Resum

Aquest projecte té com a objectiu la integració d'un dispositiu hàptic per tal de permetre la seva utilització com a telecomandament d'un braç robòtic i desenvolupar una interfície gràfica que permeti a l'usuari interactuar fàcilment amb els dispositius i la informació que intercanvien.

Un dispositiu hàptic és un aparell capaç de transmetre sensacions tàctils a l'usuari, com ara la presència palpable d'un objecte virtual, generat a l'ordinador. Aquestes sensacions es transmeten en forma de forces i moments a través del seu End Effector, element mòbil que l'usuari agafa amb la mà. Al mateix temps, la posició i orientació de l'End Effector són registrades i enviades com dades posicionals de l'avatar de l'usuari al món virtual, per exemple com la posició i orientació de la seva mà respecte a l'obstacle virtual. És possible utilitzar un dispositiu hàptic com a telecomandament en un entorn de telemanipulació robòtica. En aquest cas, les dades posicionals de l'End Effector es converteixen en consignes de posició per al robot, mentre que les forces que experimenta aquest són transmeses a l'usuari, mitjançant l'End Effector. D'aquesta manera, un dispositiu hàptic permet a l'usuari percebre el robot com una extensió remota del seu braç.

Per fer-ho, a més del dispositiu hàptic són necessaris dos dispositius més: un braç robòtic que segueixi les posicions que el dispositiu hàptic està enviant i un sensor de forces situat a l'extrem del braç que registri les forces i moments als quals està sotmès l'extrem del braç robòtic.

Per altra banda, cal un sistema informàtic que permeti el flux de les dades que els diferents dispositius han d'enviar els uns als altres.

El sistema informàtic emprat per aquesta finalitat, és Robot Operating System (ROS). ROS és un framework per al desenvolupament de software per robots que proveeix la funcionalitat d'un sistema operatiu en un conjunt de sistemes de naturalesa heterogènia.

Actualment, s'està intentant estandarditzar la utilització de ROS al laboratori de l'Institut de Robòtica i Informàtica Industrial de Barcelona, que és on s'ha desenvolupat el projecte. ROS, entre altres aspectes, és un sistema modular, que permet a l'usuari decidir quines parts vol fer servir i quines no, disposa d'una comunitat molt activa d'usuaris que comparteixen informació i consta de llicències molt permissives. . D'aquesta manera, tot i desenvolupar-se a l'IRI, aquest projecte té potencialment una difusió molt més àmplia.

Sumari

1. Glossari

DHD: Delta Haptic Device. És el nom del dispositiu hàptic utilitzat.

FTC: És el nom del sensor de forces i moments situat a l'extrem del braç robòtic.

IRI: Institut de Robòtica i Informàtica Industrial.

ROS: Robot Operating System. És el nom del framework utilitzat per a la comunicació dels diferents dispositius.

SR: Stäubli Robot. És el nom del braç robòtic telecomandat.

2. Prefaci

2.1. Origen del projecte

A l'Institut de Robòtica i Informàtica Industrial de Barcelona (IRI) s'està estandarditzant la utilització de ROS per operar els diferents dispositius que es troben al laboratori.

El dispositiu hàptic que hi ha en aquest centre era un dels pocs sistemes que encara no havien sigut integrats en un entorn de ROS i era necessari realitzar aquesta implementació per tal de poder utilitzar el dispositiu hàptic conjuntament amb altres aparells del laboratori.

Anteriorment, ja hi havia un software ad hoc capaç de comunicar el dispositiu hàptic amb un braç robòtic industrial i un sensor de forces i moments, controlant tots aquests dispositius mitjançant una aplicació gràfica especialment dissenyada per aquesta tasca.

Quan es va començar a utilitzar ROS per treballar amb els diferents dispositius del laboratori, aquest sistema va quedar obsolet i era, per tant, necessari actualitzar el sistema de comunicació entre els diferents mecanismes i l'aplicació emprant ROS amb l'objectiu de poder reproduir els mateixos experiments que es podien realitzar anteriorment, així com realitzar-ne de nous.

2.2. Motivació

La principal motivació és la necessitat de combatre la infrautilització del dispositiu hàptic en el laboratori de l'IRI, permetent que es pugui utilitzar tant per interactuar amb altres aparells com per incloure'l en aplicacions robòtiques de realitat augmentada.

2.3. Requeriments previs

Per poder integrar el dispositiu hàptic en ROS, és necessari tenir coneixements de programació en els llenguatges C i en C++.

També és important entendre quina és la filosofia de treball de ROS, per poder implementar el dispositiu hàptic de manera que treballi amb harmonia amb la resta d'elements del laboratori.

D'altra banda, per tal de poder programar l'aplicació gràfica que permeti a l'usuari interactuar amb els diferents dispositius que formen el sistema (dispositiu hàptic, braç robòtic i sensor de forces), és necessari tenir un cert coneixement de Qt, que és una biblioteca multiplataforma àmpliament utilitzada pel desenvolupament d'aplicacions gràfiques.

L'estudiant autor d'aquest projecte no tenia cap mena de coneixements previs sobre els llenguatges de programació C i C++, sobre ROS o sobre aplicacions gràfiques, fet que ha suposat un esforç important a l'hora de desenvolupar aquest projecte.

Afortunadament, el protocol de nouvinguda del laboratori de l'IRI facilita als estudiants una sèrie de tutorials molt beneficiosos per a l'aprenentatge d'aquests llenguatges i mètodes de treball, a més d'oferir sempre ajuda als estudiants per tal que aquests puguin continuar amb la seva formació.

3. Introducció

3.1. Objectius del projecte

Es poden classificar els diferents objectius que hi ha per a cada dispositiu de la següent manera:

- Dispositiu hàptic

- Creació d'un driver de baix nivell programat en C++.
- Creació d'un node de ROS capaç d'enviar les dades de posició de l'End Effector al braç robòtic i a l'aplicació gràfica i de rebre les dades de forces i moments del sensor de forces i moments.
- Permetre a l'aplicació exercir les forces i moments que l'usuari dessitgi.

- Braç robòtic

- Capacitar el braç robòtic per seguir les consignes de posició enviades pel dispositiu hàptic.
- Permetre a l'usuari moure el braç robòtic tant a l'espai articular com a l'espai cartesià i enviar les dades de la posició de l'extrem del braç (Tool Centre Point) a l'aplicació gràfica.

- Sensor de forces

- Capacitar el sensor de forces per enviar adequadament les forces i moments que està registrant al dispositiu hàptic i a l'aplicació gràfica.
- Permetre a l'usuari que fa servir l'aplicació gràfica designar com a zero la força i moment que el sensor estigui enregistrant en aquell moment.

- Aplicació gràfica

- Mostrar de manera entenedora a l'usuari la interacció entre els diferents dispositius que conformen el sistema.
- Interactuar amb la informació existent que els dispositius estan enviant o, fins i tot, modificar-la.
- Recollir aquestes dades en fitxers per tal de poder realitzar tasques d'aprenentatge basat en demostracions.

Per fer tot això es realitza una planificació temporal del projecte en forma de diagrama de Gant que s'adjunta a l'Annex A 1.1.

També es fa una estimació del cost econòmic del projecte que es pot veure a l'Annex A 1.2 i de l'impacte mediambiental del projecte a l'Annex A 1.3.

3.2. Abast del projecte

El projecte es centrarà en la creació del driver del dispositiu hàptic i del seu node de ROS. No s'aprofundirà significativament en els drivers o nodes del sensor de forces o del braç robòtic, ja que aquests drivers i nodes ja han estat desenvolupats prèviament a aquest projecte.

També revisarà i redisenyarà el funcionament de l'aplicació gràfica i com aquesta interactua amb els altres dispositius.

Quant a l'aplicació, es completarà tot el cicle de telecomandament del robot i la recol·lecció de dades per a la posterior fase d'aprenentatge, però no s'incidirà en aquesta fase directament.

3.3. Estructura de la memòria

El present capítol introductori d'aquesta memòria, com s'acaba de veure, és una explicació dels objectius del projecte en relació a cadascun dels components que integren el sistema.

