstract data type
 *****************************************************************/
status  init_graph(graph *p_G);
void    destroy_graph(graph *p_G);
status  add_vertex(graph* p_G, vertex vertex_number, generic_ptr specific_data);
status  add_edge(graph G, vertex vertex1, vertex vertex2, generic_ptr specific_data);
status  add_edge_with_pointers(list pv1, list pv2, generic_ptr specific_data);
status  add_edge2(graph G, vertex vertex1, vertex vertex2, generic_ptr specific_data);
status  add_edge_with_pointers2(list pv1, list pv2, generic_ptr specific_data);
status  edge_append(list* p_L, vertex vertex_ini, vertex vertex_number,
                    generic_ptr specific_data, list vertex_pointer, int copy);
list    find_vertex(graph G, vertex vertex_number);
int     cmp_vertex( generic_ptr p_datapointer1, generic_ptr p_datapointer2);
status  delete_edge(graph G, vertex vertex1, vertex vertex2);
status  delete_edge_with_pointers(list origin1, list destin1, list origin2, list destin2);
list    find_destin(list L, vertex v);
list    find_destin_with_pointer(list L, list pv);
status  delete_vertex(graph* p_G, vertex vertex_number);
status  delete_vertex_with_pointer(graph* p_G, list vertex_pointer);
void    free_vertex_data(generic_ptr data);
void    free_edge_data(generic_ptr data);
void    free_specific_vertex_data(generic_ptr data);
void    free_specific_edge_data(generic_ptr data);
void    free_cycles_data(generic_ptr data);
void    free_specific_edge_data(generic_ptr p_data);
void    free_specific_vertex_data(generic_ptr p_data);
status  depth_first_vertices_edges(graph G, list vertex_pointer,
                                   bool   (*p_is_reachable)(),
                                   status (*p_action_vertex_before)(),
                                   status (*p_action_edge_before)(),
                                   status (*p_action_edge_after)());
int     degree(list vertex_pointer);
status  mark_all_unvisited(graph G);
status  mark_senses_down(graph G, list vertex_pointer, bool (*p_is_reachable)() );
void    clear_all_senses(graph G);
status  find_fundamental_cycles(graph G, list vertex_pointer,
                                bool (*p_is_reachable)(), list* cycles);
status  detect_fundamental_cycles(graph G, list* cycles,
                                  bool (*p_is_reachable)());
status  cycle_search(list curr_vertex, list curr_edge, list end_vertex, int label,
                     list* vertices, list* edges);

#endif
```

H.30 glr.c

```
/*
 * GLR 0.1
 * Graph Library v. 0.1
 *
 * glr_graph.c
 * Copyright: (c) Lluis Ros 1995, 1996
 * Creation: April 1995
 * Last revision: 5 March 1996
 *
 * What are LLR & GLR?
 *
 *    LLR
 *      is a library of functions to handle lists.
 *
 *    GLR
 *      is a library of functions to handle (undirected) graphs. The
 *      implementation of GLR's functions makes use of many (low level)
 *      functions included in LLR. The functions are compiled and grouped
 *      in libglr.a. Basically, the routines in GLR allow to create a
 *      (non-directed) graph and work with it: create/destroy nodes/branches,
 *      perform depth-first searches, find the set of fundamental cycles,
 *      count the number of spanning trees in a graph, and so on.
 *
 *    Library:
 *      llr_list.c and glr_graph.c are compiled and grouped in libllrglr.a
 *
 *
 * Who designed the code in LLR & GLR?
 *
 *    LLR:
 *      The major part of the code in this library has been taken from the book by
```

```
 *       Jeffrey Esakov & Tom Weiss "Data Structures, an Advanced Approach Using C".
 *       These are: allocate_node, free_node, init_list, empty_list, insert, append delete,
 *       delete_node, find_key, list_iterator, destroy_list.
 *       The rest of routines are mine (blame me for their errors :-) ):concatenate, length,
 *       print_prolog_list.
 *
 *    GLR:
 *       All of them are mine, although -as you'll see- they follow Esakov's programming
 *       style. The code in GLR implements most of Esakov's basic primitive operations over the
 *       'graph abstract data type' and many others I added. The main difference between my
 *       implementation of the basic primitive operations and Esakov's is that I do not assume
 *       a "static" vertex set. That is: you may add or remove vertices from the graph as the
 *       computations go on.
 *       To do this, the set of vertices has been stored in a dynamic list rather than in a
 *       statically-dimensioned vector as in Esakov's book. Another difference is that I allow
 *       vertices to have the label you want (in Esakov's implementation they must be numbered
 *       from 1 to n). Finally, both edges and vertices may store information (only edges can in
 *       Esakov's).
 *
 * Acknowledgements:
 *
 *    Special thanks to Pablo Jimenez, Manel Guerris and Albert Castellet for using LLRß 0.1 &
 *    GLRß 0.1 and act asß -testers.
 *    Pablo Jimenez suggested that the code in glr_graph.c could be used to handle digrafs.
 *    I'll try to generalise the implementation of these functions to allow this.
 *    Pablo also suggested many good improvements on the documentation that comes with LLRß 0.1
 *    & GLRß 0.1. Manel found a very subtle "potential bug" in find_fundamental_cycles().
 *
 * Pendent:
 * - cycle_search() ha de replicar els camps de dades dels vertexs i els arcs, i no guardar
 *   punters a ells.
 * - revisar quan es fa backtracking en cycle_search().
 * - com podem emprar aquestes rutines per a digrafs?
 * - documentar les funcions print_***
 */


#include <stdio.h>
#include <malloc.h>
#include "general.h"
#include "utils.h"
#include "llr.h"
#include "glr.h"


/*-------------------------------------------------------------------------------------------------------------------*/
status init_graph(graph* p_G)
{
    return(init_list(p_G));
}
/*-------------------------------------------------------------------------------------------------------------------*/
void destroy_graph(graph* p_G)
/*
 * Destroys the graph pointed to by *p_G. *p_G is set to the NULL value.
 * Vertex and edge lists are deleted by means of the list primitive
 * operation destroy_list. Destroy_list takes as an argument the name
 * of the function to be used to remove the list's data field. We use:
 *
 *     free_vertex_data()  for the vertex data field
 *     free_edge_data()    for the edge data field.
 *
 * Also, in order to be as open as possible, and respect the polimorphic
 * conception of these graph handling routines, we give the liberty to the
 * user to define his(her) own specific vertex/edge data field, and hence
 * we  provide a means to specify how these fields must be removed. This
 * is accomplished by  free_vertex_data() and free_edge_data() by calling
 * (respectively) these two functions:
 *
 *              free_specific_vertex_data(generic_ptr p_data)
 *              free_specific_edge_data(generic_ptr p_data)
 *
 * See also: free_edge_data() and free_vertex_data().
 */
{
    list curr=NULL;

    /*
      * we first destroy all lists of edges. Note that destroy_list
      * already sets the contents of its first argument (the address
      * of the pointer to the first element of the list) to NULL
      * (see destroy_list in llr_list.c).
      */
    while( (curr=list_iterator(*p_G, curr))!=NULL)
      destroy_list(&EMANATING(curr), free_edge_data);

    /*
      * We destroy the list of vertices. *p_G is implicitly set to
      * NULL in destroy_list .
      */
    destroy_list(p_G, free_vertex_data);
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

```
status add_vertex(graph* p_G, vertex vertex_number, generic_ptr specific_data)
{
    vertex_data* p_vertexdata;
    static int vertex_index=0;

    /* update vertex index */
    vertex_index++;

    /* allocate memory space for the data stored in this vertex */
    p_vertexdata=(vertex_data*)malloc(sizeof(vertex_data));
    if(p_vertexdata==NULL)
        return ERROR;

    /* store the data */
    p_vertexdata->specific_data=specific_data;
    p_vertexdata->emanating=NULL;
    p_vertexdata->vertex_number=vertex_number;
    p_vertexdata->vertex_index=vertex_index;
    p_vertexdata->visited=FALSE;
    p_vertexdata->label=NOLABEL;
    p_vertexdata->labels=NULL;

    /* append the node corresponding to this vertex to the list of vertices */
    if(append(p_G, (generic_ptr)p_vertexdata)==ERROR) {
        free(p_vertexdata);
        p_vertexdata=NULL;
        return ERROR;
    }
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------------------*/
status add_edge(graph G, vertex vertex1, vertex vertex2, generic_ptr specific_data)
{
    list pv1=NULL;
    list pv2=NULL;

    if( (pv1=find_vertex(G, vertex1))==NULL) return ERROR;
    if( (pv2=find_vertex(G, vertex2))==NULL) return ERROR;

    if(add_edge_with_pointers(pv1, pv2, specific_data)==ERROR)
      return ERROR;

    return OK;
}



/*-------------------------------------------------------------------------------------------------------------------------------*/
status add_edge_with_pointers(list pv1, list pv2, generic_ptr specific_data)
{
  vertex vertex1, vertex2;

  if(pv1==NULL || pv2==NULL) return ERROR;

  vertex1 = VERTEX_NUMBER(pv1);
  vertex2 = VERTEX_NUMBER(pv2);

  if(edge_append(&EMANATING(pv1), vertex1, vertex2, specific_data, pv2, 1)==ERROR)
    return ERROR;
  if(edge_append(&EMANATING(pv2), vertex2, vertex1, specific_data, pv1, 2)==ERROR)
    return ERROR;

  return OK;
}

status add_edge2(graph G, vertex vertex1, vertex vertex2, generic_ptr specific_data)
{
    list pv1=NULL;
    list pv2=NULL;

    if( (pv1=find_vertex(G, vertex1))==NULL) return ERROR;
    if( (pv2=find_vertex(G, vertex2))==NULL) return ERROR;

    if(add_edge_with_pointers2(pv1, pv2, specific_data)==ERROR)
      return ERROR;

    return OK;
}

status add_edge_with_pointers2(list pv1, list pv2, generic_ptr specific_data)
{
  vertex vertex1, vertex2;

  if(pv1==NULL || pv2==NULL) return ERROR;

  vertex1 = VERTEX_NUMBER(pv1);
  vertex2 = VERTEX_NUMBER(pv2);

  if(edge_append(&EMANATING(pv1), vertex1, vertex2, specific_data, pv2, 1)==ERROR)
    return ERROR;
  return OK;
```

```
}

/*-----------------------------------------------------------------------------------------------------------*/
status edge_append(list* p_L,
                   vertex vertex_ini,
                   vertex vertex_fi,
                   generic_ptr specific_data,
                   list vertex_pointer, int copy)
{
    edge_data* p_edgedata;

    p_edgedata=(edge_data*)malloc(sizeof(edge_data));
    if(p_edgedata==NULL) return ERROR;

    p_edgedata->specific_data=specific_data;
    p_edgedata->vertex_ini=vertex_ini;
    p_edgedata->vertex_number=vertex_fi;
    p_edgedata->vertex_pointer=vertex_pointer;
    p_edgedata->copy=copy;
    p_edgedata->sense=NOSENSE;


    if(append(p_L,(generic_ptr)p_edgedata)==ERROR) {
        free(p_edgedata);
        p_edgedata=NULL;
        return ERROR;
    }
    return OK;
}
/*-----------------------------------------------------------------------------------------------------------*/
list find_vertex(graph G, vertex vertex_number)
{
    list vertex_pointer;
    vertex_data v;

    v.vertex_number=vertex_number;
    if(find_key(G, (generic_ptr)&v, cmp_vertex, &vertex_pointer)==ERROR)
        return(NULL);

    return(vertex_pointer);
}
/*-----------------------------------------------------------------------------------------------------------*/
int cmp_vertex(generic_ptr p_datapointer1, generic_ptr p_datapointer2)
{
    return( ((vertex_data*)p_datapointer1)->vertex_number -
            ((vertex_data*)p_datapointer2)->vertex_number );
}
/*-----------------------------------------------------------------------------------------------------------*/
status delete_edge(graph G, vertex vertex1, vertex vertex2)
{
    list origin1, origin2;
    list destin1, destin2;

    if( (origin1=find_vertex(G, vertex1))==NULL)
        return ERROR;
    if( (origin2=find_vertex(G, vertex2))==NULL)
        return ERROR;
    if( (destin1=find_destin(EMANATING(origin1), vertex2))==NULL)
        return ERROR;
    if( (destin2=find_destin(EMANATING(origin2), vertex1))==NULL)
        return ERROR;

    if(delete_edge_with_pointers(origin1, destin1, origin2, destin2)==ERROR)
      return ERROR;

    return OK;
}
/*-----------------------------------------------------------------------------------------------------------*/
status delete_edge_with_pointers(list origin1, list destin1, list origin2, list destin2)
/*
 * Deletes an edge from G through its pointers.
 * origin1 (origin2): pointer to the first (second) vertex of the edge
 * destin1 (destin2): pointer to the edge, seen as emanating from origin1 (origin2).
 */
{
    if(delete_node(&EMANATING(origin1), destin1, free_edge_data)==ERROR)
        return ERROR;
    if(delete_node(&EMANATING(origin2), destin2, free_edge_data)==ERROR)
        return ERROR;
    return OK;
}
/*-----------------------------------------------------------------------------------------------------------*/
list find_destin(list L, vertex v)
{
    list curr=NULL;
    while( (curr=list_iterator(L, curr))!=NULL)
        if(DESTINATION(curr)==v)
            return curr;
    return NULL;
}
/*-----------------------------------------------------------------------------------------------------------*/
```

```
list find_destin_with_pointer(list L, list pv)
{
    list curr=NULL;
    while( (curr=list_iterator(L, curr))!=NULL)
        if(VERTEX_POINTER(curr)==pv)
            return curr;
    return NULL;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status delete_vertex(graph* p_G, vertex vertex_number)
{
    list vertex_pointer=NULL;
    if((vertex_pointer=find_vertex(*p_G, vertex_number))==NULL)
        return ERROR;
    if(!empty_list(EMANATING(vertex_pointer)))
        return ERROR;

    if(delete_node(p_G,vertex_pointer,free_vertex_data)==ERROR)
        return ERROR;
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status delete_vertex_with_pointer(graph* p_G, list vertex_pointer)
{
    if(vertex_pointer==NULL || (*p_G)==NULL)
        return ERROR;
    if(!empty_list(EMANATING(vertex_pointer)))
        return ERROR;

    if(delete_node(p_G,vertex_pointer,free_vertex_data)==ERROR)
        return ERROR;
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
void free_edge_data(generic_ptr data)
/*
 * Frees all space occupied by the data linked to an edge.
 * The space occupied by specific_edge_data is also freed by calling free_specific_edge_data().
 * Remember that free(pointer) does not set pointer to NULL, hence we must do it. This is accomplished
 * inside destroy_list(), where data is set to NULL.
 */
{
  edge_data* garbage;
  list edge;

  garbage = (edge_data*)data;
  if(garbage!=NULL) {
    free_specific_edge_data(garbage->specific_data);
    garbage->specific_data=NULL;
    /* We've just freed specific_data, but this is a block of memory which is still accessible
       * from the other representative of this edge. We mend this up by setting "the other"
       * specific_data field to NULL, if it still exists.
       */
    if((edge=find_destin(EMANATING(garbage->vertex_pointer),
                         garbage->vertex_ini))!=NULL) {
      ((edge_data*)DATA(edge))->specific_data=NULL;
    }
    garbage->vertex_pointer=NULL;
    free(data);
  }
}
/*-------------------------------------------------------------------------------------------------------------------*/
void free_vertex_data(generic_ptr data)
/*
 * Frees all space occupied by the data linked to a vertex.
 * The list of emanating edges is not destroyed now, but the pointer emanating
 * is set to NULL and thus cannot be used to access the emanating edges anymore.
 * The list of labels is completely destroyed.
 * The space occupied by specific_vertex_data is also freed by calling free_specific_vertex_data().
 */
{
  vertex_data* garbage = (vertex_data*)data;

  if(garbage!=NULL) {
    destroy_list(&(garbage->labels), free);
    free_specific_vertex_data(garbage->specific_data);
    garbage->emanating=NULL;
    garbage->specific_data=NULL;
    free(data);
    /* Obs: garbage is set to NULL in the calling function, destroy_list */
  }
}
/*-------------------------------------------------------------------------------------------------------------------*/
void free_cycles_data(generic_ptr data)
/*
 * Frees all the space occupied by the data linked to a list of cycles.
 * Each node of this list contains information relative to one cycle.
 * This info is stored in the data field of the node, and consists of:
 *     a list of vertices of the cycle
 *     a list of edges of the cycle
 *     an integer label identifying the cycle
 */
```

```
{
  cycle_data* garbage = (cycle_data*)(data);

  if(garbage!=NULL) {
    /* compte, no hem de destruir el camp de dades de la llista edges
      *  o vertices, caré s precisament el camp de dades d'un vertex del
      * graf, i el graf el volem deixar intacte. Per aixo especifiquem
      * NULL com a funcio de destruccio de dades.
      */
    destroy_list(&(garbage->vertices), NULL);
    destroy_list(&(garbage->edges), NULL);
    free(data);
  }
}

/*-------------------------------------------------------------------------------------------------------------------*/
status mark_all_unvisited(graph G)
{
    list vertex;
    vertex=NULL;
    if(G==NULL) return ERROR;
    while((vertex=list_iterator(G, vertex))!=NULL)
        VISITED(vertex)=FALSE;
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
void clear_all_senses(graph G)
{
  list V=NULL;
  list E=NULL;

  while((V=list_iterator(G,V))!=NULL)
    while((E=list_iterator(EMANATING(V),E))!=NULL)
      SENSE(E)=NOSENSE;
}
/*-------------------------------------------------------------------------------------------------------------------*/
int degree(list vertex_pointer)
{
    int degree=0;
    list edge_pointer;
    list L;

    if(vertex_pointer==NULL)
        return -1;

    L=EMANATING(vertex_pointer);
    edge_pointer = NULL;
    while((edge_pointer=list_iterator(L, edge_pointer))!=NULL)
        degree++;

    return degree;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status detect_fundamental_cycles(graph G, list* cycles, bool (*p_is_reachable)() )
/*
 * Obtains a set of fundamental cycles on the graph G.
 * The output parameter "cycles" is a "list of cycles". Each
 * element of the list stores the edges and vertices of a fundamental cycle.
 *
 * Warning: it always returns OK.
 */
{
  /* give a sense to each one of the edges */
  mark_all_unvisited(G);
  clear_all_senses(G);
  mark_senses_down(G, G, p_is_reachable);

  /* reset the visited field to FALSE */
  mark_all_unvisited(G);

  /* call the recursive function that detects all cycles of G */
  if(find_fundamental_cycles(G, G, p_is_reachable, cycles)==ERROR)
    return ERROR;

  return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status find_fundamental_cycles(graph G,
                               list curr_vertex,
                               bool (*p_is_reachable)(),
                               list* cycles)
/*
 * Recursive function.
 * Given a connected graph G, this function finds a set of fundamental cycles of G.
 * This set is returned via the "cycles" argument, which is a list.
 * Each node (pointed to by pNode) of the cycles list stores the following information related to
 * one cycle:
 *
 *      CYC_VERTICES(pNode): a list of the vertices of the cycle
 *      CYC_EDGES(pNode)   : a list of the edges of the cycle
 *      CYC_LABEL(pNode)   : the label of this cycle. After 'find_fundamental_cycles' is
 *                           done, all vertices along the cycle will remain labelled with
```

```
 *                              CYC_LABEL(pNode). This label being stored in the field 'labels'
 *                              of vertex_data structure.
 */
{
    list edge_pointer;
    static int label=0;
    list vertices, edges;
    cycle_data* p_cycle_data;
    list adj_vertex_pointer;
    bool is_reachable;

    if(G==NULL || curr_vertex==NULL)
      return ERROR;

    /* printf("visiting %u\n", VERTEX_NUMBER(curr_vertex));*/
    VISITED(curr_vertex)=TRUE;

    /*
     * execute a depth-first travel along  the graph searching for already
     * visited vertices which determine the existence of a cycle along them.
     */
    edge_pointer = NULL;
    while((edge_pointer=list_iterator(EMANATING(curr_vertex), edge_pointer))!=NULL)  {

      /* get pointer to the current adjacent vertex */
      adj_vertex_pointer = VERTEX_POINTER(edge_pointer);

      /* see if the current adjacent vertex is reachable under the (eventual) user's criterion */
      if(p_is_reachable == NULL)
        is_reachable = TRUE;
      else
        is_reachable = (*p_is_reachable)(edge_pointer, adj_vertex_pointer);

      /* if adjacent vertex is reachable and not yet visited ... */
      if(is_reachable){
        if(!VISITED(adj_vertex_pointer)) {
          if(find_fundamental_cycles(G, adj_vertex_pointer, p_is_reachable, cycles)==ERROR)
            return ERROR;
        }
        else if(SENSE(edge_pointer)==DOWN)         {
          /*
           * verify that this sense is down!! the edge could be the one leading
           * to the parent node from where we are coming!!
           */

          /* new cycle detected */
          label++;

          /* printf("Found a cycle. Cycle %d.\n", label); fflush(stdout); */
          /* initialize lists "vertices" and "edges" */
          if(init_list(&vertices)==ERROR || init_list(&edges)==ERROR)
            cerror("Error: init_list() failed in find_fundamental_cycles()");

          /*
           * go backwards to reach the vertex pointed to by
           * adj_vertex_pointer once again
           */
          if(cycle_search(curr_vertex, edge_pointer,
                          adj_vertex_pointer,
                          label,
                          &vertices,
                          &edges)==ERROR)
            cerror("Error: cycle_search() failed in find_fundamental_cycles()");

          /* store cycle in cycles list */
          if((p_cycle_data=(cycle_data*)malloc(sizeof(cycle_data)))==NULL)
            cerror("Error: malloc() failed in find_fundamental_cycles()\n");
          p_cycle_data->vertices=vertices;
          p_cycle_data->edges=edges;
          p_cycle_data->label=label;
          if(append(cycles, (generic_ptr)p_cycle_data)==ERROR)
            cerror("Error: append() failed in find_fundamental_cycles()\n");
        }
      }
    }
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------*/
status cycle_search(list curr_vertex,
                    list curr_edge,
                    list end_vertex,
                    int label,
                    list* vertices, list* edges)
/*
 * Recursive function.
 * Given the vertex "curr_vertex" cycle_search() goes upwards in the graph until it finds
 * the vertex "end_vertex". Along the way, cycle_search() inserts "curr_vertex" in the
 * "vertices" list and "curr_edge" in the "edges" list. cycle_search() is used to search
 * for all edges and vertices of a cycle, once its presence has been detected by
 * find_fundamental_cycles().
 *
 * The lists "vertices" and "edges" must have been initialized by the calling function.
```

```
 * Order inside vertices (edges): The cycle vertices (edges) are stored in the oposite
 * sense in which they are found during the cycle's tracing.
 */
{
  list edge;
  static bool found=FALSE;
  int* p_label_data;

  if(curr_vertex==NULL) return ERROR;

  LABEL(curr_vertex)=label;


  /* insert current vertex and last edge in vertices and edges */
  if(insert(vertices, (generic_ptr)(DATA(curr_vertex)))==ERROR)
    cerror("Error: insert() failed to insert vertex in cycle_search()");
  if(insert(edges, (generic_ptr)(DATA(curr_edge)))==ERROR)
    cerror("Error: insert() failed to insert edge in cycle_search()");
  /* insert current label in the labels list associated to this vertex */
  if((p_label_data=(int*)malloc(sizeof(int)))==NULL)
    cerror("Error: malloc() failed in cycle_search()");
  (*p_label_data)=label;
  if(insert(&LABELS(curr_vertex), (generic_ptr)p_label_data)==ERROR)
    cerror("Error: insert() failed to insert label in cycle_search()");

  found=(curr_vertex==end_vertex);

  edge = NULL;
  while((edge=list_iterator(EMANATING(curr_vertex), edge))!=NULL && !found)
  {
    /* printf("exploring through edge to %u\n", DESTINATION(edge));*/
    if(SENSE(edge)==UP && LABEL(VERTEX_POINTER(edge))!=label)
      if(cycle_search(VERTEX_POINTER(edge), edge, end_vertex, label, vertices, edges)==ERROR)
        return ERROR;
  }
  if(!found)
  {
    /*
     * There is only one case in which the last while loop cannot find "end_vertex".
     * This happens when the upwards travel along G enters a cycle inside the cycle we
     * are following (see documentation). We must provide backtracking capability
     * to go on with the search.
     */

    LABEL(curr_vertex)=NOLABEL;
    delete_node(vertices, *vertices, NULL);
    delete_node(edges, *edges, NULL);
    delete_node(&(LABELS(curr_vertex)), LABELS(curr_vertex), free);
  }
  return OK;
}
/*-------------------------------------------------------------------------------------------------------------*/
status mark_senses_down(graph G, list vertex_pointer, bool (*p_is_reachable)())
/*
 * Preconditions on first entry: it requires that all sense fields at all edges be equal to NOSENSE.
 * Marks the relative position of a vertex when seen from another.
 * The relative position of vertex v is DOWN from vertex u if edge (u,v) is traversed
 * from u to v in a depth-first search, otherwise it is UP. The attribute UP or DOWN
 * is stored in the nodes of the EMANATING list of a particular vertex.
 * Initially, all edges have to be labeled with sense NOSENSE. See edge_append().
 */
{
  list edge_pointer;
  list other_edge_pointer;
  list adj_vertex_pointer;
  bool is_reachable;

  /* protection against the case of NULL pointers */
  if(G==NULL || vertex_pointer==NULL)
    return ERROR;

  VISITED(vertex_pointer)=TRUE;

  edge_pointer = NULL;
  while((edge_pointer=list_iterator(EMANATING(vertex_pointer), edge_pointer))!=NULL)
  {
    /* get pointer to the current adjacent vertex */
    adj_vertex_pointer = VERTEX_POINTER(edge_pointer);

    /* see if the current adjacent vertex is reachable under the user's criterion (if provided) */
    if(p_is_reachable == NULL)
      is_reachable = TRUE;
    else
      is_reachable = (*p_is_reachable)(edge_pointer, adj_vertex_pointer);

    if(is_reachable) {

      /* mark sense of this edge. The sense is DOWN at the current edge_pointer, and UP in the other one  */
      if(SENSE(edge_pointer)==NOSENSE) SENSE(edge_pointer)=DOWN;
      if((other_edge_pointer=find_destin(EMANATING(adj_vertex_pointer),
                                         VERTEX_NUMBER(vertex_pointer)))==NULL)
        cerror("Error: find_destin() failed in mark_senses_down()");
```

```
      if(SENSE(other_edge_pointer)==NOSENSE) SENSE(other_edge_pointer)=UP;

      /* go on with the DFS */
      if(!VISITED(adj_vertex_pointer))
        if(mark_senses_down(G, adj_vertex_pointer, p_is_reachable)==ERROR)
          return ERROR;
    }
  }
  return OK;
}

/*----------------------------------------------------------------------------------------------------------------------*/
status depth_first_vertices_edges(graph G, list vertex_pointer,
                                  bool   (*p_is_reachable)(),
                                  status (*p_action_vertex_before)(),
                                  status (*p_action_edge_before)(),
                                  status (*p_action_edge_after)())
/*
 * Visits all vertices and edges of the graph G, in a depth-first order, and performs the following
 * actions on the edges and vertices found along the way (if they are not NULL):
 *
 *     action vertex (*p_action_vertex_before)(): is an action on the current vertex (pointed to by
 *            vertex_pointer), in the descending phase.
 *     action before (*p_action_edge_before)(): is an action on the current edge (pointed to by
 *            edge_pointer), in the descending phase.
 *     action after (*p_action_edge_after)():   is an action on the current edge in the ascending
 *            phase, once all vertices of the graph have been reached.
 *
 * If some of these actions is specified as a NULL pointer, nothing is done.
 * The function traverses all arcs of the graph that "are traversable". The user must
 * provide a function (*p_is_reachable)() to determine whether a given edge is traversable.
 *
 * Preconditions: at the top-level call all vertices must be marked as non-visited.
 */
{
    list edge_pointer;
    list adj_vertex_pointer;
    bool is_reachable;

    adj_vertex_pointer=NULL;

    /* protection against the case of NULL pointers */
    if(G==NULL || vertex_pointer==NULL)
        return ERROR;

    /* mark the current vertex as visited and do action_vertex_before() on it, if specified */
    VISITED(vertex_pointer)=TRUE;
    if(p_action_vertex_before != NULL)
      (*p_action_vertex_before)(vertex_pointer);

    /* for all neighboring vertices of vertex_pointer ... */
    edge_pointer = NULL;
    while((edge_pointer=list_iterator(EMANATING(vertex_pointer), edge_pointer))!=NULL){

      /* get pointer to the current adjacent vertex */
      adj_vertex_pointer = VERTEX_POINTER(edge_pointer);

      /* see if the current adjacent vertex is reachable under the user's criterion (if provided) */
      if(p_is_reachable == NULL)
        is_reachable = TRUE;
      else
        is_reachable = (*p_is_reachable)(edge_pointer, adj_vertex_pointer);

      /* if adjacent vertex is reachable and not yet visited ... */
      if(is_reachable && !VISITED(adj_vertex_pointer)){
        /* do action_edge_before() on the edge */
        if(p_action_edge_before !=NULL)
          (*p_action_edge_before)(vertex_pointer, edge_pointer, adj_vertex_pointer);
        /* visit the vertex and its descendants ... */
        if(depth_first_vertices_edges(G, adj_vertex_pointer,
                                      p_is_reachable,
                                      p_action_vertex_before,
                                      p_action_edge_before,
                                      p_action_edge_after)==ERROR){
          return ERROR;
        }
        /* do action_edge_after() on the edge */
        if(p_action_edge_after !=NULL)
          (*p_action_edge_after)(vertex_pointer, edge_pointer, adj_vertex_pointer);
      }
    }
    return OK;
}
/*----------------------------------------------------------------------------------------------------------------------*/
void free_specific_vertex_data(generic_ptr p_data)
{
}
/*----------------------------------------------------------------------------------------------------------------------*/
void free_specific_edge_data(generic_ptr p_data)
{
}
/*----------------------------------------------------------------------------------------------------------------------*/
```

H.31 glrtools.h

```
#ifndef GLRTOOLS_H
#define GLRTOOLS_H

int    count_vertices(graph G);
int    count_edges(graph G);
void   print_graph_skeleton(graph G);
void   print_label_id(FILE* fd, list label_pointer);
void   print_vertex_id(FILE* fd, list vertex_pointer);
void   print_list_vertices(FILE* fd, list vertices);
void   print_list_cycles(list cycles);
status simply_visit_all(graph G);
status depth_first(graph G, list vertex_pointer, status (*p_visit_f)());

#endif
```

H.32 glrtools.c

```
#include <stdio.h>
#include <malloc.h>
#include "general.h"
#include "utils.h"
#include "llr.h"
#include "glr.h"
#include "llrtools.h"
#include "glrtools.h"



/* private functions' prototypes */
static status simply_visit(list vertex_pointer);

/*-------------------------------------------------------------------------------------------------------------------*/
int count_vertices(graph G)
{
  list V;
  int count=0;

  V=NULL;
  while((V=list_iterator(G,V))!=NULL)
    count++;
  return(count);
}
/*-------------------------------------------------------------------------------------------------------------------*/
int count_edges(graph G)
{
  list V, E;
  int count = 0;

  for(V=G; V!=NULL; V=NEXT(V))
    for(E=EMANATING(V); E!=NULL; E=NEXT(E))
      count ++;
  return (count/2);
}
/*-------------------------------------------------------------------------------------------------------------------*/
void print_graph_skeleton(graph G)
/*
 * ASCII output to the terminal.
 * Prints only the basic structure of the graph G. It does not print information
 * contained in the specific_*_data typed fields of the vertices and edges.
 */
{
    list V;

    if(G==NULL)
    {
        printf("Empty graph in print_graph()\n");
        return;
    }
    printf("\n\nGraph map\n");

    V=NULL;
    while((V=list_iterator(G, V))!=NULL)
    {

        printf("[%u, %u]", VERTEX_NUMBER(V), VERTEX_INDEX(V));
        print_prolog_list(stdout, LABELS(V), print_label_id);
        printf("->");
        /* print the list of EMANATING vertices */
        {
            list E=NULL;
            printf("[vdest,sense]: ");
            while((E=list_iterator(EMANATING(V), E))!=NULL)
                printf("[%u,%d] ",
                       DESTINATION(E),
                       SENSE(E));
        }
        printf("\n");

    }
```

```
    printf("\n");
}
/*-------------------------------------------------------------------------------------------------------------------*/
void print_label_id(FILE* fd, list label_pointer)
{
  fprintf(fd, "\'%d\'", THISLABEL(label_pointer));
}
/*-------------------------------------------------------------------------------------------------------------------*/
void print_vertex_id(FILE* fd, list vertex_pointer)
{
  if(DATA(vertex_pointer)!=NULL)
    fprintf(fd, "\'%u\'", VERTEX_NUMBER(vertex_pointer));
  else
    cerror("Fatal error:: print_vertex_id() failed, there's no data stored in this vertex!!");
}
/*-------------------------------------------------------------------------------------------------------------------*/
void print_list_vertices(FILE* fd, list vertices)
{
  print_prolog_list(fd, vertices, print_vertex_id);
}
/*-------------------------------------------------------------------------------------------------------------------*/
void print_list_cycles(list cycles)
/*
 * Prints the contents of the list cycles, the one obtained after detect_fundamental_circuits()
 */
{
  list curr_C;
  int n=0;

  if(cycles == NULL)
    printf("There are no cycles\n");
  for(curr_C=cycles; curr_C!=NULL; curr_C=NEXT(curr_C))
  {
    printf("Cycle %d\n", n);
    printf("-----------------------------\n");
    printf("list of vertices=");
    print_list_vertices(stdout, CYC_VERTICES(curr_C));
    printf("\n");

    /*
     * The edges do not have an identifier by the moment. The following can only be done
     * if we link with cinout.c
     */
    /*     printf("list of edges="); */
    /*     print_list_edges(stdout, CYC_EDGES(curr_C)); */

    printf("\n");
    n++;
  }
}
/*-------------------------------------------------------------------------------------------------------------------*/
static status simply_visit(list vertex_pointer)
{
    if(vertex_pointer==NULL) return ERROR;
    printf("visiting vertex %u\n", VERTEX_NUMBER(vertex_pointer));
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status simply_visit_all(graph G)
{
    return(depth_first(G, G, simply_visit)==ERROR);
}
/*-------------------------------------------------------------------------------------------------------------------*/
status depth_first(graph G, list vertex_pointer, status (*p_visit_f)())
/*
 * Preconditions: on entry, all vertices must be marked as non-visited.
 */
{
    list edge_pointer;

    /* protection against the case of NULL pointers */
    if(G==NULL || vertex_pointer==NULL)
        return ERROR;

    VISITED(vertex_pointer)=TRUE;
    if((*p_visit_f)(vertex_pointer)==ERROR)
        return ERROR;

    /* for all neighboring vertices of vertex_pointer ... */
    edge_pointer = NULL;
    while((edge_pointer=list_iterator(EMANATING(vertex_pointer), edge_pointer))!=NULL)
      /* if the vertex has not been visited */
      if(!VISITED(VERTEX_POINTER(edge_pointer)))
        /* visit the vertex and its descendants */
        if(depth_first(G, VERTEX_POINTER(edge_pointer), p_visit_f)==ERROR)
          return ERROR;

    return OK;
}
```