Seguidament, es fa una descripció dels components per separat, parlant de característiques físiques de l'equipament, els sistemes de referència emprats, i la funció que desenvolupen a la cel·la de telecomandament (Capítol 5). També s'explica com es posen en funcionament, és a dir, com s'alimenten els dispositius i s'enllesteixen per a la seva utilització (Capítol 6).

A continuació s'introdueix un petit "manual" dels aspectes més bàsics de ROS (Capítol 7), que intenta ser una guia per consultar en qualsevol moment que el lector vulgui entendre amb més detall qualsevol aspecte explicat durant el desenvolupament de la memòria.

Arribat aquest punt, es fa una petita anàlisi dels nodes del braç robòtic (Capítol 8) i del sensor de forces (Capítol 9), de manera que s'entengui clarament l'entorn de ROS existent en la situació prèvia a aquest projecte i es pugui comprendre com encaixa el nou node del DHD en aquest entorn. A més, s'analitzen els sistemes de referència de cada dispositiu per tal que la informació enviada entre els dispositius tingui coherència (Capítol 10).

Un cop fet això, s'explica en detall la creació de tot el software relacionat amb el DHD, tant a baix nivell (driver) com a alt nivell (ROS), partint de l'anàlisi de l'apartat anterior (Capítol 11).

Finalment, es fa una explicació del procès de creació de l'aplicació gràfica (Capítol 12) de manera molt conceptual sense entrar massa en detall amb el codi, ja que fer-ho seria massa extens i l'objectiu del projecte és restringir-se el màxim possible en l'àmbit de ROS.

Al final de la memòria s'inclou un apartat de tasques posteriors al projecte que cal implementar per a millorar la interacció entre els diferents dispositius i entre l'usuari i el sistema dissenyat (Capítol 13).

4. Estat de l'art

En el món actual es realitzen moltes operacions que requereixen una precisió molt elevada. Per exemple en l'àmbit de la industria, això inclou tasques com microensamblaments electrònics o diferents aplicacions en ciències de materials. [14]

Hi ha dues alternatives possibles per realitzar aquestes tasques. La primera i més habitual és la implementació de sistemes automàtics que s'encarreguen de realitzar aquestes operacions sense necessitat d'intervenció humana. Aquesta opció pot ser molt interessant, però té l'inconvenient que només és una solució vàlida quan les operacions a realitzar s'executen en entorns altament controlats [15]. També pot arribar a suposar una gran inversió de temps en fases inicials de projectes, quan els protocols d'operació encara no han estat establerts. A més, l'experiència de l'operari encarregat d'establir aquests protocols és molt important, ja que ha de tenir en compte possibles pertorbacions de l'entorn no contemplades prèviament.

És per això que la utilització de màquines tele-operades on l'usuari mou l'eina gràcies a algun tipus de joystick és una solució àmpliament acceptada [16]. No obstant això, únicament operaris amb molta experiència són capaços de realitzar tasques de teleoperació amb èxit. Per solucionar aquest problema, es fan servir els dispositius hàptics per fer l'operació de telecomandament molt més senzilla i intuïtiva, de manera que la tasca de teleoperació pugui ser realitzada per un nombre molt més gran d'usuaris. Això ha permès als dispositius hàptics estendre's en camps com la medicina o el disseny artístic en 3D.

És especialment destacable el camp de la telemedicina, on existeixen dispositius com ara el sistema da Vinci, que és un sistema de cirurgia robòtica creat per l'empresa americana Intuitive Surgical que consisteix en una estació dotada de quatre braços robòtics equipats amb diferents estris quirúrgics i un telecomandament haptic que permet al cirurgià realitzar la tasca de teleoperació de manera més natural (Figura 1. . A més, el sistema s'encarrega de reduir factors com ara la tremolor de les mans o evitar errors humans gràcies a diverses mesures de seguretat en el moviment [18].

Figura 1. Imatge del sistema da Vinci en la qual es pot veure les estacions de telecomandament (esquerra) i d'actuació del robot (part central).

Els telecomandaments hàptics afronten, però, tres grans reptes [14]. El primer és l'estabilitat. Els factors d'escala introduïts per assimilar els moviments del dispositiu hàptic en un entorn on les distàncies amb les quals es treballa són molt més petites, indueix inestabilitat. Un segon repte és la inevitable pèrdua d'informació en la realimentació de forces. La força aplicada sobre l'extrem de la màquina és habitualment deduïda de la deformació de l'eina. Això provoca que moltes vegades, a causa de la geometria de l'eina, aquestes forces quedin reduïdes a una o dues dimensions únicament, proporcionant així informació limitada sobre les forces o moments reals. Per últim, els retards o delays entre el dispositiu hàptic i la màquina poden fer que l'experiència de teleoperació no sigui gens intuïtiva, especialment quan la distància entre el telecomandament i la màquina és gran.

Avui dia, s'intenta combatre els efectes de la inestabilitat i el retard temporal mitjançant millores al llaç de control del controlador del dispositiu hàptic [14]. Per disminuir l'efecte de la pèrdua d'informació sobre les forces, la visió és un recurs molt interessant [17]: mitjançant sensors òptics es calculen les deformacions de l'eina, estimant la força resultant gràcies a les propietats del material d'aquesta eina.

Aquest projecte enfrontarà principalment el repte de la comunicació entre els diferents dispositius, considerant-se satisfactori un resultat en què el braç robòtic segueixi les consignes de posició del DHD amb una fidelitat raonable i que l'usuari sigui capaç de sentir les forces i moments enviats pel sensor de forces gràcies al sentit del tacte.

El principal objectiu d'aquest projecte no és superar cap d'aquests grans reptes, sinó incorporar el DHD del laboratori de l'IRI, que en l'actualitat es troba en un estadi avançat de la seva vida útil, en un entorn de ROS per tal d'allargar aquest cicle de vida i de possibilitar les operacions de telecomandament amb una qualitat raonable.

A més, ROS presenta una relativa facilitat per canviar qualsevol dels elements que conformen un sistema per un altre, facultat que dóna molta flexibilitat al DHD per treballar amb altres tipus de braços robòtics o fins i tot altres aparells com drones.

5. Components del sistema

Com ja s'ha vist prèviament a l'informe, aquest projecte considera diferents dispositius que interactuaran entre ells. Els dispositius que formen el sistema són els següents:

- Ordinador amb OS Ubuntu 14.04 LTS
- Robot manipulador de 6 GDL – ***Stäubli RX60***
- Interfase hàptica - ***DHD de 6 GDL***
- Sensor de força/moment – ***FTC-50 de Schunk***
- Aplicació gràfica prèviament existent

5.1. Ordinador

L'ordinador amb el qual s'han creat i executat tots els fitxers del projecte té instal·lada la versió 14.04 d'Ubuntu com a sistema operatiu. També té instal·lat el compilador GNU Compiler Collection (GCC) per poder compilar els arxius de C++ que formen el projecte.

L'ordinador té instal·lat el sistema operatiu ROS, la versió Indigo. Qualsevol versió posterior a ROS Hydro també és suficient.

La seva funció principal en el sistema que s'estudia en aquest projecte és connectar i processar tota la informació que comparteixen els altres dispositius mitjançant ROS.

5.2. Stäubli RX60

A la Figura 2 es pot veure el dispositiu Stäubli RX60, un robot manipulador sèrie industrial amb 6 graus de llibertat.

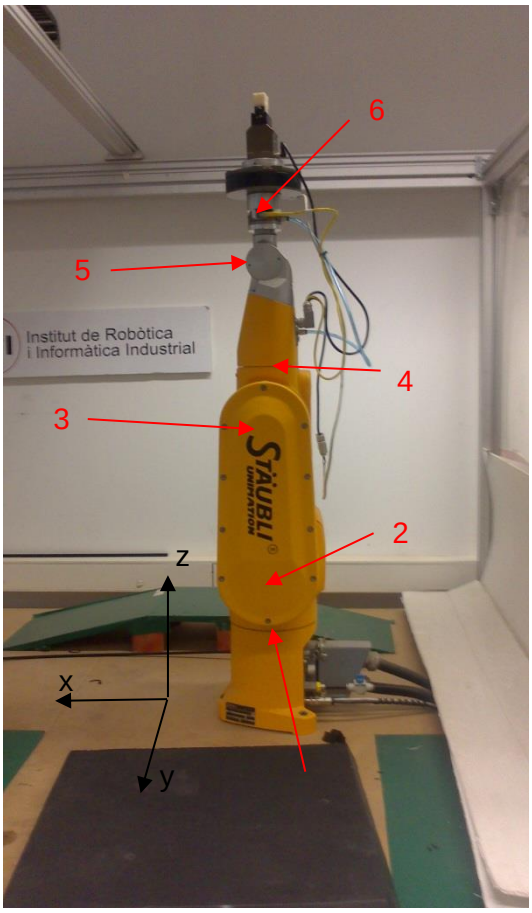


Figura 2. Fotografia del braç robòtic Stäubli on es poden veure les articulacions numerades i els eixos de coordenades del dispositiu.