H.33 Gmotion.c

```
/***************************************************************
*       PUMA 560 Off-line Programming and Simulation Tool
*       (c) IRI, Institute of Industrial Robotics
*       Barcelona, Spain
*       Version 2.0
*
* Module:       Mmotion.c
* Author:       Federico Thomas
* Date:         April 1989
* Updated:      February 2001
***************************************************************/

#include "Gtypes.h"
#include "math.h"
#include "general.h"

/***************************************************************
*  equal_inside_range() verifica si los angulos de cada una de las
*                       articulaciones se hallan dentro del rango. Para
*                       aquellas articulaciones que se hallen fuera de
*                       rango calcula un angulo equivalente al que posee
*                       la articulacion pero que se encuentre dentro del
*                       rango de validez angular para dicha articulacion
*****
***************************************************************/

void equal_inside_range(JNTS* j,JNTS jmin_c,JNTS jmax_c)
{
  int njoint;

  for (njoint = 1; njoint <= 6; njoint++){
    switch(njoint) {
    case 1 :  if ( (j->th1 < jmin_c.th1) || (j->th1 < (-PIT2)) )
      j->th1 += PIT2;
      if ( (j->th1 > jmax_c.th1) || (j->th1 > PIT2) )
        j->th1 -= PIT2;
      break;
    case 2 :  if ( (j->th2 < jmin_c.th2) || (j->th2 < (-PIT2)) )
      j->th2 += PIT2;
      if ( (j->th2 > jmax_c.th2) || (j->th2 > PIT2) )
        j->th2 -= PIT2;
      break;
    case 3 :  if ( (j->th3 < jmin_c.th3) || (j->th3 < (-PIT2)) )
      j->th3 += PIT2;
      if ( (j->th3 > jmax_c.th3) || (j->th3 > PIT2) )
        j->th3 -= PIT2;
      break;
    case 4 :  if ( (j->th4 < jmin_c.th4) || (j->th4 < (-PIT2)) )
      j->th4 += PIT2;
      if ( (j->th4 > jmax_c.th4) || (j->th4 > PIT2) )
        j->th4 -= PIT2;
      break;
    case 5 :  if ( (j->th5 < jmin_c.th5) || (j->th5 < (-PIT2)) )
      j->th5 += PIT2;
      if ( (j->th5 > jmax_c.th5) || (j->th5 > PIT2) )
        j->th5 -= PIT2;
      break;
    case 6 :  if ( (j->th6 < jmin_c.th6) || (j->th6 < (-PIT2)) )
      j->th6 += PIT2;
      if ( (j->th6 > jmax_c.th6) || (j->th6 > PIT2) )
        j->th6 -= PIT2;
      break;
    }
  }
}

/***************************************************************
* explain_code()  en funcion del codigo de error "code", emite el mensaje
*                 de error correspondiente que indica cual es la
*                 articulacion origen de los problemas
***************************************************************/


void explain_code(int code)
{
 if ((code & 01)  > 0) printf("* Joint 1 out of range *\n");
 if ((code & 02)  > 0) printf("* Joint 2 out of range *\n");
 if ((code & 04)  > 0) printf("* Joint 3 out of range *\n");
 if ((code & 010) > 0) printf("* Joint 4 out of range *\n");
 if ((code & 0100)> 0) printf("* Location out of range *\n");
 if ((code & 0400)> 0) printf("* Degenerate case *\n");
 if ((code & 020) > 0) printf("* Joint 5 out of range *\n");
 if ((code & 040) > 0) printf("* Joint 6 out of range *\n");
}



/***************************************************************
 *  range()   se emplea para limitar el valor que pueden alcanzar los
 *            angulos de las diferentes articulaciones. Devuelve  un
 *            valor de "a" en el rango 0 < a <= 360
```

```
 ***************************************************************************/

Greal range(Greal a)
{
  //if (a > 15. || a < -15.)
  // return(10.);

  while(a >= PIT2)
    a -= PIT2;
  while(a <0.)
    a += PIT2;
  //if (a > PIT2)
  // return(0.);
  return(a);
}

/***************************************************************************
*  jns_to_tr_n()  resuelve el problema cinematico directo
***************************************************************************/

void jns_to_tr_n(PDH p,JNTS* j,TRSF* t,char* c)
{
        Greal c1, s1, c2, s2, c23, s23, c4, s4, c5, s5, c6, s6,
        k1, k2, k3, k4, k5, k6, k7, x;

        SINCOS(s1, c1, j->th1);
        SINCOS(s2, c2, j->th2);
        SINCOS(s23, c23, j->th2 + j->th3);
        SINCOS(s4, c4, j->th4);
        SINCOS(s5, c5, j->th5);
        SINCOS(s6, c6, j->th6);

        k1  = c23 * s5;
        k2  = c4 * c5 * s6 + s4 * c6;
        k3  = - c23 * k2 + s23 * s5 * s6;
        k4  = - s4 * c5 * s6 + c4 * c6;
        k5  = s4 * s5;
        k6 = c23 * c4 * s5 + s23 * c5;
        k7 = p.d4 * s23 + p.a3 * c23 + p.a2 * c2;

        t->o.x = c1 * k3 - s1 * k4;
        t->o.y = s1 * k3 + c1 * k4;
        t->o.z = s23 * k2 + k1 * s6;

        t->a.x = c1 * k6 - s1 * k5;
        t->a.y = s1 * k6 + c1 * k5;
        t->a.z = - s23 * c4 * s5 + c23 * c5;

        t->p.x = c1 * k7 - s1 * p.d3;
        t->p.y = s1 * k7 + c1 * p.d3;
        t->p.z = p.d4 * c23 - p.a3 * s23 - p.a2 * s2;


        /* n = o X a */

        t->n.x = t->o.y * t->a.z - t->o.z * t->a.y;
        t->n.y = t->o.z * t->a.x - t->o.x * t->a.z;
        t->n.z = t->o.x * t->a.y - t->o.y * t->a.x;

        x = range(atan2(t->p.y, t->p.x) - j->th1);
        c[0] = (x < PIB2 || x > PI + PIB2) ? 'l' : 'r';

        x = (c[0] =='r') ? 1.: -1.;
        c[1] = ((x*cos(j->th3))<0.) ? 'd' :'u';      /* FT  04/89  */

/*      x = range(p.aa3d4 - j->th3 );
        (c)[1] = ((x < PIB2 || x > PI + PIB2) &&
                        (c[0] == 'l')) ? 'd' : 'u'; */

        c[2] = ((range(j->th5) > PI) || (j->th5 == 0)) ? 'f' : 'n';   /* CM 04/89 */
        c[3] = '\0';
        trslm(t,0.,0.,p.d6);  /* Transformacion de TOOL */
        //printf("tipus_de_configuracio\n");
        //printf("%c_%c_%c_%c\n",c[0],c[1],c[2],c[3]);

}


/***************************************************************************
 * range_j6()   limita el angulo de giro "a" de la sexta articulacion
 *              dejandolo en un rango comprendido entre 0 <= a < 798
 ***************************************************************************/

Greal range_j6(Greal a)
{
  if (a > 15. || a < -15.)
   return(10.);

  while(a >= (DEGTORAD * 540))
    a -= (DEGTORAD * 540);
  while(a < 0.)
    a += (DEGTORAD * 540);
```

```
  return(a);
}

/***************************************************************************
 *  tr_to_jns_n()  resuelve el problema cinematico inverso
 ***************************************************************************/

int tr_to_jns_n(TRSF* tr,PDH p,char c[4],JNTS* j,JNTS_RANGE* jr)
{
  int code = 0;
  Greal t1d, f11p, d, t3d, f13a, f11a, f31a, f13o, f11o, w1, w2, wd,
    s1, c1, s23, c23, s3, c3, s4, c4, s5, c5, s6, c6,d32,e,
    px, py, pz, ax, ay, az, ox, oy, oz,
    t1, t2, t3, t4, t5, t6;

  static Greal t4o;
  static boolean lefty = FALSE, up = TRUE, flip = FALSE;
  TRSF t;
  t = *tr;


  trslm(&t,0.,0.,-p.d6);  /* Transformacion inversa de TOOL */
  while (*c) {
    switch (*c++) {
    case 'l' :
      lefty = TRUE;
      break;
    case 'r' :
      lefty = FALSE;
      break;

    case 'u' :
      up = TRUE;
      break;

    case 'd' :
      up = FALSE;
      break;

    case 'f' :
      flip  = TRUE;
      break;

    case 'n' :
      flip  = FALSE;
      break;

    default  : printf("CONF_NOT_DEFINED._ERROR\n");
      break;
    }
  }
  /* printf("%i\n",lefty); */
/*   printf("%i\n",up); */
/*   printf("%i\n",flip); */


  d32=p.d3*p.d3;
  e=4.*p.a2*p.a2*p.a3*p.a3+4.*p.a2*p.a2*p.d4*p.d4;
  /* printf("valor de e\n"); */
/*   printf("%f\n",e); */
  px = t.p.x;
  py = t.p.y;
  pz = t.p.z;
  ax = t.a.x;
  ay = t.a.y;
  az = t.a.z;
  ox = t.o.x;
  oy = t.o.y;
  oz = t.o.z;

  /* theta1 */

  if ((px * px + py * py - d32) <= 0.) return(0100);

  t1d = sqrt(px * px + py * py - d32);
  t1 = atan2(py, px) - atan2(p.d3, (lefty) ? t1d : - t1d);

  j->th1 = t1;


  if (!angle_en_rang(1,t1,(Greal)(jr->jmin_c.th1),(Greal)(jr->jmax_c.th1)))
    code |= 01;


  SINCOS(s1, c1, t1);


  /* theta3 */

  f11p = c1 * px + s1 * py;
  d = f11p * f11p + pz * pz -p.d4*p.d4-p.a3*p.a3-p.a2*p.a2;
```

```
  if ((e - d * d) <= 0.) return(0100);

  t3d = sqrt(e - d * d);
 /* aquest valor aqui val zero aa3d4=atan2(p.a3,p.d4);  */
  t3 = atan2(d, (lefty ^ up) ? t3d : -t3d); /* JJA 02/85 */

  j->th3 = t3;

  //printf("el valor del angle minim3 i maxim3\n");
  //printf("%lf %lf\n",(Greal)(jr->jmin_c.th3),(Greal)(jr->jmax_c.th3));

  if (!angle_en_rang(3,t3,(Greal)(jr->jmin_c.th3),(Greal)(jr->jmax_c.th3)))
    code |= 04;


  SINCOS(s3, c3, t3);

  /* theta2 */

  w1 = p.a2 * c3 + p.a3;
  w2 = p.d4 + p.a2 * s3;
  wd = f11p * f11p + pz * pz;
  if (FABS(wd) < SMALL) {
    return(01000);
  }
  s23 = (w2 * f11p - w1 * pz) / wd;
  c23 = (w1 * f11p + w2 * pz) / wd;
  t2 = atan2(s23, c23) - t3;

  j->th2 = t2;

  //printf("el valor del angle minim2 i maxim2\n");
  //printf("%lf %lf\n",(Greal)(jr->jmin_c.th2),(Greal)(jr->jmax_c.th2));

  if (!angle_en_rang(2,t2,(Greal)(jr->jmin_c.th2),(Greal)(jr->jmax_c.th2)))
    code |= 02;

  /* theta4 */

  f13a = - s1 * ax + c1 * ay;
  f11a =   c1 * ax + s1 * ay;
  f31a = c23 * f11a - s23 * az;
  t4 = (flip) ? atan2(- f13a, - f31a) : atan2(  f13a,  f31a);

  j->th4 = t4;

  //printf("el valor del angle minim4 i maxim4\n");
  //printf("%lf %lf\n",(Greal)(jr->jmin_c.th4),(Greal)(jr->jmax_c.th4));


  SINCOS(s4, c4, t4);

  /* theta5 */

  s5 = c4 * f31a + s4 * f13a ;
  c5 = s23 * f11a + c23 * az;

  t5 = atan2(s5, c5);



  if (FABS(t5) < SMALL) {
    j->th4 = range(t4o);
    if (angle_en_rang(4,(j->th4),(Greal)(jr->jmin_c.th4),(Greal)(jr->jmax_c.th4)))
      code |= 0400;

    /*j->th4 = range(t4o - jr->jofst_c.th4);*/
    SINCOS(s4, c4, t4o);
    s5 = 0.;
    c5 = 1.;
  }
  else {
    t4o = t4;
    if (!angle_en_rang(4,t4,(Greal)(jr->jmin_c.th4),(Greal)(jr->jmax_c.th4)))
      code |= 010;
  }

  j->th5 = t5;

  //printf("el valor del angle minim5 i maxim5\n");
  //printf("%lf %lf\n",(Greal)(jr->jmin_c.th5),(Greal)(jr->jmax_c.th5));

  if (!angle_en_rang(5,t5,(Greal)(jr->jmin_c.th5),(Greal)(jr->jmax_c.th5)))
    code |= 020;


  /* theta6 */

  f13o = - s1 * ox + c1 * oy;
  f11o =   c1 * ox + s1 * oy;
  s6 = - c5 * (c4 * (c23 * f11o - s23 * oz) + s4 * f13o)
```

```
    + s5 * (s23 * f11o + c23 * oz);
  c6 = - s4 * (c23 * f11o - s23 * oz) + c4 * f13o;

  t6 = atan2(s6, c6);

  j->th6 = t6;

  //printf("el valor del angle  minim6 i maxim6\n");
  //printf("%lf %lf\n",(Greal)(jr->jmin_c.th6),(Greal)(jr->jmax_c.th6));

  if (!angle_en_rang(6,t6,(Greal)(jr->jmin_c.th6),(Greal)(jr->jmax_c.th6)))
    code |= 040;

/* CM 04/89 */
  //printf("code:%d\n",code);
  //explain_code(code);
  if (code == 0) {
    equal_inside_range(j,jr->jmin_c,jr->jmax_c);                    /* CM 05/89 */
  }
  return(code);
}


float angle_positiu(Greal a)
{
  while (a<0.)
    {
      a += PIT2;
    }
  return ((float)a);

}

Greal dangle_positiu(Greal a)
{
  //printf("l'angle abans d'entrar a dangle_positiu()\n");
  while (a<0.)
    {
      a += PIT2;
    }
  while (a>PIT2)
    {
      a-=PIT2;
    }
  //printf("l'angle despres d'entrar a dangle_positiu()\n");
  return (a);

}

Greal convertir_angle_a_pi_pi(Greal th)
{
  Greal th_pos;
  th_pos=dangle_positiu(th);
  if (th_pos>PI)
    {
      /* transformar l'angle de 0 -180*/
      th_pos=th_pos-PIT2;

    }
  /* transformar l'angle de 0 180*/
  return th_pos;
}

/* veiem que la rutina a la capcalera es independent del tractament
 * que fem a l'interior d'aquesta. Nosaltres nomes ens interesa
 * si donat un valor th, esta compres entre thmin i thmax
 */

bool angle_en_rang(int angle,Greal th,Greal thmin,Greal thmax)
{
  //printf("angle abans de fer transformacio %f\n",th);

  switch(angle)
    {
    case 1:
      /* transformacio per l'angle 1*/
      th=th-JOFST1;
      break;
    case 2:
      th=-th+JOFST2;
      break;
    case 3:
      th=-th+JOFST3;
      break;
    case 4:
      break;
    case 5:
      th=-th;
      break;
    case 6:
      break;
    default:
```

```
      return FALSE;
      break;
    }
  //printf("th%d:%f\n",angle,th);
  th=convertir_angle_a_pi_pi(th);
  //printf("th%d_pi_pi:%f\n",angle,th);

  return (angle_en_rang_pi_pi(th,thmin,thmax));

}
bool angle_en_rang_pi_pi(Greal th,Greal thmin,Greal thmax)
{
/* els angles d'entrada han d'estar entre -180 i 180
 */

  return((th>=(thmin))&&(th<=(thmax)));

}
```

H.34 Gutil.c

```
/***************************************************************************
*       PUMA 560 Off-line Programming and Simulation Tool
*       (c) IRI, Institute of Industrial Robotics
*       Barcelona, Spain                                                   *
*       Version 2.0
*
* Module:       Mutil.c
* Date:         April 1989
* Updated:      February 2001
*
***************************************************************************/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "Gtypes.h"


/*************************************************************************
*   trmult()    asigna a la  transformacion "r" el producto  de
*               transformaciones (matrices) "t" por "v" .
*               Las transformaciones "r", "t" y  "u" deben ser
*               diferentes.
*************************************************************************/

TRSF* trmult(TRSF* r, TRSF* t, TRSF* u)
{
        r->n.x = t->n.x * u->n.x + t->o.x * u->n.y + t->a.x * u->n.z;
        r->n.y = t->n.y * u->n.x + t->o.y * u->n.y + t->a.y * u->n.z;
        r->n.z = t->n.z * u->n.x + t->o.z * u->n.y + t->a.z * u->n.z;
        r->o.x = t->n.x * u->o.x + t->o.x * u->o.y + t->a.x * u->o.z;
        r->o.y = t->n.y * u->o.x + t->o.y * u->o.y + t->a.y * u->o.z;
        r->o.z = t->n.z * u->o.x + t->o.z * u->o.y + t->a.z * u->o.z;
        r->a.x = t->n.x * u->a.x + t->o.x * u->a.y + t->a.x * u->a.z;
        r->a.y = t->n.y * u->a.x + t->o.y * u->a.y + t->a.y * u->a.z;
        r->a.z = t->n.z * u->a.x + t->o.z * u->a.y + t->a.z * u->a.z;
        r->p.x = t->n.x * u->p.x + t->o.x * u->p.y + t->a.x * u->p.z + t->p.x;
        r->p.y = t->n.y * u->p.x + t->o.y * u->p.y + t->a.y * u->p.z + t->p.y;
        r->p.z = t->n.z * u->p.x + t->o.z * u->p.y + t->a.z * u->p.z + t->p.z;
        return(r);
}


/*************************************************************************
*   trmultinp()   multiplica la tranformacion "r" por "m", dejando
*                 el resultado en "r" .
*************************************************************************/

TRSF* trmultinp(TRSF* r, TRSF* m)
{
        TRSF temp;
        temp = *r;
        return(trmult(r, &temp, m));
}

/*************************************************************************
*   trsl()   asigna la traslacion definida por  "x, y, z"  a
*            las componentes  de traslacion de la transformacion "t".
*************************************************************************/

TRSF* trsl(TRSF* t, Greal x, Greal y, Greal z)
{
        t->p.x = x;
        t->p.y = y;
        t->p.z = z;
        return(t);
}


/*************************************************************************
```

```
*   trslm()   multiplica la transformacion "t" por una
*             transformacion de traslacion pura definida por
*             las componentes "x, y, z" .
*************************************************************************/

TRSF* trslm(TRSF* t, Greal x, Greal y, Greal z)
{
        static TRSF id = {{1.,0.,0.},{0.,1.,0.},{0.,0.,1.},{0.,0.,0.}};
        TRSF temp;
        temp=id;
        return(trmultinp(t, trsl(&temp, x, y, z)));
}
```

H.35 graf.h

```
#ifndef GRAF_H
#define GRAF_H

#include <stdio.h>
#include <stdlib.h>
#include "general.h"
#include "utils.h"
#include "llr.h"
#include "glr.h"
#include "llrtools.h"
#include "glrtools.h"

#endif
```

H.36 llr.h

```
#ifndef LLR_H
#define LLR_H
/*
 *  LLRß 0.1
 *  List Libraryß 0.1
 *
 *  list_types.h
 *  Copyright: (c) Lluis Ros 1995, 1996
 *  Creation: April 1995
 *  Revision: 5 March 1996
 *
 *  Declaration of list types and prototypes
 */

typedef struct node node;
typedef struct node* list;
struct node {
    generic_ptr datapointer;
    list next;
};

/*
 * List type macros
 */

#define DATA(L) ((L)->datapointer)
#define NEXT(L) ((L)->next)


/*
 * Public primitive operations prototypes of list abstract data type
 */

status  init_list(list *p_L);
bool    empty_list(list L);
status  append(list *p_L, generic_ptr data);
status  insert(list *p_L, generic_ptr data);
status  delete_first(list* p_L, generic_ptr* p_data, void (*p_func_f)());
status  delete_node(list* p_L, list node, void (*p_func_f)());
status  find_key(list L, generic_ptr key, int (*p_cmp_f)(), list* p_keynode);
list    list_iterator(list L, list lastreturn);
void    destroy_list(list* p_L, void (*p_func_f)());
int     length(list L);
list    last(list L);
list    first(list L);
void    free_node(list *p_L);

#endif
```

H.37 llr.c

```
/*
 * LLRß 0.1
 * List Libraryß 0.1
 *
 * llr_list.c
 * Copyright: (c) Lluis Ros 1995, 1996
 * Creation: April 1995
```

```
 * Revision: 5 March 1996
 *
 *
 * What are LLR & GLR?
 *
 *    LLR
 *      is a library of functions to handle lists.
 *
 *    GLR
 *      is a library of functions to handle (undirected) graphs. The
 *      implementation of GLR's functions makes use of many (low level)
 *      functions included in LLR. Basically, the routines in GLR allow to
 *      create a (non-directed) graph and work with it: create/destroy
 *      nodes/branches, perform depth-first searches, find the set of
 *      fundamental cycles, count the number of spanning trees in a graph,
 *      and so on.
 *
 *    Library:
 *      llr_list.c and glr_graph.c are compiled and grouped in libllrglr.a
 *
 *
 * Who designed the code in LLR & GLR?
 *
 *    LLR:
 *      The major part of the code in LLR has been taken from the book by
 *      Jeffrey Esakov & Tom Weiss "Data Structures, an Advanced Approach Using C".
 *      These are: allocate_node, free_node, init_list, empty_list, insert, append delete,
 *      delete_node, find_key, list_iterator, destroy_list.
 *      The rest of routines are mine (blame me for their errors :-) ):concatenate, length,
 *      print_prolog_list.
 *
 *    GLR:
 *      All of them are mine, although -as you'll see- they follow Esakov's programming
 *      style. The code in GLR implements most of Esakov's basic primitive operations over the
 *      'graph abstract data type', and many others I added. The main difference between my
 *      implementation of the basic primitive operations and Esakov's one is that I do not assume
 *      a "static" vertex set. That is: you may add or remove vertices from the graph as the
 *      computations go on.
 *      To do this, the set of vertices has been stored in a dynamic list rather than in a
 *      statically-dimensioned vector as in Esakov's book. Another difference is that I allow
 *      vertices to have the label you want (in Esakov's implementation they must be numbered
 *      from 1 to n). Finally, both edges and vertices may store information (only edges can in
 *      Esakov's).
 *
 * Acknowledgements:
 *
 *    Special thanks to Pablo Jimenez, Manel Guerris and Albert Castellet for using LLRß 0.1 &
 *    GLRß 0.1 and act asß -testers.
 *    Pablo Jimenez suggested that the code in glr_graph.c could be used to handle digrafs.
 *    I'll try to generalise the implementation of these functions to allow this.
 *    Pablo also suggested many good improvements on the documentation that comes with LLRß 0.1
 *    & GLRß 0.1. Manel found a very subtle "potential bug" in find_fundamental_cycles().
 */

#include <stdio.h>
#include <stdlib.h>
#include "general.h"
#include "llr.h"

/*-------------------------------------------------------------------------------------------------------------------*/
status allocate_node(list* p_L, generic_ptr data)
/*
 * allocate_node() performs memory allocation for a list node. *p_L returns the
 * memory address where the node is being stored. The NEXT field is initialized
 * to NULL (because at this moment the node is not yet linked to the list). The
 * node must store "data", and hence "data" is assigned to the node's DATA field.
 * allocate_node() returns ERROR if the memory allocation failed, OK otherwise.
 */
{
    list L;

    L=(list)malloc(sizeof(node));
    if(L==NULL) return ERROR;

    *p_L=L;
    DATA(L)=data;
    NEXT(L)=NULL;

    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
void free_node(list *p_L)
/*
 * free_node() frees the memory space occupied by the node pointed to by *p_L.
 * Caution free_node() does not free the space occupied by the data that can be
 * reached through DATA(*p_L). This consideration must be taken into account by
 * any higher-level function that uses free_node(). See delete_node() and
 * destroy_list() below.
 * No error checking is performed, since it is difficult that free() fails, unless
 * you pass an invalid/wrong address, and this is difficult to detect.
 */
{
```

```
    free(*p_L);
    *p_L=NULL;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status init_list(list* p_L)
/*
 * As any other variable, a list variable must be declared and initialized. We
 * declare lists by stating:
 *                          "list listname;"
 * We initialize lists by stating:
 *                          initlist(*listaname);
 */
{
    *p_L=NULL;
    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
bool empty_list(list L)
/*
 * Returns TRUE/FALSE if list L is empty/not-empty.
 */
{
    return ((L==NULL) ? TRUE : FALSE);
}
/*-------------------------------------------------------------------------------------------------------------------*/
status insert(list* p_L, generic_ptr data)
/*
 * insert() adds a new node to the begining of the list pointed to by *p_L.
 * Stores the data "data" in this node. Since we add the node at
 * the beginning of the list, *p_L will always change. Hence we declare
 * list* p_L (instead of list L) to give back the new address of the list.
 * IMPORTANT: we must cast data to the type generic_ptr in any call to
 * insert().
 * Returns ERROR if the memory allocation failed for this node. OK otherwise.
 */
{
    list L;

    if(allocate_node(&L, data) == ERROR)
        return ERROR;

    NEXT(L)=*p_L;
    *p_L=L;

    return OK;
}

/*-------------------------------------------------------------------------------------------------------------------*/
status append(list* p_L, generic_ptr data)
/*
 * append() adds a new node to the end of the list pointed to by
 * *p_L. Stores the data "data" in this node. We declare list* p_L
 * (instead of list L) since the list might possibly be empty.
 * IMPORTANT: we must cast data to the type generic_ptr in any call to
 * append(). *
 * Returns ERROR if the memory allocation failed for this node. OK
 * otherwise.
 */
{
    list L, tmplist;

    if(allocate_node(&L, data)==ERROR)
        return ERROR;

    if(empty_list(*p_L)==TRUE)
        *p_L=L;
    else
    {
        for(tmplist=*p_L;NEXT(tmplist)!=NULL;
                        tmplist=NEXT(tmplist));
        NEXT(tmplist)=L;
    }

    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status delete_first(list* p_L, generic_ptr* p_data, void (*p_func_f)())
/*
 * delete_first() deletes the first node of the list pointed to by
 * *p_L. It returns the data stored in this node through the output
 * parameter p_data. The (old) second node of the list is passed to
 * be the first now. Since the deletion of the first node is a particular
 * case of deleting any node in a list, we implement delete_first() by
 * calling another primitive operation: delete_node().
 *
 * ON FREEING  USER DEFINED DATA:
 *     Since the list abstract data type is a polymorphic type, we don't
 *     know the structure of the data stored in the DATA field of a node.
 *     Hence, we cannot define a generic function that frees this data.
 *     It is the user who must write an specific function that frees all
 *     the memory space accessible through "DATA(list_node)". We provide
 *     a means to pass the address of this user-defined "free_function" by
```

```
 *    adding the third parameter void (*p_func_f)(). The user defined
 *    function must look like this:
 *    void free_user_data(generic_ptr* data)
 */
{
    if(empty_list(*p_L)) return ERROR;
    *p_data=DATA(*p_L);
    return delete_node(p_L, *p_L, p_func_f);
}
/*-------------------------------------------------------------------------------------------------------------------*/
status delete_node(list* p_L, list node, void (*p_func_f)())
{
    if(empty_list(*p_L)==TRUE) return ERROR;

    /* si node es el primer */
    if(*p_L==node)
    {
        *p_L=NEXT(*p_L);
    }
    else
    {
        /* node no es el primer i el busquem */
        list L;
        for(L=*p_L; L!=NULL && NEXT(L)!=node; L=NEXT(L));
        if(L==NULL)
            return ERROR;
        else
            NEXT(L)=NEXT(node);
    }
    if(p_func_f!=NULL)
        (*p_func_f)(DATA(node));
    free_node(&node);

    return OK;
}
/*-------------------------------------------------------------------------------------------------------------------*/
status find_key(list L, generic_ptr key, int (*p_cmp_f)(), list* p_keynode)
{
    list curr=NULL;

    while( (curr=list_iterator(L,curr))!=NULL)
    {
        if((*p_cmp_f)(key, DATA(curr))==0)
        {
            *p_keynode=curr;
            return OK;
        }
    }
    return ERROR;
}
/*-------------------------------------------------------------------------------------------------------------------*/
list list_iterator(list L, list lastreturn)
{
        return (lastreturn==NULL)? L : NEXT(lastreturn);
}
/*-------------------------------------------------------------------------------------------------------------------*/
void destroy_list(list* p_L, void (*p_func_f)())
{
  if (empty_list(*p_L)==FALSE) {
    destroy_list(&NEXT(*p_L), p_func_f);
    if(p_func_f!=NULL) {
      /*
        * (*p_func_f)(generic_ptr data) fa un free del que s'emmagatzema a DATA(*p_L)
        * òper édesprs cal posar DATA(*p_L) a NULL: per si les mosques.
        */
      (*p_func_f)(DATA(*p_L));
      DATA(*p_L)=NULL;
    }
    free_node(p_L);
  }
}
/*-------------------------------------------------------------------------------------------------------------------*/
int length(list L)
/*
 * Obtains the length of the list L.
 */
{
  int len=0;
  list curr=NULL;
  while( (curr=list_iterator(L,curr))!=NULL)
    len++;
  return len;
}
/*-------------------------------------------------------------------------------------------------------------------*/
list first(list L)
/*
 * Returns a pointer to the first element of L
 */
{
  return L;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