Fabricat per l'empresa Stäubli a Suïssa [1], consta de 6 articulacions independents amb un motor pas a pas en cada una d'elles i es pot moure tant a l'espai articular com a l'espai cartesià. Està situat en una cel·la de treball protegida amb un sensor làser que com a mesura de seguretat atura el robot quan qualsevol objecte passa a través del sensor.

Aquest robot manipulador pot ser equipat amb diferents eines a l'extrem del braç. Concretament, en aquest projecte, el robot Stäubli està equipat amb prensor (pinces en aquest cas). Entre el canell del braç i les pinces es situa el sensor de forces i moments FTC-50.

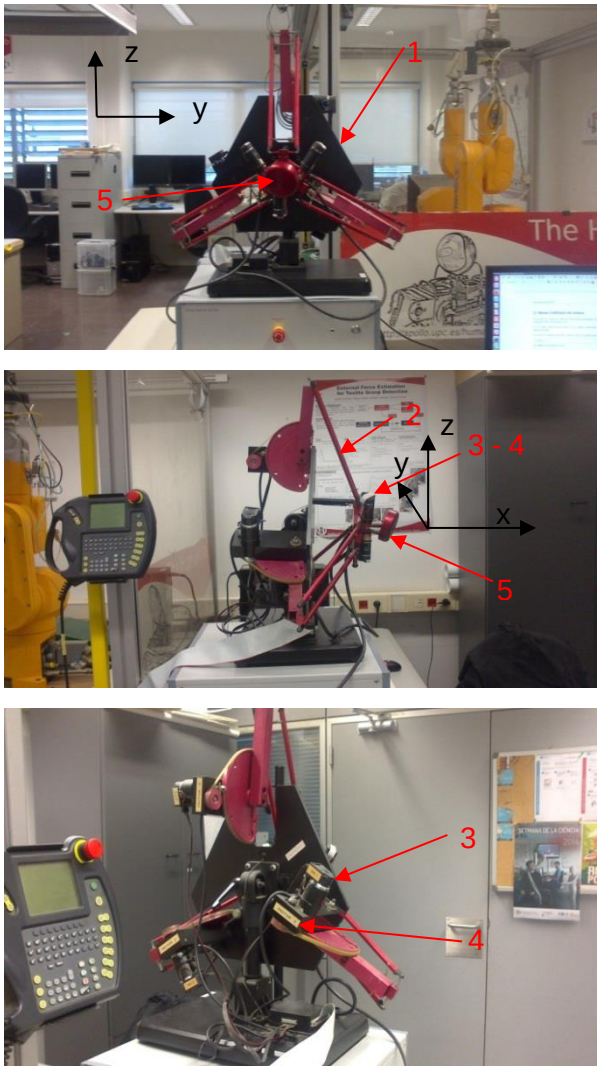
La seva funció principal en el sistema d'estudi és executar les consignes de posició del dispositiu hàptic.

5.3. Delta Haptic Device (DHD)

El dispositiu hàptic amb el qual s'ha treballat és un model Delta 6 Haptic Device, fabricat per l'empresa suïssa Force Dimension [2] . La seva funció és telecomandar en posició i orientació el braç robòtic i transmetre a l'usuari les forces que llegeixi el sensor de forces.

5.3.1. Elements físics

Segons la Figura 3, els diferents elements físics del dispositiu hàptic són:



1. **Plataforma**

Base sobre la qual està recolzada tota l'estructura del dispositiu hàptic. Els braços paral·lels están units a la mateixa mitjançant unions reticulars. També els encoders i els motors estan recolzats directament en ella.

2. Braços paral·lels

Hi ha tres braços paral·lels que van des de la plataforma fins a la base de l'End Effector. Cadascun d'aquests braços consta de dos trams units entre si amb ròtules esfèriques. El tram connectat amb la plataforma amb una unió reticular consta d'un engranatge que està connectat amb els motors mitjançant una corretja, que permet transmetre la força dels mateixos a l'End Effector.

Figura 3. Mapa del DHD on es mostren els seus components, amb la numeració corresponent al text principal, i el sistema de referència associat.

El segon tram consta de dues barres paral·leles unides entre si amb dues molles, una a cada extrem. Aquest segon tram de cada braç arriba fins a l'End Effector, al qual hi està unit també mitjançant ròtules esfèriques.

Entre cadascun dels tres braços hi ha una separació de 120º.

3. **Encoders**

A cada disc de l'engranatge del primer tram de cada braç hi ha un encoder que s'encarrega de transformar l'angle girat pel disc en un senyal digital de la posició de l'End Effector.

De la mateixa manera, hi ha 3 encoders als motors de la plataforma de l'End Effector que s'encarreguen de registrar l'orientació d'aquest.

4. **Motors**

Hi ha dos tipus de motor en el dispositiu. Per una banda, els motors que estan recolzats a la plataforma, un per cada braç, que s'encarreguen de transmetre forces a l'End Effector mitjançant una corretja, com es veu a la Figura 4.

Per l'altra banda, tres motors addicionals situats a la base de l'end effector que estiren o arronsen un fil metàl·lic cadascun connectat a l'End Effector. Aquests conjunts motor-fil permeten reproduir moments a l'End Effector.

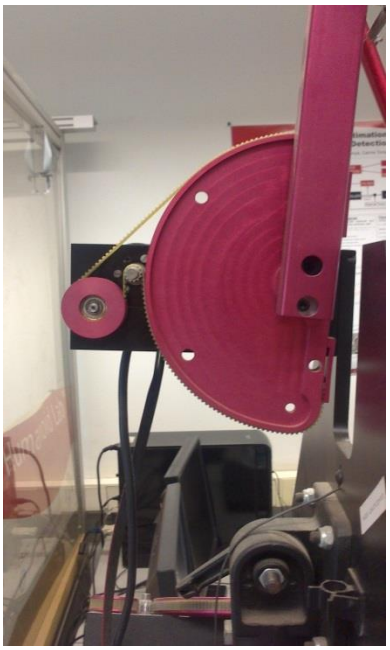


Figura 4. Detall del mecanisme de transmissió de forces.

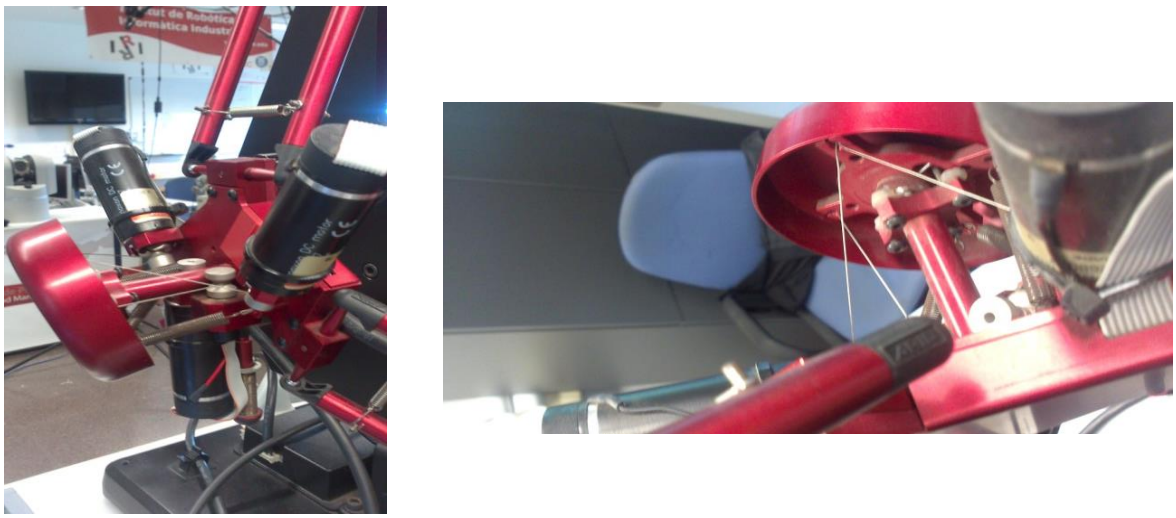


Figura 5. Detall del sistema de transmissió de moments a l'End Effector.

5. End Effector

L'End Effector és la part del dispositiu que l'usuari ha d'agafar amb la mà. Consisteix en una peça metàl·lica amb forma de tap amb un diàmetre de 8,15 cm. Està unit a una base mitjançant una ròtula esfèrica al punt mitjà de la peça. També està connectat a aquesta base amb tres molles i als motors de la base de l'End Effector amb els tres fils metàl·lics esmentats.

5.4. FTC-50

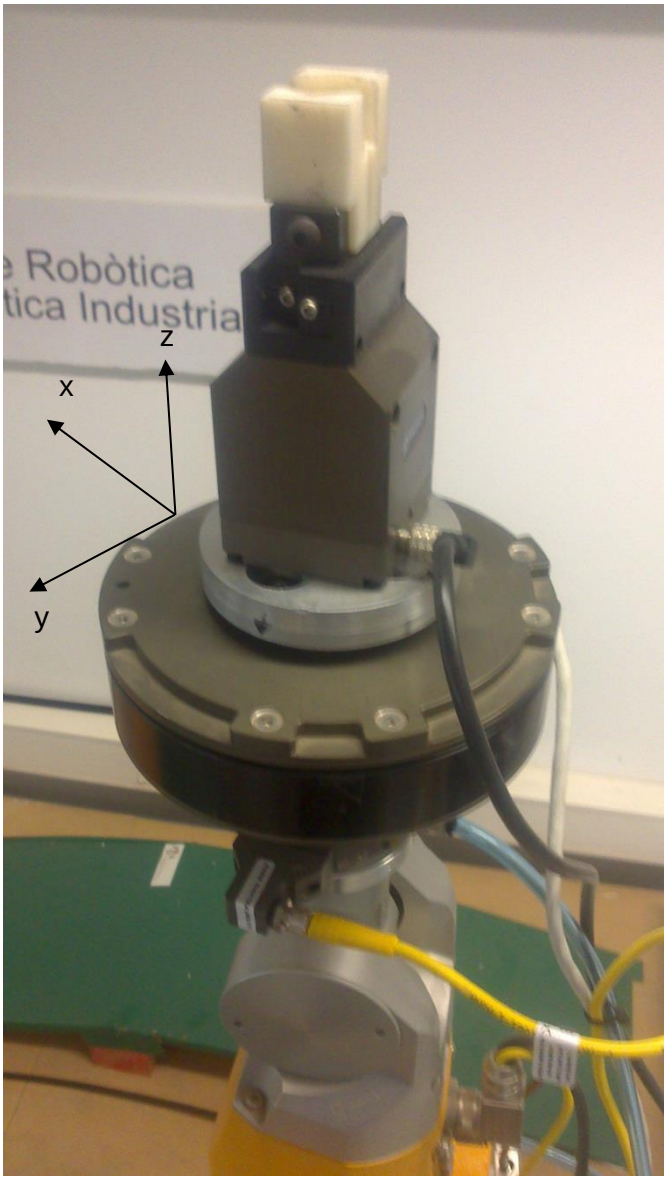


Figura 6. Figura on es pot veure el sensor de forces i el seu sistema de referència.

Fabricat per la empresa alemanya Schunk [3], el dispositiu FTC-50 de la Figura 6 és un sensor de forces i moments que està situat a l'extrem del robot manipulador Stäubli. És capaç de mesurar les forces i moments que experimenta el gripper del sensor en tot moment.

La seva funció principal dintre del sistema a estudiar és llegir correctament les forces i moments que estigui captant durant l'operació per tal que el dispositiu hàptic pugui aplicar-les a l'End Effector.

Els seu sistema de referència és equivalent al sistema de referència de la base del braç robòtic girat -45º entorn l'eix z, eix que és coincident en tots dos dispositius.

5.5. Aplicació gràfica prèviament existent

Abans de realitzar aquest projecte, ja existia una aplicació gràfica capaç de mostrar a l'usuari les dades que els tres dispositius comparteixen [4].

L'aspecte d'aquesta aplicació gràfica (Figura 7) és el següent:

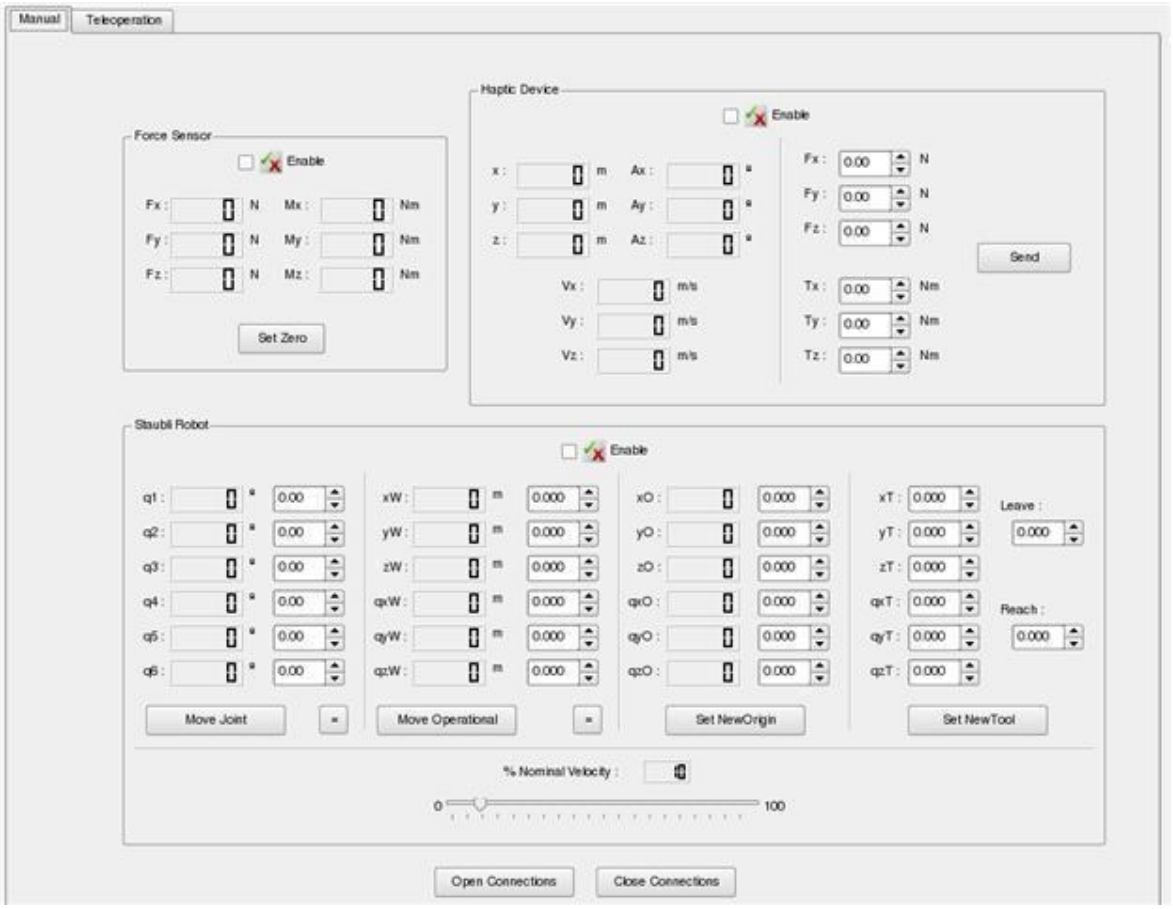


Figura 7. Aspecte de la pestanya Manual de l'aplicació gràfica prèvia a aquest projecte.

L'aplicació consta de dues pestanyes, Manual i Teleoperation. A la pestanya Manual es poden diferenciar tres zones diferents, cadascuna per un dels dispositius.