```
list last(list L)
/*
 * Returns a pointer to the last element of L
 */
{
  list curr=NULL;
  if(L==NULL) return NULL;
  for(curr=L; NEXT(curr)!=NULL; curr=NEXT(curr));
  return curr;
}
```

H.38 llrtools.h

```
#ifndef LLRTOOLS_H
#define LLRTOOLS_H
status concatenate(list* p_list1, list* p_list2);
void print_prolog_list(FILE* fd, list alist, void (*p_print_funct)());
status copy_list(list List1, list* p_List2, status (*p_cp_dat)());
#endif
```

H.39 noms_gll.h

```
#ifndef NOMS_GLL_H
#define NOMS_GLL_H


#define ART_ESQ_1                   "EJOIN_1"
#define ART_ESQ_2                   "EJOIN_2"
#define ART_ESQ_3                   "EJOIN_3"
#define ART_ESQ_4                   "EJOIN_4"
#define ART_ESQ_5                   "EJOIN_5"
#define ART_ESQ_6                   "EJOIN_6"

#define ART_DRET_1                  "DJOIN_1"
#define ART_DRET_2                  "DJOIN_2"
#define ART_DRET_3                  "DJOIN_3"
#define ART_DRET_4                  "DJOIN_4"
#define ART_DRET_5                  "DJOIN_5"
#define ART_DRET_6                  "DJOIN_6"

#define TRANSL_OBJECTE              "p1Trans_objecte"
#define ROT_X_OBJECTE               "p1Rot_x"
#define ROT_Y_OBJECTE               "p1Rot_y"
#define ROT_Z_OBJECTE               "p1Rot_z"

#define TRANSL_ROB_ESQ              "ERobot"
#define ROT_ROB_ESQ                 "ETransformacio_1"
//#define ROT_Y_ROB_ESQ               "ERobot_rot_y"
//#define ROT_Z_ROB_ESQ               "ERobot_rot_z"
#define TRANSL_BASE0_ROB_ESQ        "Et_r_0"

#define TRANSL_ROB_DRET             "DRobot"
#define ROT_ROB_DRET               "DTransformacio_1"
//#define ROT_Y_ROB_DRET              "DRobot_rot_y"
//#define ROT_Z_ROB_DRET              "DRobot_rot_z"
#define TRANSL_BASE0_ROB_DRET       "Dt_r_0"

#define TRANSL_TRIEDRE_ESQ_OBJCT    "p1Trans_pine"
#define ROT_X_TRIEDRE_ESQ_OBJCT     "p1Rot_xpine"
#define ROT_Y_TRIEDRE_ESQ_OBJCT     "p1Rot_ypine"
#define ROT_Z_TRIEDRE_ESQ_OBJCT     "p1Rot_zpine"

#define TRANSL_TRIEDRE_DRET_OBJCT  "p1Trans_pind"
#define ROT_X_TRIEDRE_DRET_OBJCT   "p1Rot_xpind"
#define ROT_Y_TRIEDRE_DRET_OBJCT   "p1Rot_ypind"
#define ROT_Z_TRIEDRE_DRET_OBJCT   "p1Rot_zpind"

#endif
```

H.40 traject.h

```
#ifndef TRAJECT_H
#define TRAJECT_H

#include "configuracio.h"
#include "graf.h"

typedef struct traject_aux
{
  int num_node;
  list pnode;
  CONFIG* config_ant;
  CONFIG* config;
  CONFIG* config_seg;
  real reduc_distancia;
  bool planif_local_cridat;
}NODE_TRAJECT_AUX;

typedef struct struct_taula_traject_aux{
```

```
  int n;
  NODE_TRAJECT_AUX* t;
}TAULA_TRAJECT_AUX;

TAULA_TRAJECT_AUX* copiar_traject_en_taula_traject_auxiliar(list* p_traject);
//void alliberar_taula_traject_auxiliar(list** p_traject);
void calcular_distancies_taula_traject_auxiliar(DADES_CONFIG* dades_config,
                                                TAULA_TRAJECT_AUX* taula_traject_aux);
real calcular_distancia_de_cel_la_i(DADES_CONFIG* dades_config,TAULA_TRAJECT_AUX* taula_traject_aux,int i);
int trobar_posicio_taula_donat_pnode(TAULA_TRAJECT_AUX* taula_traject_aux,list pnode);
void ordenar_taula_traject_auxiliar_segons_reduc_distancia(TAULA_TRAJECT_AUX* taula_traject_aux);
void ordenar_taula_traject_auxiliar_segons_num_node(TAULA_TRAJECT_AUX* taula_traject_aux);
int compare_nodes_traject_aux_reduc_distancia (const void * a, const void * b);
int compare_nodes_traject_aux_num_node(const void * a, const void * b);
void desplacament_taula_traject_auxiliar(TAULA_TRAJECT_AUX* taula_traject_aux,int i);
void escriure_taula_traject_auxiliar(TAULA_TRAJECT_AUX* taula_traject_aux);
real calcular_distancia_total_traject(DADES_CONFIG* dades_config,list traject);
#endif
```

H.41 traject.c

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// graf_config.c
//
// Autor:   Gorka Bonals
// Creat:   Juliol 2005
// Revisat: Agost 2005
// óVersi: 1.0
//
// traject.cé s el òmdul que écont les rutines que treballen
// sobre una taula auxiliar, que écont la óinformaci d'una
// òtrajectria. La utilitzem per que íaix podem utilitzar la ófunci
// qsort() que es moltú til per ordenar una taula segons un
// camp desitjat.
//
// rutines úpbliques:
//
// -copiar_traject_en_taula_traject_auxiliar()
//    Rutina que copia una llista de nodes que écont la óinformaci d'una
// ò   trajectria, en una taula auxiliar.
// -calcular_distancies_taula_traject_auxiliar()
//    Rutina que calcula les àdistncies entre nodes adjacents d'una taula
//    auxiliar.
// -ordenar_taula_traject_auxiliar_segons_reduc_distancia()
//    Rutina que ordena la taula auxiliar segons el camp àdistncia creixent.
// -compare_nodes_traject_aux_reduc_distancia()
//    Rutina que estableix el criteri d'óordenaci del camp àdistncia que
// à   utilitzar la rutina qsort().
// -ordenar_taula_traject_auxiliar_segons_num_node()
//    Rutina que ordena la taula auxiliar segons el camp num node de forma
//    creixent.
// -compare_nodes_traject_aux_num_node()
//    Rutina que estableix el criteri d'óordenaci creixent, del camp
// ú   nmero de node que àutilitzar la ófunci qsort().
// -desplacament_taula_traject_auxiliar()
//    Rutina que realitza el çdesplaament de la taula auxiliar en una
// ó   posici enrera. Ho fa a partir de la ·cella que écont com a camp de
//    pvertex un donat per la ófunci. De feté s com si ísuprimssim el
//    node de la taula auxiliar.
// -escriure_taula_traject_auxiliar()
//    Rutina que escriu per pantalla la óinformaci de la taula auxiliar.
// -calcular_distancia_total_traject()
//    Rutina que calcula la àdistncia total d'una òtrajectria.
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

#include "traject.h"

////////////////////////////////////////////////////////////////
// copiar_traject_en_taula_traject_auxiliar()
//
// Aquesta rutina copia una òtrajectria queé s una llista
// çenllaada en una taula auxiliar. Cada node de la taula auxiliaré
// s del tipus NODE_TRAJECT_AUX, que àest descrit en traject.h
//
// Arguments:
//   p_traject:
// É     s un punter a la llista que écont la òtrajectria.
// Retorna:
//   Un punter a l'estructura que àcontindr la
//   taula auxiliar i la longitud de la mateixa.
//
////////////////////////////////////////////////////////////////

TAULA_TRAJECT_AUX* copiar_traject_en_taula_traject_auxiliar(list* p_traject)
{
  int i,n;
  TAULA_TRAJECT_AUX* taula_traject_aux;
  list pnode_ant;
  list pnode_act=NULL;
```

```
  list pnode_seg;

  //Calculem la longitud de la òtrajectria(en nombre de nodes).
  n=length(*p_traject);

  //Calculem un punter a l'estructura que àcontindr la taula auxiliar
  //i la longitud de la mateixa.
  if(NEW(taula_traject_aux,1,TAULA_TRAJECT_AUX)==NULL)
    cerror("error de malloc copiar traject en taula traject auxiliar()\n");
  if(NEW(taula_traject_aux->t,n,NODE_TRAJECT_AUX)==NULL)
    cerror("error de malloc copiar traject en taula traject auxiliar()\n");

  taula_traject_aux->n=n;

  //Suposem que els camins d'una òtrajectria tenen com a minim tres nodes
  //Per omplir la taula auxiliar a partir de la òtrajectria haurem de fer
  //tres recorreguts àsimultnis per anar calculant les àdistncies entre
  //nodes que formen un triangle.
  pnode_ant=*p_traject;
  pnode_act=NEXT(pnode_ant);
  pnode_seg=NEXT(pnode_act);

  //Tractament del primer element
  (taula_traject_aux->t)[0].num_node=0;
  (taula_traject_aux->t)[0].pnode=pnode_ant;
  //El primer element no écont óconfiguraci anterior.
  (taula_traject_aux->t)[0].config_ant=NULL;
  (taula_traject_aux->t)[0].config=(CONFIG*)DATA(pnode_ant);
  (taula_traject_aux->t)[0].config_seg=NULL;
  //Li assignem un valor molt reduit, perque aquest node al ser l'inicial mai es àpodr borrar.
  //íAix quan ordenem la taula segons els nodes que redueixin éms distancia(valor positiu),
  //el node inicial sempre àestar al final.
  (taula_traject_aux->t)[0].reduc_distancia=-20000;
  (taula_traject_aux->t)[0].planif_local_cridat=FALSE;

  i=1;

    while (pnode_seg!=NULL)
    {
      (taula_traject_aux->t)[i].num_node=i;
      (taula_traject_aux->t)[i].pnode=pnode_act;
      (taula_traject_aux->t)[i].config_ant=(CONFIG*)DATA(pnode_ant);
      (taula_traject_aux->t)[i].config=(CONFIG*)DATA(pnode_act);
      (taula_traject_aux->t)[i].config_seg=(CONFIG*)DATA(pnode_seg);
      (taula_traject_aux->t)[i].reduc_distancia=-10000;
      (taula_traject_aux->t)[i].planif_local_cridat=FALSE;

      //óActualitzaci de les variables que anem recorrent la òtrajectria.
      pnode_ant=pnode_act;
      pnode_act=pnode_seg;
      pnode_seg=NEXT(pnode_act);
      i=i+1;
    }
    //tractament de l ultim element, ja que no s'haura
    //tractat en el bucle anterior.

    (taula_traject_aux->t)[i].num_node=i;
    (taula_traject_aux->t)[i].pnode=pnode_act;
    (taula_traject_aux->t)[i].config_ant=NULL;
    (taula_traject_aux->t)[i].config=(CONFIG*)DATA(pnode_act);
    (taula_traject_aux->t)[i].config_seg=NULL;
    //éTamb com el node inicial, li assignem un valor molt gran
    //perque el node final no el podem eliminar.
    (taula_traject_aux->t)[i].reduc_distancia=-20000;
    (taula_traject_aux->t)[i].planif_local_cridat=FALSE;

  return taula_traject_aux;
}

void calcular_distancies_taula_traject_auxiliar(DADES_CONFIG* dades_config, TAULA_TRAJECT_AUX* taula_traject_aux)
{
  int i,n;
  real x,y,z;


  //Comenco per l'index 1, perque el node 0, no te dos nodes
  //per calcular la distancia.
  i=1;

  //El nombre d'elements de la taula auxiliar.
  n=taula_traject_aux->n;

  //L'ultim element te index n-1 perque estem recorrent una taula
  //de n elements.
  for(i=1;i<(n-1);i++)
    {
      x=distancia_fina_entre_configuracions(dades_config,
                                            (taula_traject_aux->t)[i].config_ant,
                                            (taula_traject_aux->t)[i].config);
      y=distancia_fina_entre_configuracions(dades_config,
                                            (taula_traject_aux->t)[i].config,
                                            (taula_traject_aux->t)[i].config_seg);
```

```
      z=distancia_fina_entre_configuracions(dades_config,
                                            (taula_traject_aux->t)[i].config_ant,
                                            (taula_traject_aux->t)[i].config_seg);

      if ((x!=BIG)&&(y!=BIG)&&(z!=BIG))
        (taula_traject_aux->t)[i].reduc_distancia=(x+y)-z;
      else
        (taula_traject_aux->t)[i].reduc_distancia=-10000;
    }
}

int trobar_posicio_taula_donat_pnode(TAULA_TRAJECT_AUX* taula_traject_aux,list pnode_borrar)
{
  int i,n;

  n=taula_traject_aux->n;

  i=0;

  //Anem a buscar l'index de la taula del node que volem borrar.
  while (i<(n-1))
    {
    if ((((taula_traject_aux->t)[i]).pnode)==pnode_borrar)
      break;
    i=i+1;
    }
  return i;
}

real calcular_distancia_de_cel_la_i(DADES_CONFIG* dades_config,TAULA_TRAJECT_AUX* taula_traject_aux,int i)
{
  real x,y,z;
  real distancia;

  x=distancia_fina_entre_configuracions(dades_config,
                                        (taula_traject_aux->t)[i].config_ant,
                                        (taula_traject_aux->t)[i].config);
  y=distancia_fina_entre_configuracions(dades_config,
                                        (taula_traject_aux->t)[i].config,
                                        (taula_traject_aux->t)[i].config_seg);
  z=distancia_fina_entre_configuracions(dades_config,
                                        (taula_traject_aux->t)[i].config_ant,
                                        (taula_traject_aux->t)[i].config_seg);

  if ((x!=10000)&&(y!=10000)&&(z!=10000))
    distancia=(x+y)-z;
  else
    distancia=-10000;
  return(distancia);
}
void ordenar_taula_traject_auxiliar_segons_reduc_distancia(TAULA_TRAJECT_AUX* taula_traject_aux)
{
  qsort(taula_traject_aux->t,taula_traject_aux->n,
        sizeof(NODE_TRAJECT_AUX),
        compare_nodes_traject_aux_reduc_distancia);
}
int compare_nodes_traject_aux_reduc_distancia (const void * a, const void * b)
{
  if (((const NODE_TRAJECT_AUX*)a)->reduc_distancia>((const NODE_TRAJECT_AUX*)b)->reduc_distancia)
    return(-1);
  else if (((const NODE_TRAJECT_AUX*)a)->reduc_distancia<((const NODE_TRAJECT_AUX*)b)->reduc_distancia)
    return(1);
  else
    return(0);
}
void ordenar_taula_traject_auxiliar_segons_num_node(TAULA_TRAJECT_AUX* taula_traject_aux)
{
  qsort(taula_traject_aux->t,taula_traject_aux->n,
        sizeof(NODE_TRAJECT_AUX),
        compare_nodes_traject_aux_num_node);
}
int compare_nodes_traject_aux_num_node (const void * a, const void * b)
{
  if (((const NODE_TRAJECT_AUX*)a)->num_node<((const NODE_TRAJECT_AUX*)b)->num_node)
    return(-1);
  else if (((const NODE_TRAJECT_AUX*)a)->num_node>((const NODE_TRAJECT_AUX*)b)->num_node)
    return(1);
  else
    return(0);
}

void desplacament_taula_traject_auxiliar(TAULA_TRAJECT_AUX* taula_traject_aux,int i)
{
 int n;

  n=taula_traject_aux->n;


  //pel node que es desplaca primer, se li fa un tractament especial
  //perque els seu vei anterior canvia
  if (i!=0)
    {
```

```
      (taula_traject_aux->t[i-1]).config_seg= (taula_traject_aux->t[i+1]).config;
    }
  (taula_traject_aux->t[i]).num_node =(taula_traject_aux->t[i+1]).num_node;
  (taula_traject_aux->t[i]).pnode =(taula_traject_aux->t[i+1]).pnode;
  (taula_traject_aux->t[i]).config= (taula_traject_aux->t[i+1]).config;
  (taula_traject_aux->t[i]).config_seg= (taula_traject_aux->t[i+1]).config_seg;
  (taula_traject_aux->t[i]).reduc_distancia =(taula_traject_aux->t[i+1]).reduc_distancia;


  i=i+1;

  //fem un recorregut de tots els els elements de la taula a partir del node que
  //s'ha de borrar
  while (i<(n-1))
    {
      (taula_traject_aux->t[i]).num_node =(taula_traject_aux->t[i+1]).num_node;
      (taula_traject_aux->t[i]).pnode =(taula_traject_aux->t[i+1]).pnode;
      (taula_traject_aux->t[i]).config_ant= (taula_traject_aux->t[i+1]).config_ant;
      (taula_traject_aux->t[i]).config= (taula_traject_aux->t[i+1]).config;
      (taula_traject_aux->t[i]).config_seg= (taula_traject_aux->t[i+1]).config_seg;
      (taula_traject_aux->t[i]).reduc_distancia =(taula_traject_aux->t[i+1]).reduc_distancia;
      i=i+1;
    }

  //Tractament final pel node que queda degut al çdesplaament dels altres.
  (taula_traject_aux->t[i]).num_node=50000;
  (taula_traject_aux->t[i]).pnode =NULL;
  (taula_traject_aux->t[i]).config_ant= NULL;
  (taula_traject_aux->t[i]).config= NULL;
  (taula_traject_aux->t[i]).config_seg= NULL;
  //Posem distancia negativa, perque aixi significa que quan ordenem
  //anira al final.
  (taula_traject_aux->t[i]).reduc_distancia =-10000;

  //es una manera de dir que la taula disminueix de longitud,
  //ja que borrarem l element que hem desplacat.
  taula_traject_aux->n=n-1;
}

void escriure_taula_traject_auxiliar(TAULA_TRAJECT_AUX* taula_traject_aux)
{
  int i;

  for (i=0;i<(taula_traject_aux->n);i++)
    {
      printf("taula_traject_aux->t[%d].num_node=%d\n",i,(taula_traject_aux->t[i]).num_node);
      printf("taula_traject_aux->t[%d].pnode=%p\n",i,(taula_traject_aux->t[i]).pnode);
      printf("taula_traject_aux->t[%d].config_ant=%p\n",i,(taula_traject_aux->t[i]).config_ant);
      printf("taula_traject_aux->t[%d].config=%p\n",i,(taula_traject_aux->t[i]).config);
      printf("taula_traject_aux->t[%d].config_seg=%p\n",i,(taula_traject_aux->t[i]).config_seg);
      printf("taula_traject_aux->t[%d].reduc_distancia=%f\n",i,(taula_traject_aux->t[i]).reduc_distancia);
    }
}
real calcular_distancia_total_traject(DADES_CONFIG* dades_config,list traject)
{

  list pnode_ant;
  list pnode_act;
  CONFIG* config_ant;
  CONFIG* config_act;
  real distancia_total=0.0;

  if (empty_list(traject))
    {
      printf("no_calculem_distancia_perque_no_hi_ha_trajectoria\n");
      distancia_total=-1;
    }
  else
    {
      pnode_ant=traject;
      pnode_act=NEXT(pnode_ant);

      while (pnode_act!=NULL)
        {
          config_ant=(CONFIG*)DATA(pnode_ant);
          config_act=(CONFIG*)DATA(pnode_act);

          distancia_total=distancia_total+distancia_fina_entre_configuracions(dades_config,
                                                                              config_ant,config_act);

          pnode_ant=pnode_act;
          pnode_act=NEXT(pnode_ant);
        }
    }
  return distancia_total;
}
```