A la zona del Force Sensor es pot veure una casella per activar o desactivar la comunicació entre l'ordinador i el sensor. Alhora, es poden veure les dades de forces i moments que detecta el sensor de forces, a més d'oferir un botó anomenat "Set Zero" amb el qual l'usuari pot fer que les forces que s'estiguin registrant en aquell moment siguin el nou origen de referència.

A la zona del Haptic Device es poden veure les dades que fan referència al dispositiu hàptic. Hi ha una casella que activa o desactiva la connexió entre el dispositiu hàptic i l'ordinador, una zona on es mostren les dades de posició, orientació i velocitat de l'End Effector i un altra zona on es poden seleccionar unes determinades forces i moments per aplicar a l'End Effector del dispositiu hàptic transferint-les amb les fletxes dels requadres i el botó "Send".

Per últim, a la zona "Stäubli Robot" a més d'una casella per activar o desactivar la comunicació del braç robòtic amb l'ordinador hi ha:

- Una primera columna que mostra la rotació en graus de cadascuna de les articulacions del braç robòtic. També hi ha un botó anomenat "Move Joint" amb el qual es mou el braç robòtic en l'espai articular fins que cada articulació prengui el valor especificat als camps de la dreta d'aquesta columna. La velocitat del moviment es pot fixar amb el control lliscant de la part inferior. També hi ha un botó amb un signe igual, que serveix transferir als camps de la dreta directament els valors actuals de cadascuna de les articulacions.

- Una segona columna on es mostra el valor de la posició i orientació de l'extrem del braç robòtic a l'espai cartesià. També hi ha un botó anomenat "Move Operational" que permet moure l'extrem del braç robòtic a la posició especificada amb l'orientació especificada als camps de la dreta d'aquesta columna. La velocitat del moviment es pot fixar amb el control lliscant de la part inferior.

També hi ha un botó amb un signe igual, que serveix per transferir als camps de la dreta directament els valors actuals de les posicions i orientacions de l'extrem del braç robòtic

en el moment de prémer el botó.

- A la tercera columna es pot fixar una nova posició i orientació d'origen, fixant els valors desitjats als camps de la dreta i prement el botó Set NewOrigin.

- A la quarta columna es pot fer fixar una nova referència de l'extrem del braç (anomenat Tool Center Point o TCP) en cas que es vulgui operar el braç robòtic amb alguna eina a l'extrem d'aquest. D'aquesta manera, les posicions i orientacions que l'usuari rebi o enviï estaran referenciades a l'extrem d'aquesta eina. Això s'aconsegueix especificant les coordenades de posició i l'orientació del nou punt de treball als camps corresponents i prement el botó "Set NewTool".

També hi ha en aquesta columna dues caselles amb els noms "Leave" i "Reach", que són dos paràmetres encarregats d'especificar com serà la trajectòria que seguirà el braç quan segueixi una sèrie de punts a l'espai.

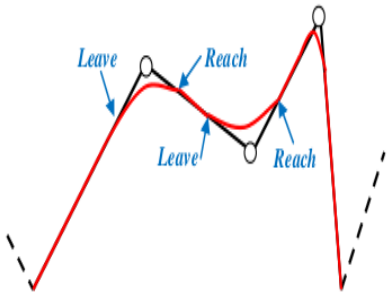


Figura 8. Esquema conceptual dels paràmetres Leave i Reach.

Com es veu a la Figura 8, el paràmetre Reach determina quan s'entra i el paràmetre Leave quan s'abandona la línia recta que uneix dos punts consecutius d'una trajectòria.

A la pestanya Teleoperation de la Figura 9 s'entra al mode de telecomunicació pròpiament dit, on els diferents dispositius treballen en conjunt.

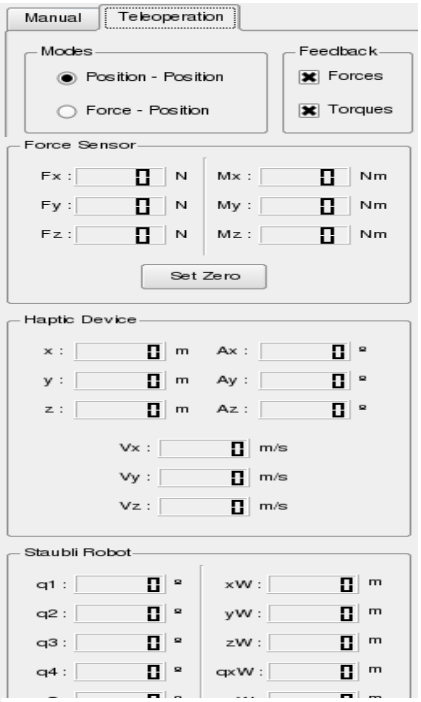


Figura 9. Aspecte de la pestanya Teleoperation de l'aplicació gràfica prèvia a aquest projecte.

A dalt de tot, hi ha dos menús. Al menú Modes es pot escollir entre el mode "Position-Position" (on el dispositiu hàptic envia les seves posicions al braç robòtic però no rep les forces i moments del sensor de forces) o el mode "Force-Position" (on sí que hi ha realimentació de forces i moments).

Al menú feedback, es permet seleccionar quines realimentacions es volen transmetre a l'end effector del dispositiu hàptic, forces, moments, tots dos o cap d'ells.

Per altra banda, hi ha tres altres menús, un per cadascun dels dispositius, on es mostren les forces i moments del sensor, les posicions, orientacions i velocitats de l'End Effector del dispositiu hàptic i la posició del TCP del braç robòtic a l'espai articular i cartesià.

Com s'ha vist a l'apartat 3.1, l'objectiu de la nova aplicació és reproduir gran part de les funcionalitats de l'aplicació antiga. Tot i això, s'ha considerat que hi ha aspectes que es podien corregir:

- En la nova aplicació no hi ha un botó per seleccionar individualment quins dispositius s'estan fent servir, ja que per fer-ho només caldria parar els nodes dels dispositius que no es volen fer servir, tot i que aquesta funcionalitat es podria afegir en projectes futurs.

- La presència d'un menú pel sensor de forces a la pestanya manual no és estrictament necessària, ja que l'usuari no pot interactuar significativament amb el sensor tret del botó Set Zero, que ja hi és a la pestanya Teleoperation.

- Els botons Leave i Reach s'han suprimit igualment, ja que el node de ROS del braç robòtic s'encarrega internament de modificar aquests paràmetres i es preferia apostar per una aplicació senzilla, prescindint d'aspectes que no es fan servir habitualment.

- De la pestanya Teleoperation s'ha suprimit el control lliscant de la velocitat, ja que normalment l'usuari vol que l'End Effector del dispositiu hàptic i el TCP del braç robòtic es moguin de manera simultània i el fet que no sigui així pot arribar a ser anti-intuïtiu.

- També s'han corregit alguns errors en les unitats de mesura d'alguns camps de l'aplicació.

6. Posada en marxa

6.1. Posada en marxa del Stäubli RX-60

Per la correcta posada en marxa del robot manipulador Stäubli RX-60 [4], el primer pas és entrar a la cel·la robòtica i fixar el sensor de forces i moments a l'extrem del braç robòtic. Per fer-ho, és necessari moure cap amunt el sensor i després accionar la vàlvula de color blau d'aire comprimit que surt de la base del Stäubli (Figura 10). Aquest mecanisme fa la funció de fusell mecànic, de manera que si durant l'operació, el sensor de forces i moments rep algun cop suficientment fort, l'aire surti de la cavitat, el sensor de forces quedi en una posició amb una certa llibertat de moviment i a més s'impedeix continuar amb qualsevol operació del robot a causa de l'electrònica del sistema.

Figura 10. Figura que mostra la válvula blava que cal accionar per fixar el sensor de forces a l'extrem del braç robòtic.

El següent pas a fer, és accionar el controlador del robot (Figura 11). Això s'aconsegueix introduint la clau accionadora a la ranura situada a la part superior dreta de la cara frontal del controlador. La clau te una fletxa gravada que ha d'apuntar a l'opció "OFF" en el moment d'introduir-la. Seguidament, es gira la clau 90º en sentit horari fins a que

la fletxa apunti a l'opció “ON”.