H.42 transf_ctnt.h

```
#ifndef TRANSF_CTNT_H
#define TRANSF_CTNT_H
```

```
//////////////////////////////////////////////////////
/// @file transf_ctnt.h
/// @brief Calcula les transf. homog constants del problema.
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///      Agrupa les transformacions constants del
///      problema en la tupla  TRANSF_CTNT.
//////////////////////////////////////////////////////

#include "transf_homog_ctnt.h"

typedef struct trnsf_ctnt{
  TRANSF_HOMOG ba_T_b0e;
  TRANSF_HOMOG ba_T_b0d;
  TRANSF_HOMOG b0e_T_ba;
  TRANSF_HOMOG b0d_T_ba;
  TRANSF_HOMOG graspe;
  TRANSF_HOMOG graspd;
  TRANSF_HOMOG graspe_inv;
  TRANSF_HOMOG graspd_inv;
}TRANSF_CTNT;

void obtenir_transf_ctnt(TRANSF_CTNT* T);

#endif
```

H.43 transf_ctnt.c

```
#include "transf_ctnt.h"

//TRANSF_CTNT T;

void obtenir_transf_ctnt(TRANSF_CTNT* T)
{
  transf_homogenia_ba_T_b0e(T->ba_T_b0e);
  transf_homogenia_ba_T_b0d(T->ba_T_b0d);
  inversa_transf_homogenia(T->ba_T_b0e,T->b0e_T_ba);
  inversa_transf_homogenia(T->ba_T_b0d,T->b0d_T_ba);
  transf_homogenia_Tgraspe (T->graspe);
  transf_homogenia_Tgraspd (T->graspd);
  inversa_transf_homogenia(T->graspe,T->graspe_inv);
  inversa_transf_homogenia(T->graspd,T->graspd_inv);
}
```

H.44 transf_homog_ctnt.h

```
#ifndef TRANSF_HOMOG_CTNT_H
#define TRANSF_HOMOG_CTNT_H

//////////////////////////////////////////////////////
/// @file transf_homog_ctnt.h
/// @brief Calcula les transf. homog constants del problema.
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///      Calcula les matrius de transformacio èhomognies
///      que ósn constants: les de óposici i óorientaci
///      dels robots i les de óprensi esquerra i dreta
///      de l'objecte.
//////////////////////////////////////////////////////

#include <stdlib.h>
#include "webots_interface.h"
#include "transf_homogenies.h"

void transf_homogenia_ba_T_b0e(TRANSF_HOMOG ba_T_b0e);
void transf_homogenia_ba_T_b0d(TRANSF_HOMOG ba_T_b0d);
void transf_homogenia_Tgraspe (TRANSF_HOMOG Tgraspe);
void transf_homogenia_Tgraspd (TRANSF_HOMOG Tgraspd);

#endif
```

H.45 transf_homog_ctnt.c

```
#include "transf_homog_ctnt.h"

void transf_homogenia_ba_T_b0e(TRANSF_HOMOG ba_T_b0e)
{
  TRANSF_HOMOG Mt1,Mre,Mrx,Mt2;
  TRANSF_HOMOG P1,P2;
  real translacio_rob_esq[3];
  int i;

  //Obtencio dels valors de translacio i rotacio
  //del robot respecte la base absoluta
  obtenir_angle_solid(TRANSL_ROB_ESQ,&(rot_x_rob_esq));
  obtenir_translacio_rotacio_solid(ROT_ROB_ESQ, transl_rot_rob_esq);
```

```
  obtenir_translacio_solid(TRANSL_BASE0_ROB_ESQ,transl_base0_rob_esq);
  avancar_simulacio();
  printf("GDL_llegits:\n");
  printf("\tttranslacio_rob_esq=(%f,%f,%f)\n",
         transl_rot_rob_esq[0],transl_rot_rob_esq[1],
         transl_rot_rob_esq[2]);
  printf("\ttransl_base0_rob_esq=(%f,%f,%f)\n",
         transl_base0_rob_esq[0],transl_base0_rob_esq[1],
         transl_base0_rob_esq[2]);
  printf("\trotacio_solid=(%f,%f,%f,%f)\n",
         transl_rot_rob_esq[3],transl_rot_rob_esq[4],
         transl_rot_rob_esq[5], transl_rot_rob_esq[6]);
  printf("\trotacioe=(%f)\n",rot_x_rob_esq);

  for(i=0;i<=2;i++)
    translacio_rob_esq[i]= transl_rot_rob_esq[i];

  //Calcul de la matriu de transformacio a partir
  //dels valors obtinguts
  transf_homogenia_rot_x(Mrx,rot_x_rob_esq);
  transf_homogenia_translacio(Mt1,translacio_rob_esq);
  //Transformacio en x per poder visualitzar éb els robots amb el mouse
  transf_homogenia_rot_eix_generic(Mre,
                                   transl_rot_rob_esq[3],transl_rot_rob_esq[4],
                                   transl_rot_rob_esq[5],transl_rot_rob_esq[6]);
  transf_homogenia_translacio(Mt2,transl_base0_rob_esq);

  producte_transf_homogenies(Mrx,Mt1,P1);
  producte_transf_homogenies(P1,Mre,P2);
  producte_transf_homogenies(P2,Mt2,ba_T_b0e);
}
void transf_homogenia_ba_T_b0d(TRANSF_HOMOG ba_T_b0d)
{
  TRANSF_HOMOG Mt1,Mre,Mrx,Mt2;
  TRANSF_HOMOG P1,P2;
  real translacio_rob_dret[3];
  int i;
  //Obtencio dels valors de translacio i rotacio
  //del robot respecte la base absoluta

  obtenir_angle_solid(TRANSL_ROB_DRET,&(rot_x_rob_dret));
  obtenir_translacio_rotacio_solid(ROT_ROB_DRET, transl_rot_rob_dret);
  obtenir_translacio_solid(TRANSL_BASE0_ROB_DRET,transl_base0_rob_dret);
  avancar_simulacio();

  printf("GDL_llegits:\n");
  printf("\tttranslacio_rob_dret=(%f,%f,%f)\n",
         transl_rot_rob_dret[0],transl_rot_rob_dret[1],
         transl_rot_rob_dret[2]);
  printf("\ttransl_base0_rob_dret=(%f,%f,%f)\n",
         transl_base0_rob_dret[0],transl_base0_rob_dret[1],
         transl_base0_rob_dret[2]);
  printf("\trotacio_solid=(%f,%f,%f,%f)\n",
         transl_rot_rob_dret[3],transl_rot_rob_dret[4],
         transl_rot_rob_dret[5],transl_rot_rob_dret[6]);
  printf("\trotaciod=(%f)\n",rot_x_rob_dret);
  for(i=0;i<=2;i++)
    translacio_rob_dret[i]= transl_rot_rob_dret[i];

  //Calcul de la matriu de transformacio
  //homogenia
  transf_homogenia_rot_x(Mrx,rot_x_rob_dret);
  transf_homogenia_translacio(Mt1,translacio_rob_dret);
  transf_homogenia_rot_eix_generic(Mre,
                                   transl_rot_rob_dret[3],
                                   transl_rot_rob_dret[4],
                                   transl_rot_rob_dret[5],
                                   transl_rot_rob_dret[6]);
  //Transf_homogenia_rot_z(Mre,rotacio_rob_dret[3]);
  transf_homogenia_translacio(Mt2,transl_base0_rob_dret);

  producte_transf_homogenies(Mrx,Mt1,P1);
  producte_transf_homogenies(P1,Mre,P2);
  producte_transf_homogenies(P2,Mt2,ba_T_b0d);
}
void transf_homogenia_Tgraspe(TRANSF_HOMOG  Tgraspe)
{
  TRANSF_HOMOG Mt,Mrxyz;

  //Obtencio dels valors de translacio i rotacio
  //del robot respecte la base absoluta
  obtenir_translacio_solid(TRANSL_TRIEDRE_ESQ_OBJCT,transl_grasp_esq);
  obtenir_angle_solid (ROT_X_TRIEDRE_ESQ_OBJCT,&(rot_grasp_esq[0]));
  obtenir_angle_solid (ROT_Y_TRIEDRE_ESQ_OBJCT,&(rot_grasp_esq[1]));
  obtenir_angle_solid (ROT_Z_TRIEDRE_ESQ_OBJCT,&(rot_grasp_esq[2]));
  avancar_simulacio();
  printf("GDL_llegits:\n");
  printf("\ttranslacio_grasp_esquerra=(%f,%f,%f)\n",
         transl_grasp_esq[0],transl_grasp_esq[1],transl_grasp_esq[2]);
  printf("\trotacio_grasp_esquerra=(%f,%f,%f)\n",
         rot_grasp_esq[0],rot_grasp_esq[1],rot_grasp_esq[2]);
   //Calcul de la matriu de transformacio
```

```
  //homogenia
  transf_homogenia_translacio(Mt,transl_grasp_esq);
  transf_homogenia_rot_xyz(Mrxyz,
                           rot_grasp_esq[0],rot_grasp_esq[1],
                           rot_grasp_esq[2]);
  producte_transf_homogenies(Mt,Mrxyz,Tgraspe);
}
void transf_homogenia_Tgraspd(TRANSF_HOMOG Tgraspd)
{
  TRANSF_HOMOG Mt,Mrxyz;

  //obtencio dels valors de translacio i rotacio
  //del robot respecte la base absoluta
  obtenir_translacio_solid(TRANSL_TRIEDRE_DRET_OBJCT,transl_grasp_dret);
  obtenir_angle_solid (ROT_X_TRIEDRE_DRET_OBJCT,&(rot_grasp_dret[0]));
  obtenir_angle_solid (ROT_Y_TRIEDRE_DRET_OBJCT,&(rot_grasp_dret[1]));
  obtenir_angle_solid (ROT_Z_TRIEDRE_DRET_OBJCT,&(rot_grasp_dret[2]));
  avancar_simulacio();
  printf("GDL_llegits:\n");
  printf("\ttranslacio_grasp_dret=(%f,%f,%f)\n",
         transl_grasp_dret[0],transl_grasp_dret[1],transl_grasp_dret[2]);
  printf("\trotacio_grasp_dret=(%f,%f,%f)\n",
         rot_grasp_dret[0],rot_grasp_dret[1],rot_grasp_dret[2]);
  //Calcul de la matriu de transformacio
  //homogenia
  transf_homogenia_translacio(Mt,transl_grasp_dret);
  transf_homogenia_rot_xyz(Mrxyz,rot_grasp_dret[0],rot_grasp_dret[1],
                           rot_grasp_dret[2]);
  producte_transf_homogenies(Mt,Mrxyz,Tgraspd);
}
```

H.46 transf_homogenies.h

```
#ifndef TRANSF_HOMOGENIES_H
#define TRANSF_HOMOGENIES_H

////////////////////////////////////////////////////
/// @file transf_homogenies.h
/// @brief éCont les rutines que treballen amb transf. homog.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///      Implementa les rutines relacionades
///      amb transformacions èhomognies.
////////////////////////////////////////////////////



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Gtypes.h"

typedef real TRANSF_HOMOG[4][4];

//Operacions constructores
void inicialitzar_matriu(TRANSF_HOMOG M);
void transf_homogenia_rot_x(TRANSF_HOMOG M, real x);
void transf_homogenia_rot_y(TRANSF_HOMOG M,real y);
void transf_homogenia_rot_z(TRANSF_HOMOG M,real z);
void transf_homogenia_rot_xyz(TRANSF_HOMOG M, real x, real y, real z);
void transf_homogenia_rot_eix_generic(TRANSF_HOMOG M,real kx, real ky, real kz, real teta);
void transf_homogenia_translacio(TRANSF_HOMOG M,real* translacio);
void producte_transf_homogenies(TRANSF_HOMOG M1,TRANSF_HOMOG M2,TRANSF_HOMOG P);
void transf_homogenia_per_vector(TRANSF_HOMOG M,real* vector);
void inversa_transf_homogenia(TRANSF_HOMOG M,TRANSF_HOMOG I);
void convertir_TRSF_a_TRANSF_HOMOG(TRSF* tr,TRANSF_HOMOG Mr);
void carregar_transf_homogenia(FILE* fp,TRANSF_HOMOG M);

//Operacions consultores
void imprimir_transf_homogenia(TRANSF_HOMOG M);
void convertir_TRANSF_HOMOG_a_TRSF(TRANSF_HOMOG Mr,TRSF* tr);
void desar_transf_homogenia(TRANSF_HOMOG M,FILE* fp);

#endif
```

H.47 transf_homogenies.c

```
/*
 transf_homogenies.h
 */
#include "transf_homogenies.h"

void inicialitzar_matriu(TRANSF_HOMOG M)
{
  int i,j;
  for (i=0;i<=3;i++)
    for(j=0;j<=3;j++)
      M[i][j]=0.0;
```

```
}

void transf_homogenia_rot_x(TRANSF_HOMOG M, real x)
{
  M[0][0]=1;
  M[0][1]=0;
  M[0][2]=0;
  M[0][3]=0;
  M[1][0]=0;
  M[1][1]=cos(x);
  M[1][2]=-sin(x);
  M[1][3]=0;
  M[2][0]=0;
  M[2][1]=sin(x);
  M[2][2]=cos(x);
  M[2][3]=0;
  M[3][0]=0;
  M[3][1]=0;
  M[3][2]=0;
  M[3][3]=1;
}

void transf_homogenia_rot_y(TRANSF_HOMOG M,real y)
{
  M[0][0]=cos(y);
  M[0][1]=0;
  M[0][2]=sin(y);
  M[0][3]=0;
  M[1][0]=0;
  M[1][1]=1;
  M[1][2]=0;
  M[1][3]=0;
  M[2][0]=-sin(y);
  M[2][1]=0;
  M[2][2]=cos(y);
  M[2][3]=0;
  M[3][0]=0;
  M[3][1]=0;
  M[3][2]=0;
  M[3][3]=1;
}

void transf_homogenia_rot_z(TRANSF_HOMOG M,real z)
{
  M[0][0]=cos(z);
  M[0][1]=-sin(z);
  M[0][2]=0;
  M[0][3]=0;
  M[1][0]=sin(z);
  M[1][1]=cos(z);
  M[1][2]=0;
  M[1][3]=0;
  M[2][0]=0;
  M[2][1]=0;
  M[2][2]=1;
  M[2][3]=0;
  M[3][0]=0;
  M[3][1]=0;
  M[3][2]=0;
  M[3][3]=1;

}

void transf_homogenia_rot_xyz(TRANSF_HOMOG M, real x, real y, real z)
{
  M[0][0]=cos(y)*cos(z);
  M[0][1]=-cos(y)*sin(z);
  M[0][2]=sin(y);
  M[0][3]=0;
  M[1][0]=sin(x)*sin(y)*cos(z)+cos(x)*sin(z);
  M[1][1]=-sin(x)*sin(y)*sin(z)+cos(x)*cos(z);
  M[1][2]=-sin(x)*cos(y);
  M[1][3]=0;
  M[2][0]=-cos(x)*sin(y)*cos(z)+sin(x)*sin(z);
  M[2][1]=cos(x)*sin(y)*sin(z)+sin(x)*cos(z);
  M[2][2]=cos(x)*cos(y);
  M[2][3]=0;
  M[3][0]=0;
  M[3][1]=0;
  M[3][2]=0;
  M[3][3]=1;
}
void transf_homogenia_rot_eix_generic(TRANSF_HOMOG M,real kx, real ky, real kz, real teta)
{
   real v_teta;

   v_teta=1-cos(teta);

  M[0][0]=pow(kx,2.0)*v_teta+cos(teta);
  M[0][1]=kx*ky*v_teta-kz*sin(teta);
  M[0][2]=kx*kz*v_teta+ky*sin(teta);
  M[0][3]=0;
```

```
  M[1][0]=kx*ky*v_teta+kz*sin(teta);
  M[1][1]=pow(ky,2.0)*v_teta+cos(teta);
  M[1][2]=ky*kz*v_teta-kx*sin(teta);
  M[1][3]=0;
  M[2][0]=kx*kz*v_teta-ky*sin(teta);
  M[2][1]=ky*kz*v_teta+kx*sin(teta);
  M[2][2]=pow(kz,2.0)*v_teta+cos(teta);
  M[2][3]=0;
  M[3][0]=0;
  M[3][1]=0;
  M[3][2]=0;
  M[3][3]=1;
}

void transf_homogenia_translacio(TRANSF_HOMOG M,real* translacio)
{
  int i,j;

  for (i=0;i<=3;i++)
    {
      for(j=0;j<=3;j++)
        {
          if (i!=j)
            M[i][j]=0.0;
          else
            M[i][j]=1.0;
        }
    }

  M[0][3]=translacio[0];
  M[1][3]=translacio[1];
  M[2][3]=translacio[2];

}

void producte_transf_homogenies(TRANSF_HOMOG M1,TRANSF_HOMOG M2,TRANSF_HOMOG P)
{
  int i,j,k;
  real sum;

  for (k=0;k<=3;k++)
    {
      for(j=0;j<=3;j++)
        {
          sum=0.0;
          for (i=0;i<=3;i++)
            {
              sum=sum+M1[k][i]*M2[i][j];
            }
          P[k][j]=sum;
        }
    }
}
// el vector ha de tenir dimensio 4
void transf_homogenia_per_vector(TRANSF_HOMOG M,real* vector)
{
  int j,k;
  real vector_resultant[4];

  //Creem un vector auxiliar que contindra
  //el resultat.
  for(j=0;j<=2;j++)
    vector_resultant[j]=0.0;
  vector_resultant[3]=1.0;

  for(j=0;j<=3;j++)
    for(k=0;k<=3;k++)
      vector_resultant[j]= M[j][k]*vector[k]+vector_resultant[j];
  //passem al vector d'entrada el resultat
  //del vector resultant
  for(j=0;j<=3;j++)
    vector[j]=vector_resultant[j];
}

void imprimir_transf_homogenia(TRANSF_HOMOG M)
{
  int i;
  for (i=0;i<=3;i++)
      printf("%f_%f_%f_%f\n",M[i][0],M[i][1],M[i][2],M[i][3]);
}

void inversa_transf_homogenia(TRANSF_HOMOG M,TRANSF_HOMOG I)
{
  I[0][0]=M[0][0];
  I[1][0]=M[0][1];
  I[2][0]=M[0][2];
  I[3][0]=M[0][3];

  I[0][1]=M[1][0];
  I[1][1]=M[1][1];
  I[2][1]=M[1][2];
  I[3][1]=M[1][3];
```

```
  I[0][2]=M[2][0];
  I[1][2]=M[2][1];
  I[2][2]=M[2][2];
  I[3][2]=M[2][3];

  I[0][3]=-(M[0][3]*M[0][0]+M[1][3]*M[1][0]+ M[2][3]*M[2][0]);
  I[1][3]=-(M[0][3]*M[0][1]+M[1][3]*M[1][1]+ M[2][3]*M[2][1]);
  I[2][3]=-(M[0][3]*M[0][2]+M[1][3]*M[1][2]+ M[2][3]*M[2][2]);

  I[3][0]=0;
  I[3][1]=0;
  I[3][2]=0;
  I[3][3]=1;
}
void convertir_TRSF_a_TRANSF_HOMOG(TRSF* tr,TRANSF_HOMOG Mr)
{
  Mr[0][0]=(real)tr->n.x;
  Mr[1][0]=(real)tr->n.y;
  Mr[2][0]=(real)tr->n.z;

  Mr[0][1]=(real)tr->o.x;
  Mr[1][1]=(real)tr->o.y;
  Mr[2][1]=(real)tr->o.z;

  Mr[0][2]=(real)tr->a.x;
  Mr[1][2]=(real)tr->a.y;
  Mr[2][2]=(real)tr->a.z;

  Mr[0][3]=(real)tr->p.x;
  Mr[1][3]=(real)tr->p.y;
  Mr[2][3]=(real)tr->p.z;

  Mr[3][0]=0.0;
  Mr[3][1]=0.0;
  Mr[3][2]=0.0;
  Mr[3][3]=1.0;
}
void convertir_TRANSF_HOMOG_a_TRSF(TRANSF_HOMOG Mr,TRSF* tr)
{
  tr->n.x=(Greal)Mr[0][0];
  tr->n.y=(Greal)Mr[1][0];
  tr->n.z=(Greal)Mr[2][0];

  tr->o.x=(Greal)Mr[0][1];
  tr->o.y=(Greal)Mr[1][1];
  tr->o.z=(Greal)Mr[2][1];

  tr->a.x=(Greal)Mr[0][2];
  tr->a.y=(Greal)Mr[1][2];
  tr->a.z=(Greal)Mr[2][2];

  tr->p.x=(Greal)Mr[0][3];
  tr->p.y=(Greal)Mr[1][3];
  tr->p.z=(Greal)Mr[2][3];
}
void desar_transf_homogenia(TRANSF_HOMOG M,FILE* fp)
{
  int i;
  for(i=0;i<=3;++i)
    fprintf(fp,"%f_%f_%f_%f\n",M[i][0],M[i][1],M[i][2],M[i][3]);
}

void carregar_transf_homogenia(FILE* fp,TRANSF_HOMOG M)
{
  int i;
  for(i=0;i<=3;++i)
    fscanf(fp,"%f_%f_%f_%f\n",&(M[i][0]),&(M[i][1]),&(M[i][2]),&(M[i][3]));
}
```

H.48 utils.h

```
#ifndef UTILS_H
#define UTILS_H
/*
 *  utils.h
 *  Lluis Ros
 *  Revision: May 1999
 */
#include "general.h"

#define VAL_INICIAL -10

bool is_one_of(unsigned int num, unsigned int* numbers,int nnumbers);
void cerror(char* error);
#endif
```

H.49 vector.h

```
#ifndef VECTOR_H
#define VECTOR_H
```

```
/////////////////////////////////////////////////////
/// @file vector.h
/// @brief éCont les rutines que treballen amb un vector.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///      Implementa totes les rutines relacionades amb
///      operacions àbsiques amb vectors.
/////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "macros_generals.h"
#include "general.h"

typedef struct vect{
  int n;
  real* v;
}VECTOR;

#define DIM(vector) ((vector)->n)
#define ELEM(vector) ((vector)->v)
#define ELEM_i(vector,i) ((vector)->v[i])

//operacions constructores

/////////////////////////////////////////////////////
///  @brief Crea un vector d'n components.
///
///  Reserva òmemria per una varialbe del tipus VECTOR
///  i pel camp v reserva òmemria a n reals.
///
///  @param n Par. ent. Nombre de components.
///
///  @return
/////////////////////////////////////////////////////

VECTOR* crear_vector(int n);

/////////////////////////////////////////////////////
///  @brief Allibera un vector d'n components.
///
///  Primer allibera òmemria dels n components del camp v
///  del vector i édesprs allibera la òmemria d'aquest
///  darrer.
///
///  @param vector Par. ent/sortÉ. s el vector que volem
///                alliberar.
/////////////////////////////////////////////////////

void alliberar_vector(VECTOR** vector);

/////////////////////////////////////////////////////
///  @brief Producte d'un escalar per un vector.
///
///  Realitza el producte d'un escalar per un vector.
///
///  @param vector Par. ent/sort. Vector de n components.
///
///  @param k Par. ent. Valor escalars.
/////////////////////////////////////////////////////

VECTOR* escalar_per_vector(VECTOR* vector,real k);

/////////////////////////////////////////////////////
///  @brief S'éobt la resta entre dos vectors.
///
///  S'éobt la resta entre dos vectors.
///
///  @param vector1 Par. ent. Vector d' n components.
///
///  @param vector2 Par. ent. Vector d' n components.
///
///  @return El vector resta.
/////////////////////////////////////////////////////

VECTOR* restar_vectors(VECTOR* vector1,VECTOR* vector2);

/////////////////////////////////////////////////////
///  @brief S'éobt la suma entre dos vectors.
///
///  S'éobt la suma entre dos vectors.
///
///  @param vector1 Par. ent. Vector d' n components.
///
///  @param vector2 Par. ent. Vector d' n components.
///
///  @return El vector suma.
/////////////////////////////////////////////////////
```

```
VECTOR* sumar_vectors(VECTOR* vector1,VECTOR* vector2);

/////////////////////////////////////////////////////
///  @brief S'éobt el òmdul d'un vector.
///
///  S'éobt el òmdul d'un vector.
///
///  @param vector Par. ent. Vector d' n components.
///
///  @return El òmdul queé s un real.
/////////////////////////////////////////////////////

real modul_vector(VECTOR* vector);

/////////////////////////////////////////////////////
///  @brief S'éobt un vector de longitud lambda.
///
///  S'éobt el òmdul d'un vector de longitud lambda.
///
///  @param vector Par. ent. Vector d' n components.
///
///  @param landa Par. ent. Longitud del pas d'çavan.
///
///  @return El vector òper amb longitud lambda.
/////////////////////////////////////////////////////

VECTOR* vector_long_landa(VECTOR* vector,real landa);

//operacions consultores

/////////////////////////////////////////////////////
///  @brief S'escriu un vector per pantalla.
///
///  S'escriu les components d'un vector
///  pel canal estandar de sortida.
///
///  @param vector Par. ent. Vector d' n components.
///
/////////////////////////////////////////////////////

void escriure_vector(VECTOR* vector);

/////////////////////////////////////////////////////
///  @brief Calcula l'óequaci vectorial de la recta discretitzada.
///
///   Calcula l'óequaci vectorial de la recta discretitzada
///   del la ósecci de la èmmoria 5.4.2.2.
///
///  @param n Par. ent. Volem anar al pas enessim.
///
///  @param vector_ini Par. ent. Vector o punt d'inci de
///                    de la recta.
///
///  @param vector_landa Vector de longitud lambda.
///
///  @return El vector de la recta en la óiteraci èenssima.
/////////////////////////////////////////////////////

VECTOR* vector_enessim(int n,VECTOR* vector_ini,VECTOR* vector_landa);

/////////////////////////////////////////////////////
///  @brief Calcula el nombre d'iteracions.
///
///  Calcula el nombre de passos de la recta discretitzada
///
///  @param vector_ini Par. ent. Vector o punt d'inci de
///                    de la recta.
///
///  @param vector_fi Par. ent. Vector o punt de final de la
///                   recta.
///
///  @param landa Par. ent. Longitud del pas d'çavan.
///
///  @return El nombre de passos que hem discretitzat la recta.
/////////////////////////////////////////////////////

int ultima_iteracio(VECTOR* vector_ini,VECTOR* vector_fi,real landa);

#endif
```

H.50 vector.c

```
#include "vector.h"

VECTOR* crear_vector(int n)
{
  VECTOR* vector;
  int i;

  if (NEW(vector,1,VECTOR)==NULL)
```

```
    cerror("error de malloc en crear vector()\n");
  if (NEW(ELEM(vector),n,real)==NULL)
    cerror("error de malloc en crear vector()\n");
  DIM(vector)=n;
  for(i=0;i<=(n-1);i++)
    ELEM_i(vector,i)=-10;

  return vector;
}
void alliberar_vector(VECTOR** vector)
{
  free(ELEM((*vector)));
  ELEM((*vector))=NULL;
  free((*vector));
  *vector=NULL;
}
VECTOR* escalar_per_vector(VECTOR* vector,real k)
{
  VECTOR* vectorf;
  int i,n;

  n=DIM(vector);
  vectorf=crear_vector(n);
  for(i=0;i<=(n-1);i++)
    ELEM_i(vectorf,i)=k*(ELEM_i(vector,i));
  return vectorf;

}
VECTOR* restar_vectors(VECTOR* vector1,VECTOR* vector2)
{
  int n;
  int i;
  VECTOR* vector;

  if ((DIM(vector1))!=(DIM(vector2)))
    {
      cerror("error en restar_vectors()\n");
      return NULL;
    }
  else
    {
      n=DIM(vector1);
      vector=crear_vector(n);
      for(i=0;i<=n-1;i++)
          ELEM_i(vector,i)=((ELEM_i(vector2,i))-(ELEM_i(vector1,i)));
      return vector;
    }
}
VECTOR* sumar_vectors(VECTOR* vector1,VECTOR* vector2)
{
   int n;
  int i;
  VECTOR* vector;

  if ((DIM(vector1))!=(DIM(vector2)))
    {
      cerror("error en restar_vectors()\n");
      return NULL;
    }
  else
    {
      n=DIM(vector1);
      vector=crear_vector(n);
      for(i=0;i<=n-1;i++)
          ELEM_i(vector,i)=((ELEM_i(vector1,i))+(ELEM_i(vector2,i)));
      return vector;
    }

}

real modul_vector(VECTOR* vector)
{
  int i,n;
  real suma_q;
  n=DIM(vector);
  suma_q=0.0;
  for(i=0;i<=n-1;i++)
    suma_q=suma_q+pow(ELEM_i(vector,i),2.0);
  return (sqrt(suma_q));
}

VECTOR* vector_long_landa(VECTOR* vector,real landa)
{
  int i,n;
  real modul;
  VECTOR* vectorla;
  n=DIM(vector);
  vectorla=crear_vector(n);
  modul=modul_vector(vector);
  for(i=0;i<=(n-1);++i)
    ELEM_i(vectorla,i)=(landa*(ELEM_i(vector,i)))/modul;
  return vectorla;
```

```
}
void escriure_vector(VECTOR* vector)
{
  int i,n;
  n=DIM(vector);
  printf("la longitud del vector es: %d\n",n);
  for(i=0;i<=(n-1);++i)
    printf("%f\n",ELEM_i(vector,i));
}
/* operacions de mes alt nivell que utilitzen el tipus vector
 */

VECTOR* vector_enessim(int n,VECTOR* vector_ini,VECTOR* vector_landa)
{
  VECTOR* vector_aux1;
  VECTOR* vector_aux2;

  if ((DIM(vector_ini))!=(DIM(vector_landa)))
    {
      cerror("error en restar_vectors()\n");
      return NULL;
    }
  else
    {
      vector_aux1=escalar_per_vector(vector_landa,(real)n);
      vector_aux2=sumar_vectors(vector_ini,vector_aux1);
      alliberar_vector(&vector_aux1);
      return (vector_aux2);
    }
}

int ultima_iteracio(VECTOR* vector_ini,VECTOR* vector_fi,real landa)
{
  VECTOR* vector_aux;
  real modul;

  vector_aux=restar_vectors(vector_ini,vector_fi);
  modul=modul_vector(vector_aux);
  alliberar_vector(&vector_aux);
  return((int)(modul/landa));
}
```

H.51 vei.h

```
#ifndef VEI_H
#define VEI_H

/////////////////////////////////////////////////////
/// @file vei.h
/// @brief Calcula els g_NVEINS éms propers.
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///     l'estructura TAULA_VERTEX_VEINS permet
///      emmagatzemar els g_NVEINS éms propers d'un èvrtex
///      c insertat.
/////////////////////////////////////////////////////

#include "globals.h"
#include "macros_generals.h"
#include "graf.h"

typedef real ARESTA;

typedef struct struct_vei
{
  ARESTA aresta;
  list pvertex;
  vertex num_vertex;
}INFO_VEI;

typedef struct struct_taula_vert_veins{
  int n;
  INFO_VEI* t;
}TAULA_VERTEX_VEINS;

#define NO_ULTIM_VEI(vertex_veins,i) ((i<g_NVEINS)&&((vertex_veins->t)[i].aresta!=BIG))

TAULA_VERTEX_VEINS* crear_taula_vertex_veins();
int index_distancia_maxima(TAULA_VERTEX_VEINS* vertex_veins);
int compare (const void * a, const void * b);
void inicialitzar_vertex_veins(TAULA_VERTEX_VEINS* vertex_veins);
void escriure_info_vertex_veins(TAULA_VERTEX_VEINS* vertex_veins);
void desar_info_vertex_veins_a_fitxer_depuracio(vertex num_vertex_c,TAULA_VERTEX_VEINS* vertex_veins);
#endif
```

H.52 vei.c

```
#include "vei.h"

TAULA_VERTEX_VEINS* crear_taula_vertex_veins()
```

```
{
  TAULA_VERTEX_VEINS* vertex_veins;
  int i;

  if(NEW(vertex_veins,1,TAULA_VERTEX_VEINS)==NULL)
    cerror("Error en crear taula vertex veins()");
  if(NEW(vertex_veins->t,g_NVEINS,INFO_VEI)==NULL)
    cerror("Error en crear taula vertex veins()");

  vertex_veins->n=g_NVEINS;

  for(i=0;i<vertex_veins->n;i++)
    {
      (vertex_veins->t)[i].aresta=BIG;
      (vertex_veins->t)[i].pvertex=NULL;
      (vertex_veins->t)[i].num_vertex=BIG;
    }
  return vertex_veins;
}

void inicialitzar_vertex_veins(TAULA_VERTEX_VEINS* vertex_veins)
{
  int i;

  //La longitud torna a ser la àmxima.
  vertex_veins->n=g_NVEINS;

  for(i=0;i<vertex_veins->n;i++)
    {
      (vertex_veins->t)[i].aresta=BIG;
      (vertex_veins->t)[i].pvertex=NULL;
      (vertex_veins->t)[i].num_vertex=BIG;
    }
}

int index_distancia_maxima(TAULA_VERTEX_VEINS* vertex_veins)
{
  int i,imax;

  imax=0;
  for(i=1;i<g_NVEINS;i++)
    if ((vertex_veins->t)[i].aresta>(vertex_veins->t)[imax].aresta)
      imax=i;
  return imax;
}
int compare (const void * a, const void * b)
{
  if (((const INFO_VEI*)a)->aresta<((const INFO_VEI*)b)->aresta)
    return(-1);
  else if (((const INFO_VEI*)a)->aresta>((const INFO_VEI*)b)->aresta)
    return(1);
  else
    return(0);
}


void escriure_info_vertex_veins(TAULA_VERTEX_VEINS* vertex_veins)
{
  int i;
  for (i=0;i<vertex_veins->n; i++)
    {
      printf ("EL VALOR DE LA DISTANCIA DE LA TAULA DE VEINS ES:%f\n",(vertex_veins->t)[i].aresta);
      printf ("EL VALOR DEL PUNTER DEL VERTEX ES VERTEX ES:%p\n",(vertex_veins->t)[i].pvertex);
      printf ("EL VALOR DEL VERTEX ES:%d\n",(vertex_veins->t)[i].num_vertex);
    }
}

void desar_info_vertex_veins_a_fitxer_depuracio(vertex num_vertex_c,TAULA_VERTEX_VEINS* vertex_veins)
{
  int i;

  TRACE1((g_fd,"Taula de èvrtex veins al èvrtex %d:\n",num_vertex_c));
  for(i=0;i<vertex_veins->n;i++)
    {
      TRACE1((g_fd,"distancia, pvertex, vertex, del vei"));
      TRACE1((g_fd,"%d",(i+1)));
      TRACE1((g_fd,"%f",(vertex_veins->t)[i].aresta));
      TRACE1((g_fd,"%p",(vertex_veins->t)[i].pvertex));
      TRACE1((g_fd,"%d",(vertex_veins->t)[i].num_vertex));
      TRACE1((g_fd,"\n"));
    }
  TRACE1((g_fd,"\n"));
}
```

H.53 vertex_exp.h

```
#ifndef VERTEXS_EXP_H
#define VERTEXS_EXP_H

//////////////////////////////////////////////////////
/// @file vertex_exp.h
```

```
/// @brief Calcula els g_NVEINS éms propers.
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///      L'estructura TAULA_VERTEX_VEINS permet
///      emmagatzemar els g_NVEINS éms propers d'un èvrtex
///      c insertat.
////////////////////////////////////////////////////


#include "globals.h"
#include "macros_generals.h"
#include "graf.h"

typedef real PES;

typedef struct struct_vert_exp
{
  PES pes;
  list pvertex;
}VERTEX_EXP;

typedef struct struct_taula_vert_exp{
  int n;
  VERTEX_EXP* t;
}TAULA_VERTEX_EXP;

#define NO_ULTIM_VERTEX_EXP(vertexs_expand,i) ((i<(vertexs_expand->n))&&(((vertexs_expand->t)[i].pvertex)!=NULL))

////////////////////////////////////////////////////////////////
// crear_taula_vertexs_expand()
//
// Aquesta rutina crea una taula de èvrtexs a expansionar del
// graf de configuracions. La longitud de la taulaé s un
// percentatge del nombre total de èvrtexs de l'etapa de
// óconstrucci.
//
// Arguments:
//   void
// Retorna:
//   Una taula on cada ·cellaé s del tipus VERTEX_EXP.
//
////////////////////////////////////////////////////////////////

TAULA_VERTEX_EXP* crear_taula_vertexs_expand();

////////////////////////////////////////////////////////////////
//  compare_pesos()
//
// Rutina que estableix el criteri d'óordenaci de la taula
// segons el camp pes. S'àutilitzar la ófunci qsort per
// ordenar la taula segons el camp pes.
//
// Arguments:
//   a, b:
//    Son óds punters que apunten a una zona que no
//    es pot modificar i queé s de tipus void.
// Retorna:
//    Retorna un enter: 1,-1,0, segons quin dels criteris
//    compleixen els dos pesos ïintroduts.
//
////////////////////////////////////////////////////////////////

int compare_pesos (const void * a, const void * b);

////////////////////////////////////////////////////////////////
// index_pes_minim()
//
// Rutina que busca lí'ndex de la taula que écont el pes ímnim.
// Arguments:
//    vertexs_expand:
//      Es un punter a la primera óposici de la taula.
// Retorna:
//    Retorna lí'ndex de la taula que écont el pes ímnim.
//
////////////////////////////////////////////////////////////////

int index_pes_minim(TAULA_VERTEX_EXP* vertexs_expand);

////////////////////////////////////////////////////////////////
// escriure_info_vertexs_expand()
//
// Rutina que escriu per pantalla totes les propietats de cada
// ·cella de la taula de èvrtexs a expansionar.
// Arguments:
//    vertexs_expand:
//      Es un punter a la primera óposici de la taula.
// Retorna:
//    void.
//
////////////////////////////////////////////////////////////////
```

```
void escriure_info_vertexs_expand(TAULA_VERTEX_EXP* vertexs_expand);
#endif
```