També és necessari energitzar el robot fent servir el panell de control situat a la part inferior esquerra de la cel·la (Figura 12). Per fer-ho, s'ha d'accionar l'interruptor superior del panell.

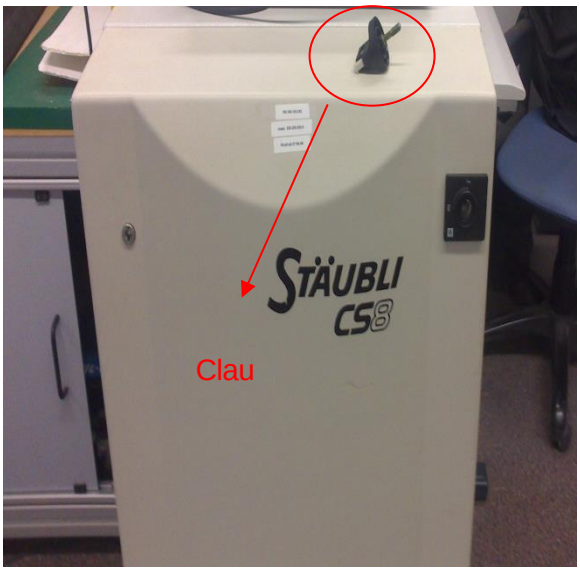


Figura 11. Controlador del braç robòtic Stäubli.

Figura 12. Panell d'energia de la cel·la robòtica.

Seguidament, s'ha de manipular el selector de mode de treball de la cel·la de la Figura 13. Primer de tot, s'ha de comprovar que l'indicador lluminós del Stäubli està encès.

Figura 13. Selector de mode de treball de la cel·la.

Un cop fet això, es pot procedir a l'etapa de calibratge. És necessari comprovar en primer lloc que la posició inicial del robot sigui la correcta. Per fer-ho, és suficient verificar que les marques a cadascuna de les articulacions del manipulador siguin coincidents, com es mostra a la Figura 14.

Figura 14. Marca de calibratge del braç Stäubli.

Segons quin sigui el resultat d'aquest pas de verificació, s'haurà de realitzar un d'aquests passos (1 o 2):

1. En cas de no ser coincidents, és necessari reubicar les articulacions del robot fins que totes coincideixin. Per fer-ho, s'haurà d'utilitzar el "teach pendant", mostrat a la Figura 15. S'haurà de seleccionar el mode de posicionament manual del robot. Això s'aconsegueix pressionant el botó de selecció de mode fins a arribar al mode manual (l'icona que s'il·lumina representa una mà). Després, s'ha de pressionar el dead man button, una barra que es troba a la part posterior del pendant. Aquest botó té tres posicions. La primera, és la posició del botó quan no està premut, la segona és quan el botó està lleugerament premut i l'última és quan el botó està totalment premut. La primera i la tercera no permeten accionar el robot per motius de seguretat. Per tant, per poder accionar el robot, s'haurà de prèmer el botó de la part posterior sense arribar a pressionar-ho completament.

Figura 15. Teach Pendant que permet maniobrar el robot.

Seguidament, s'ha de prèmer el botó Power, que s'il·luminarà per informar que el robot ja es pot moure.

En aquest punt, és necessari seleccionar l'espai on es vol operar el braç (articular o cartesià) amb el selector de mode. Per a realitzar una calibratge, generalment serà millor treballar a l'espai articular.

Per últim, amb els botons de posició, s'ha de moure cada articulació fins que totes les marques físiques coincideixin, reduint la velocitat si és necessari per a una millor precisió.

Finalment, seleccionant l'opció "Rec.", el robot es calibrarà.

2. En cas de ser coincidents, no cal realitzar cap acció addicional.

Finalment, s'ha de prémer el botó REARM del selector de mode de treball de la cel·la (Figura 13) per accionar la barrera làser de protecció i seleccionar el mode Xarxa del Teach Pendant.

Un cop fet això, el braç robòtic ja estarà llest per la seva utilització.

Per realitzar el seu tancament, només cal moure el braç robòtic a la seva posició inicial ajudant-se del Teach Pendant, mantenint premut el botó "Move", que retorna el braç robòtic a la seva posició original i, un cop fet això, tornar a accionar tots els interruptors fins a la seva posició original.

6.2. Posada en marxa de la interface hàptica DHD

El primer pas per encendre el dispositiu hàptic [4] és accionar l'interruptor que hi ha a la part superior dreta de la cara posterior del seu controlador (Figura 17).

Un cop fet això, l'End Effector realitzarà uns moviments de calibratge de manera automàtica. Durant aquesta operació, l'indicador "MOTORS ON/OFF" de la cara frontal del controlador s'il·luminarà de color vermell. Sempre que està encès, aquest indicador informa que els motors estan actuant. El mateix indicador és també un botó, que permet parar els motors o encendre'ls. A més, un botó vermell d'emergència detè completament tots els motors del dispositiu, com es pot veure a la Figura 16.

També és possible que el led de l'indicador "STATUS" de la cara frontal, un cop realitzada la calibratge de l'End Effector, comenci a parpellejar. Això és degut al fet que el dispositiu necessita també un calibratge en la posició, i per solucionar-ho s'ha d'agafar l'End Effector i moure'l fins a la posició de mínim recorregut de cadascun dels tres braços paral·lels. Un cop fet això, s'ha de retornar l'End Effector a la posició d'aparcat (descansant al recolzament de la base). En acabar, el led verd hauria de mantenir-se encès de manera constant i estar preparat per a ser utilitzat.

Figura 16. Cara frontal del controlador. Es pot veure el botó vermell d'emergència, el led verd que indica que el dispositiu està llest per ser utilitzat i a la dreta d'aquest hi ha el botó que indica si els motors estan activats amb una llum vermella.

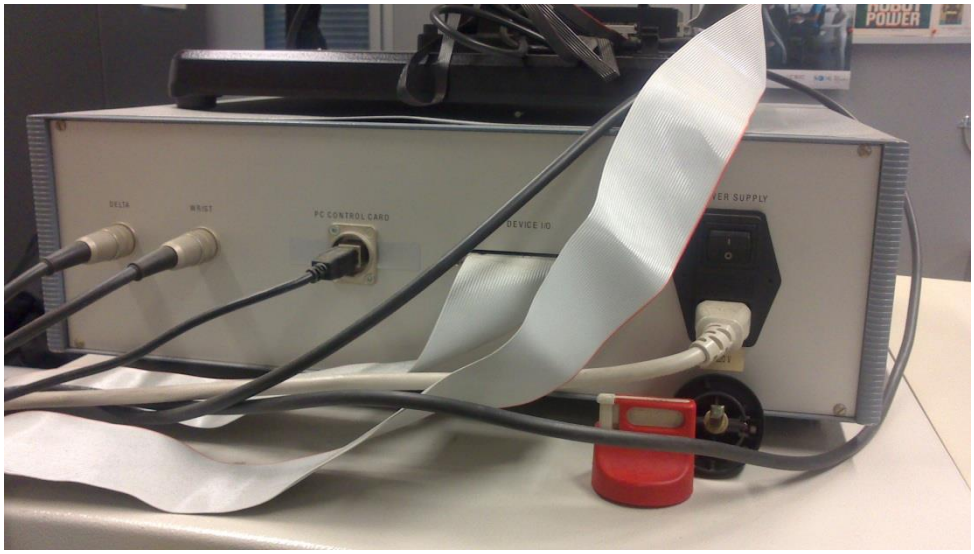


Figura 17. Part posterior del controlador del DHD on hi ha l'interruptor d'alimentació.

Per apagar el DHD, només cal encaixar la plataforma de l'End Effector al suport de la base i tornar a accionar l'interruptor de l'alimentació.

6.3. Posada en marxa del sensor de F/T FTC

Per poder utilitzar el sensor de forces [4], cal accionar l'interruptor corresponent del panell de control (Figura 18). Un led de color verd ha d'il·luminar-se en el sensor de forces per indicar que està operatiu. Per apagar el dispositiu, s'ha de girar l'interruptor fins a la posició original.

Figura 18. Interruptor d'alimentació del sensor de forces.

7. Elements bàsics de ROS

El sistema emprat per compartir la informació entre els diferents dispositius és ROS (Robot Operating System). Aquest sistema presenta unes característiques pròpies que cal considerar a l'hora de fer-ho servir [5].

7.1. Packages

El primer pas que s'ha de fer per començar a treballar amb ROS un cop instal·lat és crear un workspace.

Per fer-ho, es pot crear un directori des de el terminal, introduint per exemple:

```
$ mkdir -p ~/iri-lab/iri_ws/src
```

Un cop creat el directori, s'ha d'anar a aquest directori i introduir la següent comanda:

```
$ catkin_init_workspace
```

Això genera un arxiu CmakeList.txt, que permetrà compilar dins del workspace. Després, s'ha de retornar al directori immediatament inferior i introduir la següent comanda:

```
$ catkin_make
```

Si ara s'observa el directori actual, s'hauran creat dues carpetes més, build i devel, a més de la carpeta src que ja s'havia creat. És precisament dintre de la carpeta 'src' on es troben els packages.

Un package és la unitat atòmica més petita de ROS. Serveixen per organitzar el software que s'utilitza. Dintre d'un package poden haver-hi nodes, llibreries, … L'objectiu d'aquests packages és proveir aquesta funcionalitat d'una manera fàcil de consumir. Els packages es poden construir de manera fàcil amb eines com 'catkin_create_pkg', però en aquest projecte els packages que s'han creat, ho han estat seguint scripts proporcionats per l'IRI.

En definitiva, un package no és res més que una carpeta o directori dintre d'un workspace de ROS que conté software rellevant. També és important notar que aquest package ha de tenir obligatòriament un arxiu anomenat 'package.xml', que és un manifest que serveix per definir propietats sobre el package com el nom del package, la versió, l'autor i les dependències amb altres packages.

7.1.1. Packages utilitzats en el projecte

En aquest projecte s'han utilitzat els següents packages:

- iri_ftc_force_sensor: package que conté els arxius relatius al node del sensor de forces i moments (aquest package es troba dins del meta package iri_common_drivers)
- iri_staubli: package que conté els arxius relatius al robot Stäubli.
- iri_core: package que conté diferents arxius proporcionats per l'IRI que serveixen per facilitar la creació de projectes amb els scripts proporcionats per l'IRI.

7.1.2. Packages creats en el projecte

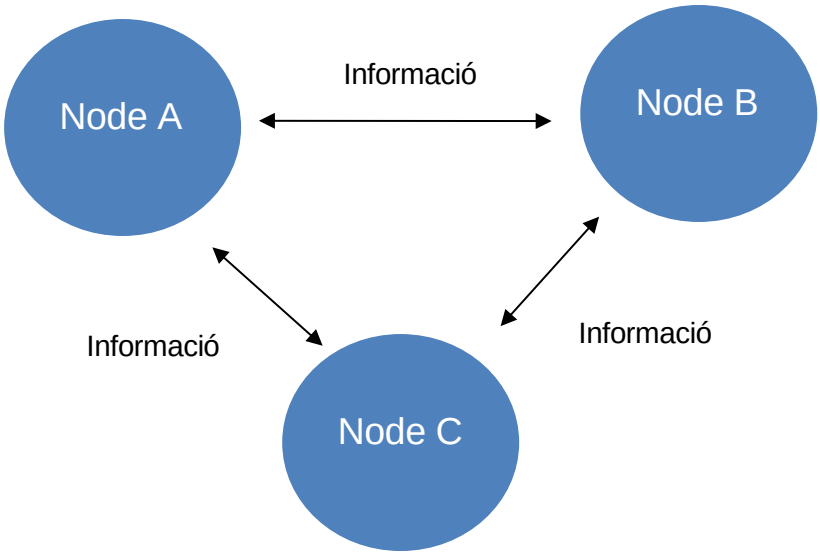
En aquest projecte, s'han desenvolupat els següents packages:

- iri_delta_haptic_device: package que conté els arxius relatius al node del dispositiu hàptic.
- iri_gui_delta_haptic_device: package que conté tots els arxius relatius a la aplicació gràfica per monitoritzar el sistema.

7.2. Nodes

Un node no és res més que un arxiu executable en un package de ROS. En un sistema ROS, els diferents nodes que existeixen tenen la capacitat de comunicar-se entre ells.

La interacció entre aquests nodes es pot representar mitjançant un diagrama de grafs com a l'Esquema 1 i, per comunicar-se, utilitzen topics i serveis, entre d'altres. En una situació real, un sistema podria presentar un node per un detector de distàncies làser, un altre node per les rodes motoritzades, un node encarregat de localitzar la posició del robot i un node que s'encarregués de realitzar planificació de moviment.



Esquema 1. Arquitectura de nodes de ROS.

Aquesta arquitectura presenta avantatges interessants, com ara el fet que, en cas d'haver-hi un error en un dels nodes, aquest error queda aïllat només en aquest node. La complexitat del codi es veu reduïda en comparació amb altres sistemes monolítics, on aspectes funcionals clarament diferenciables en el plantejament del problema (com ara els inputs, outputs, el processament de la informació, el processament dels errors, ...) no estan separats en l'arquitectura del sistema. A més, no és necessari exposar els detalls d'implementació característics de cada dispositiu, fet que fa que es pugui treballar amb diferents nodes sense la necessitat de conèixer en profunditat els aspectes tècnics de tots els dispositius, que fins i tot poden arribar a estar implementats en diferents llenguatges de programació.

Tots els nodes que s'estan executant tenen un nom únic per tal de poder identificar-los correctament. També tenen un tipus (node type) que simplifica el procés de referenciació

dels nodes. Aquest tipus no és altre més que el nom del package on es troba el node. D'aquesta manera, ROS busca tots els executables del package amb aquest nom i selecciona el node segons el nom. És, per tant, important no dissenyar nodes amb el mateix nom dintre del mateix package.

Un node de ROS s'escriu utilitzant una llibreria client de ROS, com per exemple roscpp (per nodes escrits en C++) o rospy (per nodes escrits en Python).

7.2.1. **Roscore**

Un node especialment important és el node Master. Sempre que volem començar a treballar amb ROS, primer hem d'escriure la següent línia de comandes al terminal:

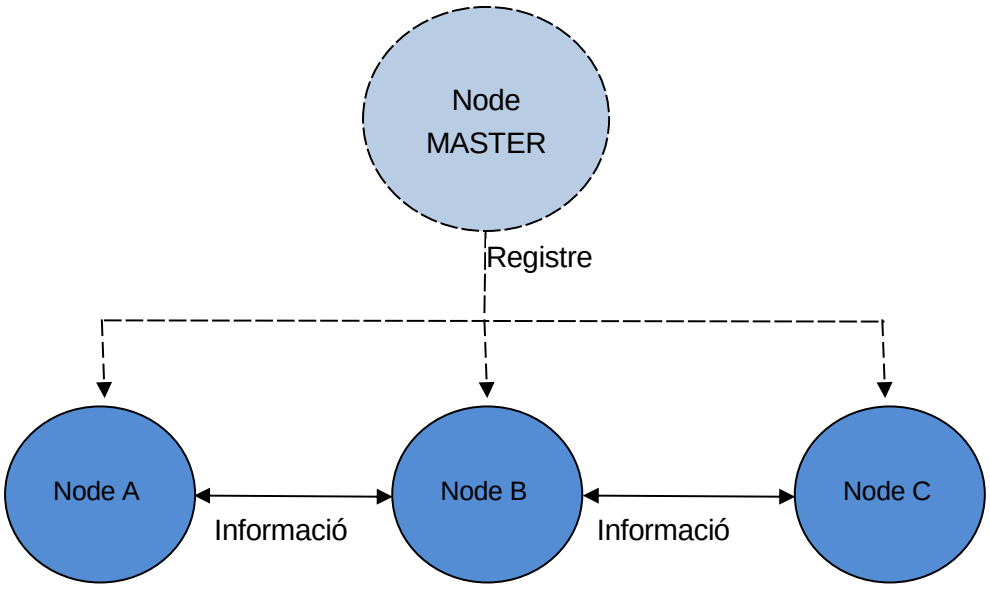
```
$ roscore
```

Aquesta comanda inicia directament un node Master de ROS, un node anomenat "rosout" i un servidor de paràmetres.