H.54 vertex_exp.c

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// vertexs_exp.c
//
// Autor:    Gorka Bonals
// Creat:    Juliol 2005
// Revisat: Agost 2005
// óVersi: 1.0
//
// vertex_exp.cé s el òmdul que écont les rutines que treballen
// sobre una taula àdinmica, que àcontindr la óinformaci dels èvrtexs
// del graf de configuracions que expandirem.
//
//
// rutines úpbliques:
//
// -crear_taula_vertexs_expand()
//     Rutina que crea una taula dels èvrtexs que voldrem expandir.
// -alliberar_taula_vertexs_expand()
//     Rutina que allibera la òmemria reservada per la rutina anterior.
// -compare_pesos()
//     Rutina que serveix per establir el criteri de ócomparaci dels
//     pesos, que s'utilitza en la ófunci qsort().
// -index_pes_minim()
//     Rutina que busca lí'ndex de la taula que ét el pes ímnim.
// -escriure_info_vertexs_expand()
//     Rutina que escriu la óinformaci de la taula de èvrtexs a
//     expansionar.
//
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

#include "vertexs_exp.h"

TAULA_VERTEX_EXP* crear_taula_vertexs_expand()
{
  TAULA_VERTEX_EXP* vertexs_expand;
  int i,num_vertexs_expand;

  //Els nombre de èvrtexs a expandiré s un percentatge del
  //nombre àmxim de èvrtexs generats en el graf.
  num_vertexs_expand=(int)(((real)g_PERC_EXP*0.01)*g_MAX_NODES);

  if(NEW(vertexs_expand,1,TAULA_VERTEX_EXP)==NULL)
    cerror("error de malloc en crear_taula_vertexs_expan()\n");

  if(NEW(vertexs_expand->t,num_vertexs_expand,VERTEX_EXP)==NULL)
    cerror("error de malloc en crear_taula_vertexs_expan()\n");

  vertexs_expand->n=num_vertexs_expand;

  //Inicialitzem la taula a uns valors per defecte.
  for(i=0;i<num_vertexs_expand;i++)
    {
      (vertexs_expand->t)[i].pes=-1.0;
      (vertexs_expand->t)[i].pvertex=NULL;
    }
  return vertexs_expand;
}

int compare_pesos (const void * a, const void * b)
{
  if (((const VERTEX_EXP*)a)->pes<((const VERTEX_EXP*)b)->pes)
    return(1);
  else if (((const VERTEX_EXP*)a)->pes>((const VERTEX_EXP*)b)->pes)
    return(-1);
  else
    return(0);
}

int index_pes_minim(TAULA_VERTEX_EXP* vertexs_expand)
{
  int i,imin,num_vertexs_expand;

  imin=0;

  //àCIcul novament del nombre de èvrtexs a expandir.
  num_vertexs_expand=vertexs_expand->n;

  //Fem un recorregut per tots els elements de la taula.
  for(i=1;i<num_vertexs_expand;i++)
    if ((vertexs_expand->t)[i].pes<(vertexs_expand->t)[imin].pes)
      imin=i;
  return imin;
}
```

```
void escriure_info_vertexs_expand(TAULA_VERTEX_EXP* vertexs_expand)
{
  int i;

  //Fem un recorregut per tots els elements de la taula de
  //èvrtexs a extendre
  i=0;
  while (i<vertexs_expand->n)
    {
      printf ("pos. taula: %d, pes: %f, n. vertex: %d\n",i,(vertexs_expand->t)[i].pes,
              VERTEX_NUMBER((vertexs_expand->t)[i].pvertex));

      i=i+1;
    }
}
```

H.55 webots_interface.h

```
#ifndef WEBOTS_INTERFACE_H
#define WEBOTS_INTERFACE_H

/////////////////////////////////////////////////////
/// @file webots_interface.h
/// @brief éCont les rutines interactuen amb les de webots.
/// @author  Gorka Bonals Sastre
/// @version 1.0
/// @date Agost 2005
/// @par óDescripici
///       Encapsula les rutines que utilitza
///       webots per interactuar amb els
///       models.
/////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <device/robot.h>
#include <device/supervisor.h>
#include "noms_gll.h"
#include "general.h"
#include <gtk/gtk.h>

#ifdef _MAIN_
#  define GLOBAL_WEBOTS
#else
#   define GLOBAL_WEBOTS extern
#endif

GLOBAL_WEBOTS real angle_rob_esq[6];
GLOBAL_WEBOTS real angle_rob_dret[6];
GLOBAL_WEBOTS real transl_objecte[4];
GLOBAL_WEBOTS real rot_objecte[4];
GLOBAL_WEBOTS real transl_grasp_esq[4];
GLOBAL_WEBOTS real rot_grasp_esq[4];
GLOBAL_WEBOTS real transl_grasp_dret[4];
GLOBAL_WEBOTS real rot_grasp_dret[4];

GLOBAL_WEBOTS real rot_x_rob_esq;
GLOBAL_WEBOTS real transl_rot_rob_esq[7];
GLOBAL_WEBOTS real transl_base0_rob_esq[3];

GLOBAL_WEBOTS real rot_x_rob_dret;
GLOBAL_WEBOTS real transl_rot_rob_dret[7];
GLOBAL_WEBOTS real transl_base0_rob_dret[3];


#define PAS_SIM 1

typedef struct camps{
  char* nom;
  NodeRef ptr;
}PCAMPS;

/////////////////////////////////////////////////////
//
// Precondicions:
//
//
// webots_init()
//
// Funcio on encapsulem totes les tasques d'inicialitzacio
// relacionades amb el programa Webots, i
// que no corresponen a cap desenvolupament d'un algorisme
//
// Paramatres:
//
// valor de retorn: buit
//
// Variables globals:
```

```
//
// g_pas_sim:Es una variable de tipus unsigned short, i es
// per indicar el pas de simulacio.
//
// Postcondicions:
//
// Tenim l'entorn de webots carregat, i disposem de tots
// els punters als objectes de l'entorn que
// volguem moure en fases posteriors.
//
//
////////////////////////////////////////////////////
void webots_init(void);

////////////////////////////////////////////////////
//
// Precondicions:
//
//
// webots_end()
//
// Funcio on encapsulem totes les tasques de finalitzacio
// relacionades amb el programa Webots, i
// que no corresponen a cap desenvolupament de l'algoritme
// de planificacio de trajectories ni de
// inicialitzacio de la interficie grafica.
//
// Paramatres:
//
// valor de retorn: buit
//
// Variables globals:
//
//
//
// Postcondicions:
//
// Bugs:No s'han detectat bugs encara.
//
////////////////////////////////////////////////////

void webots_end(void);

void avancar_simulacio(void);

////////////////////////////////////////////////////
//
// Accio on trobem tots els punters dels camps de la scene
// tree que ens interessara llegir o canviar al llarg de
// la simulacio.
//
// Paramatres:
//
// valor de retorn: buit
//
// Variables globals:
//
// g_punters_noms: Es una taula que conte en cada posicio
// el nom d'un camp de la scene tree i el  seu punter
// associat, que es pot utilitzar per canviar certes variables
// d'aquell camp a partir  d'unes funcions de webots.
//
////////////////////////////////////////////////////

void adquisicio_punters_scene_tree(void);

////////////////////////////////////////////////////
//
// Es una accio per encapsular l'accio de canviar un angle
// çmitjanant webots.D'aquesta forma  a partir del nom
// de l'objecte l'usuari es çapa de canviar l'angle òcmodament.
//
//
// Paramatres:
//
// valor de retorn: buit
//
// Variables globals:
//
// S'ha de recordar que si volem moure els objectes segons
// aquesta accio, s'haura d'aplicar fora d'aquesta un pas
// de simulacio es a dir, executar un avancar_simulacio().
////////////////////////////////////////////////////

void canviar_angle_solid (char* nom_solid, real* angle);
void obtenir_angle_solid (char* nom_solid, real* angle);
void obtenir_rotacio_solid (char* nom_solid, real* rot);
void obtenir_translacio_rotacio_solid(char* nom_solid,real* transl_rot);
void aplicar_translacio_x_solid(char* nom_solid,real* transl_x);
void obtenir_translacio_x_solid(char* nom_solid,real* transl_x);
void aplicar_translacio_y_solid(char* nom_solid,real* transl_y);
void obtenir_translacio_y_solid(char* nom_solid,real* transl_y);
```

```
void aplicar_translacio_z_solid(char* nom_solid,real* transl_z);
void obtenir_translacio_z_solid(char* nom_solid,real* transl_z);
void aplicar_translacio_solid(char* nom_solid,real* translacio);
void obtenir_translacio_solid(char* nom_solid,real* translacio);
void escriure_text_a_scenee_tree(char* text);
NodeRef nom_a_punter(char* nom_solid);
int cadenes_iguals(char* cad1, char* cad2);

#endif
```

H.56 webots_interface.c

```
#include "webots_interface.h"

#define NGRAUS_LLIBERTAT ((int)(sizeof(g_punters_noms)/sizeof(PCAMPS)))

PCAMPS g_punters_noms[]={{ART_ESQ_1,NULL},{ART_ESQ_2,NULL},
                         {ART_ESQ_3,NULL},{ART_ESQ_4,NULL},
                         {ART_ESQ_5,NULL},{ART_ESQ_6,NULL},
                         {ART_DRET_1,NULL},{ART_DRET_2,NULL},
                         {ART_DRET_3,NULL},{ART_DRET_4,NULL},
                         {ART_DRET_5,NULL},{ART_DRET_6,NULL},
                         {TRANSL_OBJECTE, NULL},{ROT_X_OBJECTE,NULL},
                         {ROT_Y_OBJECTE,NULL},{ROT_Z_OBJECTE,NULL},
                         {TRANSL_ROB_ESQ,NULL},{ROT_ROB_ESQ,NULL},
                         {TRANSL_BASE0_ROB_ESQ,NULL},{TRANSL_ROB_DRET,NULL},
                         {ROT_ROB_DRET,NULL},{TRANSL_BASE0_ROB_DRET,NULL},
                         {TRANSL_TRIEDRE_ESQ_OBJCT,NULL},{ROT_X_TRIEDRE_ESQ_OBJCT,NULL},
                         {ROT_Y_TRIEDRE_ESQ_OBJCT,NULL},{ROT_Z_TRIEDRE_ESQ_OBJCT,NULL},
                         {TRANSL_TRIEDRE_DRET_OBJCT,NULL},{ROT_X_TRIEDRE_DRET_OBJCT,NULL},
                         {ROT_Y_TRIEDRE_DRET_OBJCT,NULL},{ROT_Z_TRIEDRE_DRET_OBJCT,NULL}};



//////////////////////////////////////////////////////
//
// Precondicions:
//
//
// reset()
//
// Funcio que s'aplica abans de comencar la simulacio de
// webots.
//
// Paramatres:
//
// valor de retorn: buit
//
// Variables globals:
//
// PAS_SIM:Es una variable de tipus unsigned short, i es per indicar el pas de simulacio.
//
// Postcondicions:
//
//
// Bugs:No s'han detectat bugs encara.
//
//////////////////////////////////////////////////////
void reset()
{
  //g_pas_sim=1;
}
void webots_init(void)
{
  robot_live(reset);
  avancar_simulacio();
  adquisicio_punters_scene_tree();
}
void webots_end(void)
{
}
void avancar_simulacio()
{
  robot_step(PAS_SIM);
  gtk_main_iteration_do(FALSE);
}
void adquisicio_punters_scene_tree()

{
  int i;
  NodeRef ptr;

  printf("%d\n",NGRAUS_LLIBERTAT);
  for(i=0;i<NGRAUS_LLIBERTAT;i++)
    {
      ptr = supervisor_node_get_from_def(g_punters_noms[i].nom);
      //ptr pot ser no NULL encara que no s'hagi trobat.
      //El robot_step es necessari per que es refresquin
      //els punters als camps, i aplicar la funcio de
      //comprovacio seguent.
```

```
      robot_step(PAS_SIM);
      if (supervisor_node_was_found(ptr))
        {
          // Cas en que si trobem l'objecte.
          g_punters_noms[i].ptr = ptr;
          //printf("%d\n",supervisor_node_was_found(ptr));
        }
      else
        {
          // Cas en que no trobem l'objecte a l'scene tree
          printf("Error a adquisicio punters scene tree() elem %d\n",i);
          exit(-1);
        }
    }
}
//-------------------------------------------------------
void canviar_angle_solid (char* nom_solid, real* angle)

{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_set(psolid,
                       SUPERVISOR_FIELD_ROTATION_ANGLE,
                       angle);
}
void obtenir_angle_solid (char* nom_solid, real* angle)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_ROTATION_ANGLE,
                       (void*)angle,PAS_SIM);
}
void obtenir_rotacio_solid (char* nom_solid, real* rot)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_ROTATION,
                       (void*)rot,PAS_SIM);
}
void obtenir_translacio_rotacio_solid(char* nom_solid,real* transl_rot)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_AND_ROTATION,
                       (void*)transl_rot,PAS_SIM);

}

void aplicar_translacio_x_solid(char* nom_solid,real* transl_x)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_set(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_X,
                       transl_x);
}
void obtenir_translacio_x_solid(char* nom_solid,real* transl_x)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_X,
                       (void*)transl_x,PAS_SIM);
}
void aplicar_translacio_y_solid(char* nom_solid,real* transl_y)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_set(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_Y,
                       transl_y);
}
void obtenir_translacio_y_solid(char* nom_solid,real* transl_y)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_Y,
                       (void*)transl_y,PAS_SIM);
}
void aplicar_translacio_z_solid(char* nom_solid,real* transl_z)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_set(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_Z,
                       transl_z);
}
```

```
void obtenir_translacio_z_solid(char* nom_solid,real* transl_z)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_TRANSLATION_Z,
                       (void*)transl_z,PAS_SIM);
}

void aplicar_translacio_solid(char* nom_solid,real* translacio)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_set(psolid,
                       SUPERVISOR_FIELD_TRANSLATION,
                       translacio);
}
void obtenir_translacio_solid(char* nom_solid,real* translacio)
{
  NodeRef psolid;
  psolid=nom_a_punter(nom_solid);
  supervisor_field_get(psolid,
                       SUPERVISOR_FIELD_TRANSLATION,
                       (void*)translacio,PAS_SIM);

}
void escriure_text_a_scenee_tree(char* text)
{
  supervisor_set_label(0,text,0.3,0,0.05,0x00ff0000);
  avancar_simulacio();
}
NodeRef nom_a_punter(char* nom_solid)
{
  int i;

  for(i=0;i<NGRAUS_LLIBERTAT;i++)
    if(cadenes_iguals(nom_solid,g_punters_noms[i].nom))
      break;

  if(i==NGRAUS_LLIBERTAT)
    {
      printf("Error_a_nom_a_punter::_punter_no_trobat\n");
      exit(-1);
    }

  return(g_punters_noms[i].ptr);
}
int cadenes_iguals(char* cad1,char* cad2)
{
  return(!strcmp(cad1,cad2));
}
```

H.57 utils.c

```
/*
 *  utils.c
 *  Lluis Ros
 *  Creation: April 1995
 *  Revisiom: 28  July 1995
 *
 *  Some utilities
 *     bool is_one_of(unsigned int num, unsigned int* numbers,int nnumbers);
 *     void cerror(char* error);
 */

#include <stdio.h>
#include <stdlib.h>
#include "general.h"
#include "utils.h"
/*-------------------------------------------------------------------------------------------------------------*/
void cerror(char* error)
{
    printf("%s\n", error);
    printf("exiting...\n");
    exit(1);
}
/*-------------------------------------------------------------------------------------------------------------*/
bool is_one_of(unsigned int num, unsigned int* numbers,int nnumbers)
/*
 * returns TRUE if the number 'num' is one the elements of the vector numbers[0..nnumbers].
 */
{
    int i;
    for(i=0; i<nnumbers &&  numbers[i]!=num; i++);
    return(i<nnumbers);
}
/*-------------------------------------------------------------------------------------------------------------*/
```

H.58 wgraph.h

```
#ifndef WGRAPH_H
#define WGRAPH_H

/*      wgraph.h

        Header file for weighted graph type

        by Steven Skiena
*/

/*
Copyright 2003 by Steven S. Skiena; all rights reserved.

Permission is granted for use in non-commerical applications
provided this copyright notice remains intact and unchanged.

This program appears in my book:

"Programming Challenges: The Programming Contest Training Manual"
by Steven Skiena and Miguel Revilla, Springer-Verlag, New York 2003.

See our website www.programming-challenges.com for additional information.

This book can be ordered from Amazon.com at

http://www.amazon.com/exec/obidos/ASIN/0387001638/thealgorithmrepo/

*/

#include "general.h"


#define MAXV            100             /* maximum number of vertices */
#define MAXDEGREE       100             /* maximum outdegree of a vertex */
#define MAXINT  100007
#define MAX_LONG        100

typedef struct {
  int v;                                /* neighboring vertex */
  real weight;                  /* edge weight */
  bool lliure;
} edge;

typedef struct {
  edge edges[MAXV+1][MAXDEGREE];        /* adjacency info */
  int degree[MAXV+1];           /* outdegree of each vertex */
  int nvertices;                        /* number of vertices in the graph */
  int nedges;                   /* number of edges in the graph */
} wgraph;

typedef int tau_cami[MAX_LONG];

typedef struct struct_cami{
 tau_cami v;
 int n;
}CAMI;

CAMI cami;

void initialize_wgraph(wgraph* g);
void read_wgraph(wgraph* g, bool directed);
void insert_edge_wgraph(wgraph* g,int x, int y, bool directed, real w);
void delete_edge_wgraph(wgraph* g, int x, int y, bool directed);
void afegir_lliure_aresta_wgraph(wgraph* g,int x, int y, bool directed,bool lliure);
bool llegir_lliure_aresta_wgraph(wgraph* g,int x, int y);
void print_wgraph(wgraph* g);
void initialize_search_wgraph(wgraph* g);
void dfs_wgraph(wgraph* g,int v);
void process_vertex_wgraph(int v);
void process_edge_wgraph(int x, int y);
void find_path_wgraph(int start, int end, int parents[]);
void connected_components_wgraph(wgraph* g);
void dijkstra_wgraph(wgraph *g, int start);

#endif
```

H.59 wgraph.c

```
/*      wgraph.c

        A generic weighted graph data type

        by Steven Skiena
*/


/*
Copyright 2003 by Steven S. Skiena; all rights reserved.
```

```
Permission is granted for use in non-commerical applications
provided this copyright notice remains intact and unchanged.

This program appears in my book:

"Programming Challenges: The Programming Contest Training Manual"
by Steven Skiena and Miguel Revilla, Springer-Verlag, New York 2003.

See our website www.programming-challenges.com for additional information.

This book can be ordered from Amazon.com at

http://www.amazon.com/exec/obidos/ASIN/0387001638/thealgorithmrepo/

*/

#include <stdio.h>
#include "wgraph.h"

void initialize_wgraph(g)
wgraph *g;                              /* graph to initialize */
{

        int i;                          /* counter */

        g -> nvertices = 0;
        g -> nedges = 0;

        for (i=1; i<=MAXV; i++) g->degree[i] = 0;
}

void read_wgraph(g,directed)
wgraph *g;                               /* graph to initialize */
bool directed;                          /* is this graph directed? */
{
        int i;                          /* counter */
        int m;                          /* number of edges */
        int x,y;                        /* placeholder for edge and weight */
        real w;

        initialize_wgraph(g);

        scanf("%d %d\n",&(g->nvertices),&m);

        for (i=1; i<=m; i++) {
                printf("%d\n",i);
                scanf("%d %d %f\n",&x,&y,&w);
                printf("hola\n");
                insert_edge_wgraph(g,x,y,directed,w);

        }
}

void insert_edge_wgraph(g,x,y,directed,w)
wgraph *g;                                      /* graph to initialize */
int x, y;                                       /* placeholder for edge (x,y) */
bool directed;                                  /* is this edge directed? */
real w;                                         /* edge weight */
{
        if (g->degree[x] > MAXDEGREE)
                printf("Warning: insertion(%d,%d) exceeds degree bound\n",x,y);

        g->edges[x][g->degree[x]].v = y;
        g->edges[x][g->degree[x]].weight = w;
        g->edges[x][g->degree[x]].lliure = FALSE;
        /*g->edges[x][g->degree[x]].in = FALSE;*/
        g->degree[x] ++;

        if (directed == FALSE)
                insert_edge_wgraph(g,y,x,TRUE,w);
        else
                g->nedges ++;
}
void afegir_lliure_aresta_wgraph(wgraph* g,int x, int y, bool directed,bool lliure)
{
  int i;

    for (i=0;i<g->degree[x];i++)
      if (g->edges[x][i].v == y)
        {
          g->edges[x][i].lliure=lliure;

          if (directed == FALSE)
            afegir_lliure_aresta_wgraph(g,y,x,TRUE,lliure);

        }
}

bool llegir_lliure_aresta_wgraph(wgraph* g,int x, int y)
{
  int i;
  bool lliure=FALSE;
```

```
  for (i=0;i<g->degree[x];i++)
    if (g->edges[x][i].v == y)
      lliure=(g->edges[x][i].lliure);

  return lliure;
}

void delete_edge_wgraph(g,x,y,directed)
wgraph *g;                              /* graph to initialize */
int x, y;                               /* placeholder for edge (x,y) */
bool directed;                          /* is this edge directed? */
{
        int i;                          /* counter */

        for (i=0; i<g->degree[x]; i++)
                if (g->edges[x][i].v == y) {
                        g->degree[x] --;
                        g->edges[x][i] = g->edges[x][g->degree[x]];

                        if (directed == FALSE)
                                delete_edge_wgraph(g,y,x,TRUE);

                        /*return;*/
                }

        //printf("Warning: deletion(%d,%d) not found in g.\n",x,y);
}

void print_wgraph(g)
wgraph* g;                               /* graph to print */
{
        int i,j;                        /* counters */

        for (i=1; i<=g->nvertices; i++) {
                printf("%d: ",i);
                for (j=0; j<g->degree[i]; j++)
                        printf(" %d",g->edges[i][j].v);
                printf("\n");
        }
}

bool processed[MAXV];           /* which vertices have been processed */
bool discovered[MAXV];          /* which vertices have been found */
int parent[MAXV];               /* discovery relation */


void initialize_search_wgraph(g)
wgraph *g;                              /* graph to traverse */
{
        int i;                          /* counter */

        for (i=1; i<=g->nvertices; i++) {
                processed[i] = FALSE;
                discovered[i] = FALSE;
                parent[i] = -1;
        }
}


void dfs_wgraph(g,v)
wgraph *g;                               /* graph to traverse */
int v;                                  /* vertex to start searching from */
{
        int i;                          /* counter */
        int y;                          /* successor vertex */

        discovered[v] = TRUE;
        process_vertex_wgraph(v);

        for (i=0; i<g->degree[v]; i++) {
                y = g->edges[v][i].v;
                if (discovered[y] == FALSE) {
                        parent[y] = v;
                        dfs_wgraph(g,y);
                } else
                        if (processed[y] == FALSE)
                                process_edge_wgraph(v,y);
        }

        processed[v] = TRUE;
}

void process_vertex_wgraph(v)
int v;                                  /* vertex to process */
{
        printf(" %d",v);
}

void process_edge_wgraph(x,y)
int x,y;                                /* edge to process */
{
```

```
}

void find_path_wgraph (start,end,parents)
int start;                              /* first vertex on path */
int end;                                /* last vertex on path */
int parents[];                          /* array of parent pointers */
{
        if ((start == end) || (end == -1))
             {
                cami.v[0]=start;
                //printf("\n%d",start);
        }
        else {
                find_path_wgraph(start,parents[end],parents);
                cami.v[cami.n]=end;
                //printf(" %d",end);
        }
  cami.n=cami.n+1;
}

void connected_components_wgraph(g)
wgraph *g;                              /* graph to analyze */
{
        int c;                          /* component number */
        int i;                          /* counter */

        initialize_search_wgraph(g);

        c = 0;
        for (i=1; i<=g->nvertices; i++)
                if (discovered[i] == FALSE) {
                        c = c+1;
                        printf("Component %d:",c);
                        dfs_wgraph(g,i);
                        printf("\n");
                }
}
void dijkstra_wgraph(g,start)                   /* WAS prim(g,start) */
wgraph* g;
int start;

{
        int i;                  /* counters */
        bool intree[MAXV];              /* is the vertex in the tree yet? */
        real distance[MAXV];            /* distance vertex is from start */
        int v;                          /* current vertex to process */
        int w;                          /* candidate next vertex */
        real weight;                    /* edge weight */
        real dist;                      /* best current distance from start */

        for (i=1; i<=g->nvertices; i++) {
                intree[i] = FALSE;
                distance[i] = MAXINT;
                parent[i] = -1;
        }

        distance[start] = 0;
        v = start;

        while (intree[v] == FALSE) {
                intree[v] = TRUE;
                for (i=0; i<g->degree[v]; i++) {
                        w = g->edges[v][i].v;
                        weight = g->edges[v][i].weight;
/* CHANGED */                   if (distance[w] > (distance[v]+weight)) {
/* CHANGED */                           distance[w] = distance[v]+weight;
/* CHANGED */                           parent[w] = v;
                        }
                }

                v = 1;
                dist = MAXINT;
                for (i=1; i<=g->nvertices; i++)
                        if ((intree[i] == FALSE) && (dist > distance[i])) {
                                dist = distance[i];
                                v = i;
                        }
        }
/*for (i=1; i<=g->nvertices; i++) printf("%d %d\n",i,distance[i]);*/
}
```

H.60 tau_list.h

```
#ifndef TAU_LIST_H
#define TAU_LIST_H

#include"ctnt_generals.h"
#include "graf.h"

typedef list tau_nodes[MAX_LONG+1];
```

```
typedef struct tau_vertexs{
  tau_nodes nodes;
  int n;
}TAU_LIST;

void traspassar_list_TAU_PLIST(list ptraject, TAU_LIST* t);

#endif
```

H.61 tau_list.c

```
#include"tau_list.h"

void traspassar_list_TAU_PLIST(list traject, TAU_LIST* t)
{
  list pnode=NULL;
  int i;

  t->n=length(traject);

  i=1;
  while ((pnode=list_iterator(traject,pnode))!=NULL)
    {
      t->nodes[i]=pnode;
      i=i+1;
    }
}
```

H.62 Gdata.h

```
#ifndef GDATA_H
#define GDATA_H

//oftsets pel robot
#ifndef JOFST1
#define JOFST1  3.14159265358979320
#endif

#ifndef JOFST2
#define JOFST2 -1.57079632679489660
#endif

#ifndef JOFST3
#define JOFST3  1.57079632679489660
#endif

#endif
```

H.63 generals.h

```
#ifndef GENERAL_H
#define GENERAL_H


/*
 *  general.h
 *  Lluis Ros
 *  Creation: April 1995
 *  Modificated: Gorka Bonals.
 *  Revision: 28 July
 *
 *  General purpose types and definitions
 */


#define OK 1
#define ERROR 0

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

typedef float real;
typedef int status;

typedef int  bool;

typedef void* generic_ptr;

#endif
```

H.64 ctnt_generals.h

```
#ifndef CTNT_GENERALS_H
#define CTNT_GENERALS_H

#define PIB2              1.57079632679489660     /* pi / 2              */
#define PI                3.14159265358979320     /* pi                  */
#define PIT2              6.28318530717958650     /* pi * 2              */
#define RADTODEG          57.2957795130823210     /* 180 / pi            */
#define DEGTORAD          0.01745329251994330     /* pi / 180            */
#define BIG   (1.e+5)                     /* considered as big    */
#define MAX_CADENA 100
#define MAX_LONG 100

//constants del PRM
#define NVEINS 30
#define MAX_DIST 3.3
#define MAX_NODES 50
#define PERC_EXP 30
#define MIN_NODES 100
#define CONT_SUP_DIST_MAX 100
#define CONT_NO_CAMI 20
#define MAX_NODES_CAMI 10

//constants de la peca
#define XMAX 0.34
#define XMIN -0.34
#define YMAX  -0.075
#define YMIN -1
#define ZMAX  0.6
#define ZMIN -0.6

#define GMIN -0.5
#define GMAX  0.5

#define ROT_XMAX PI
#define ROT_YMAX PI
#define ROT_ZMAX PI
#define ROT_XMIN -PI
#define ROT_YMIN -PI
#define ROT_ZMIN -PI

//constants dels parametres del robot

#define A2           0.290000000000000
#define A3           0.000000000000000
#define D3           -0.0490000000000
#define D4           0.310000000000000
#define D6           0.170000000000000
#define pas          0.1
#define pas_fi       0.001

#define JOFST1  180.*DEGTORAD
#define JMIN1  -160.*DEGTORAD
#define JMAX1   160.*DEGTORAD
#define JRNG1   320.*DEGTORAD
#define JOFST2 -90.*DEGTORAD
#define JMIN2  -127.5*DEGTORAD
#define JMAX2   127.5*DEGTORAD
#define JRNG2   255.*DEGTORAD
#define JOFST3  90.*DEGTORAD
#define JMIN3  -134.5*DEGTORAD
#define JMAX3   134.5*DEGTORAD
#define JRNG3   269.*DEGTORAD
#define JOFST4  0.0
#define JMIN4  -180.*DEGTORAD
#define JMAX4   180.*DEGTORAD
#define JRNG4   360.*DEGTORAD
#define JOFST5  0.0
#define JMIN5  -97.4*DEGTORAD
#define JMAX5   120.5*DEGTORAD
#define JRNG5   230* DEGTORAD
#define JOFST6  0.0
#define JMIN6  -180.* DEGTORAD
#define JMAX6   180.* DEGTORAD
#define JRNG6   360.*DEGTORAD

#define PAS_CAMI_LOCAL 0.045
#define PAS_ANGULAR 0.5
#define PAS_TRANSL 0.05
#define MAX_INTENTS 20
#define MAX_PROFUND 10
#define NUM_COMPONENTS 20

//constants de la fase de cerca

#define N_PASOS 20
#define N_FORA_DIST_MAX 100
#define MAX_NO_CAMI_LOCAL 20
```

```
#endif
```

H.65 funcions_generals.h

```
#ifndef FUNCIONS_GENERALS_H
#define FUNCIONS_GENERALS_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "ctnt_generals.h"
#include "general.h"

real valor_aleatori(real a, real b);
real quadrat_obertura(real a,real b);
real percentatge(int cont,int cont_total);
real convertir_angle_real_a_pi_pi(real th);
real rangle_positiu(real a);
void copiar_taula_ini_a_taula_fi(real* taula_ini,real* taula_fi);
void escriure_evolucio_fase(char* nom_fase,int vertex_c,int max_nodes);
#endif
```

H.66 funcions_generals.c

```
#include "funcions_generals.h"

real valor_aleatori(real a, real b)
{
  real aux;

  aux=((b-a)*((real)rand()/RAND_MAX)+a);
  //printf("aux:%f_a:%f_b:%f\n",aux,a,b);

  return((b-a)*((real)rand()/RAND_MAX)+a);
}
real quadrat_obertura(real a,real b)
{
  real xg,xp;

  if (a<b)
    {
      xg=b;
      xp=a;
    }
  else
    {
      xg=a;
      xp=b;
    }
  return(pow((xg-xp),2.0));
}
real percentatge(int cont,int cont_total)
{
  return (100.0*((real)(cont))/((real)cont_total));
}
real convertir_angle_real_a_pi_pi(real th)
{
  real th_pos;
  th_pos=rangle_positiu(th);
  if (th_pos>PI)
    {
      /* transformar l'angle de 0 -180*/
      th_pos=th_pos-PIT2;

    }
  /* transformar l'angle de 0 180*/
  return th_pos;
}
real rangle_positiu(real a)
{
  //printf("l'angle_abans_d'entrar_a_dangle_positiu()\n");
  while (a<0.)
    {
      a += PIT2;
    }
  while (a>PIT2)
    {
      a-=PIT2;
    }
  //printf("l'angle_despres_d'entrar_a_dangle_positiu()\n");
  return (a);

}
void copiar_taula_ini_a_taula_fi(real* taula_ini,real* taula_fi)
{
  int i;
```

```
  for(i=0;i<=5;i++)
    taula_fi[i]=taula_ini[i];
}
void escriure_evolucio_fase(char* nom_fase,int vertex_c,int max_nodes)
{
  int percen_act,i;
  static int percen_previ=0;

  percen_act=(int)(percentatge(vertex_c,max_nodes));
  //printf("%d%%\n",(int)percen_act);
  if (percen_act>percen_previ)
    {
      system("clear");
      printf("Fase de %s del graf\n\n",nom_fase);
      printf("Executant-se...\n");
      for(i=1;i<=percen_act;i++)
        printf("|");
      printf("%d%%\n",(int)percen_act);
      percen_previ=percen_act;
      //quan s'ha acabat de generar el graf
      if (percen_previ==100)
        percen_previ=0;
    }
}

/* void temps_comput(struct tms t_ini,struct tms t_end)  */
/* { */
/*   double ts; */

/*   ts=(double)sysconf(_SC_CLK_TCK); //tics per second */
/*   tm=(double)((t_end.tms_utime-t_init.tms_utime)+ */
/*            (t_end.tms_stime-t_init.tms_stime)+ */
/*            (t_end.tms_cutime-t_init.tms_cutime)+ */
/*            (t_end.tms_cstime-t_init.tms_cstime))/ts; */
/*   printf("Temps d'éexecuci: %f\n",tm); */
/* } */
```

H.67 macros_generals.h

```
#ifndef MACROS_GENERALS_H
#define MACROS_GENERALS_H

#include <stdio.h>
#include <stdlib.h>
#include "utils.h"

#ifdef DEPURACIO_COMPLETA
# define DEPURACIO_BASICA
# define TRACE2(args) fprintf args
#else
# define TRACE2(args)
#endif

#ifdef DEPURACIO_BASICA
# define TRACE1(args) fprintf args
#else
# define TRACE1(args)
#endif

#define NEW(_var,_n,_type) ((_var)=(_type *)malloc(sizeof(_type)*(_n)))
#define Min( a, b)       ( (a)<(b) ? (a) : (b) )
#define Max( a, b)       ( (a)>(b) ? (a) : (b) )
#define SIGNE(a)          (((a) >= 0.0) ? 1.0 : -1.0)
#endif
```

Bibliografia

[1] A. ATRAMENTOV and S. M. LAVALLE. Efficient nearest neighbor searching for motion planning. *IEE Internat. Confer. Robot. Automat*, pages 632–637, 2002.

[2] B. BARRAQUAND, B. LANGLOIS, and J.-C. LATOMBE. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224–241, 1992.

[3] S. BERCHTOLD and B. GLAVINA. A scalable optimizer for automatically generated manipulator motions. In *Proceedings of the IEEE/RSJ/GI International Conference on*, pages 1796–1802, 12-16 Sep. 1994.

[4] J. F. CANNY. *The complexity of robot motion planning*. MIT Press, Cambridge, MA, 1988.

[5] M. A. CHASE. Vector analysis of linkages. *ASME J. Eng. Ind. Communications of the ACM*, 85:289–287, 1963.

[6] J. DENAVIT and R. S. HARTENBERG. A kinematic notation for lower pair mechanisms based on matrices. *ASME J. Appl. Mech.*, 77:215–221, 1955.

[7] J. DUFFY. *Analysis of mechanisms and robot manipulators*. Wiley, New York, 1980.

[8] J. DUFFY and ROONEY. A foundation for a unified theory of analysis of spatial mechanisms. *ASME J. Eng. Ind.*, 97:1159–1164, 1975.

[9] H. EDELSBRUNNER. *Algorithms in combinatorial geometry, monographs on theoretical computer science*. Springer-Verlag, Berlin, 1987.

[10] J. ESAKOV and T. WEISS. *Data structures and advanced approach Using C*. Prentice-Hall International Editions, 1989.

[11] B. FAVERJON. Object-level programming of industrial robots. In *IEEE International Conference on Robotics and Automation*, pages 1406–1412. IEEE Press, 1986.

[12] Introduction GTK+. *http://www.gtk.org/*.

[13] Webot 4. User guide. *http://www.cyberbotics.com/*.

[14] L. HAN and N. AMATO. A kinematics-based probabilistic roadmap method for closed chain systems. In *International Workshop on Algorithmic Foundations of Robotics*, pages 233–246, 2000.

[15] P. E. HART, N. J. NILSSON, and B. RAPHAEL. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, SSC-4(2):100–107, 1968.

[16] L. E. KAVRAKI, M. N. KOLOUNTSAKIS, and J.-C LATOMBE. Analysis of probabilistic roadmaps for path planning. *IEEE Trans. Robot. Automat.*, 14:166–171, 1998.

[17] L. E. KAVRAKI, P. SVETKA, J-C. LATOMBE, and M. H. OVERMARS. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[18] D. E. KODITSCHEK. Exact robot navigation by means of potential functions: Some topological considerations. In *IEEE International Conference on Robotics and Automation*, pages 1–6. IEEE Press, 1987.

[19] J.-C. LATOMBE. *Robot motion planning*. Kluwer Academic Publishers, 1991.

[20] J. P. LAUMOND. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Preprints of the International Conference on Intelligent Autonomous Systems*, pages 346–354. Elsevier Science Publishers B.V., 1986.

[21] S. M. LAVALLE. *Planning Algorithms*. Cambridge University Press, Apareixerà a principis de 2006. Versió electrònica disponible a http://msl.cs.uiuc.edu/ lavalle.

[22] H. Y. LEE and LIANG. Displacement analysis of the general spatial 7-link 7R mechanism. *Mech. Mach. Theory*, 23:219–226, 1988.

[23] H. Y. LEE and LIANG. A new vector theory for the analysis of spatial mechanisms. *Mech. Mach. Theory*, 3:209–217, 1988.

[24] A. LINGAS. The power of non-rectilinear holes. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming, Aarhus*, pages 369–383. LNCS Springer-Verlag, 1982.

[25] T. LOZANO-PÉREZ. A simple motion-planning algorithm for general robot manipulators. *IEEE Transactions on Robotics and Automation*, 3(3):224–238, 1987.

[26] T. LOZANO-PÉREZ and M. A. WESLEY. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22:560–570, 1979.

[27] N. J. NILSSON. *Principles of artificial intelligence*. Morgan Kaufmann, Los Altos, CA, 1980.

[28] P. RICHARD. Kinematic control equations for simple manipulators. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-11:449–460, 1981.

[29] E. RIMON and D. E. KODITSCHEK. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In *IEEE International Conference on Robotics and Automation*, pages 21–26. IEEE Press, 1989.

[30] E. RIMON and D. E. KODITSCHEK. Exact robot navigation in geometrically complicated but topologically simple spaces. In *IEEE International Conference on Robotics and Automation*, pages 1937–1942. IEEE Press, 1990.

[31] J. F. SCHARTZ and C.K. YAP. *Algorithmic and geometric aspects of robotics*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[32] J. T. SCHWARTZ and M. SHARIR. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983b.

[33] F. SCHWARZER, M. SAHA, and J.-C. LATOMBE. Exact collision checking of robot paths. In *In Workshop on Algorithmic Foundations of Robotics*, 2 Dec 2002.

[34] S. SKIENA. *Implementing Discrete Mathematics: Combinatorics and Graph Theory With Mathematica*. Addison-Wesley, July 1,1990.

[35] L.-W TSAI. *Robot analysis*. John Wiley and sons, 1999.

[36] Floppy's VRML97 Tutorial. *http://web3d.vapourtech.com/tutorials/vrml97/*.

[37] E. WELZL. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Information Processing Letters*, 20:167–171, 1985.

[38] J. H. YAKEY, S. LAVALLE, and L. E. KAVRAKI. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958, 2001.