El node Master és l'encarregat d'anomenar i registrar la resta de nodes del sistema ROS. També s'encarrega de recordar quins nodes envien i quins nodes reben una determinada informació. En poques paraules, un node Master és l'encarregat d'ajudar a la resta de nodes a comunicar-se entre ells, com es mostra a l' Esquema 2.

El node "rosout" és el nom de la consola registradora de connexions de ROS, encarregat de recol·lectar la informació que els nodes envien entre si.

Per últim, el servidor de paràmetres s'encarrega de guardar diferents valors que donen informació sobre la configuració de ROS.



Esquema 2. Diagrama que mostra la funcionalitat del node Master.

7.2.2. rosnode i rosrun

Dues comandes molt interessants per interactuar amb els nodes de ROS son rosnode i rosrun.

La comanda rosnode ens ofereix informació sobre un node o grup de nodes. Així, s'introdueix la següent comanda:

```
$ rosnode list
```

s'obté una llista amb tots el nodes que actualment s'estan executant.

Si s'introdueix la següent comanda:

```
$ rosnode info /node_A
```

s'obté informació concreta sobre el node anomenat /node_A, com per exemple quina informació envia, quina informació rep o de quins serveis disposa.

La comanda rosrun permet executar un node en particular. La manera d'utilitzar aquesta comanda és la següent:

```
$ rosrun package_A node_A
```

D'aquesta manera, executem el node anomenat "node_A" que es troba dins del package "package_A".

7.2.3. Nodes utilitzats al projecte

Els nodes utilitzats en aquest projecte són els següents:

- /ftc_force_sensor_driver_node: node que s'encarrega d'executar totes les funcionalitats requerides del sensor de forces i moments.
- /iri_staubli_controller: node que s'encarrega d'executar totes les funcionalitats requerides del braç robòtic Stäubli.

7.2.4. Nodes creats al projecte

- /delta_haptic_device_driver_node: node que s'encarrega d'executar totes les funcionalitats requerides del dispositiu hàptic.

- Per últim, hi ha el node de l'aplicació gràfica, amb un nom que a causa del funcionament de rqt varia amb cada execució, però que sempre té l'estructura "rqt_gui_cpp_node_x", on el que varia cada cop és un número que substitueix a la "x".

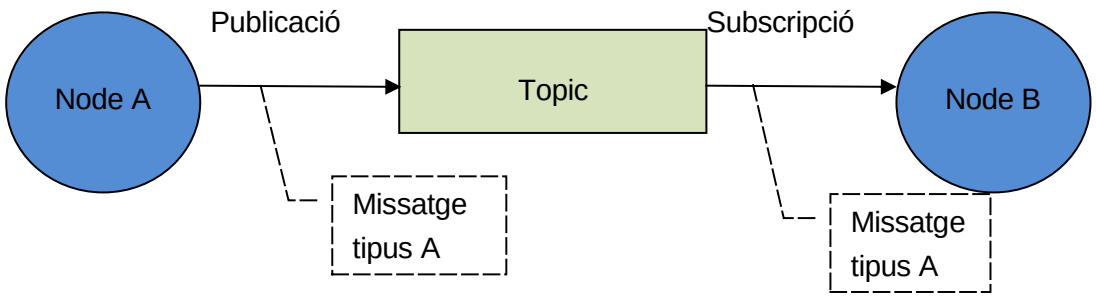
7.3. Topics

Un cop vist què són els nodes, cal estudiar quins mecanismes permeten que aquests es comuniquin entre ells. Un dels mecanismes per dur a terme aquesta comunicació és el que s'anomenen topics, que es pot veure a l'Esquema 3.

Com a metàfora que permeti entendre millor què és un topic, es podria dir que un topic és un tema de conversació entre dos nodes. Cada node pot publicar o estar subscrit a un topic en particular. Quan el node publica informació en aquest topic, el node en qüestió correspondria a la persona que està parlant sobre aquest tema de conversació. Quan el node està subscrit en aquest topic, escolta el que un altre node està dient sobre aquest tema de conversació.

És important, però, que tant els nodes que publiquen com els nodes subscrits, tinguin en compte el tipus del topic, igual que dues persones que parlen acostumen a fer-ho en el mateix llenguatge, per continuar amb la metàfora.

També és interessant el fet que aquest sistema per compartir la informació és anònim: un node que publica informació a un topic no necessita saber qui està rebent aquesta informació i un node que està rebent informació d'un topic no necessita saber qui l'està enviant. Aquest sistema permet desacoblar la producció d'informació del seu consum.



Esquema 3. Diagrama del funcionament del sistema de comunicació amb topics.

Per tant, direm que els topics són una eina de comunicació unidireccional, ja que quan un node publica informació en un topic no espera mai cap tipus de resposta, de la mateixa manera que un node subscrit a un topic no necessita un avís per començar a rebre la informació.

Òbviament, diferents nodes poden publicar informació en un topic de la mateixa manera que un grup de nodes pot estar subscrit al mateix topic.

7.3.1. rostopic

L'eina de ROS per obtenir informació sobre els topics és rostopic. Presenta diverses funcions interessants, però les més bàsiques són les següents:

```
$ rostopic list
```

De la mateixa manera que amb els nodes, amb aquesta comanda podem obtenir una llista de tots els topics que estiguin actius. Un topic estarà actiu quan un node hi publiqui informació o un node hi estigui subscrit.

```
$ rostopic echo /nom_del_topic
```

Aquesta comanda ens permet observar per la pantalla del terminal quina informació s'està publicant en un topic concret.

7.3.2. Tipus de topics utilitzats

Els tipus de topics utilitzats en el projecte són els següents:

- geometry_msgs/Pose: Un missatge de tipus Pose s'utilitza per representar una localització a l'espai i consta d'una posició i una orientació. Per la part de la posició, s'han de considerar 3 variables x, y i z de tipus float64 que corresponen a les coordenades de posició que es volen representar. Per la part d'orientació, s'han de considerar 4 variables x, y, z i w de tipus float64 que corresponen als angles d'orientació que es volen representar en forma de quaternió.
  Aquest tipus de missatges s'utilitzarà principalment per transmetre posicions i orientacions que siguin fixes, com una posició i orientació inicials.

- geometry_msgs/PoseStamped: Aquest tipus de missatge és exactament igual que un Pose, però també té un header que consta d'un string identificador, un marcador de temps i un seq, que modifica de manera incremental un número que identifica cada missatge que s'envia. S'utilitzarà en missatges que continguin posicions i orientacions que canviïn amb el temps, com la posició de l'End Effector del dispositiu hàptic.

- geometry_msgs/Vector3: Aquest tipus de missatge conté simplement un vector de 3 components, anomenades x, y i z. S'utilitzarà principalment per enviar les tres velocitats lineals de l'End Effector del dispositiu hàptic.

- geometry_msgs/WrenchStamped: Aquest tipus de missatge s'utilitza principalment per representar forces i moments. Està format per 2 missatges Vector3, és a dir, un vector de 3 components per les forces en els tres eixos i un altre vector de 3 components pels moments en els tres eixos.

- std_msgs/Float64: Missatge que consta únicament d'una variable numèrica de tipus float64. S'utilitzarà principalment per aspectes del projecte on només és necessari enviar un determinat valor, com en els factors d'escala de posició i d'angle.

- std_msgs/Bool: Missatge que consta únicament d'una variable booleana de tipus

bool. S'utilitzarà en tot un seguit de funcionalitats de l'aplicació gràfica que serviran per seleccionar modes de treball.

- rosgraph_msgs/Log: Missatge que serveix per transmetre informació dels log_messages de ROS. La informació més rellevant en aquest projecte d'aquests missatges són els camps "level" (que especifica el tipus de log message) i "msg" (que conté un string amb el missatge que es mostra en el log message).

7.4. Serveis

Els topics són un sistema molt flexible, però el fet que siguin un paradigma unilateral on molts nodes poden publicar missatges en un topic fent que molts altres nodes rebin aquest missatge pot fer que a vegades sigui difícil treballar amb topics quan volem que només un dels nodes executi una determinada tasca.

És aquí quan els serveis es tornen útils. Al contrari que els topics, els serveis es basen en interaccions de request/response.

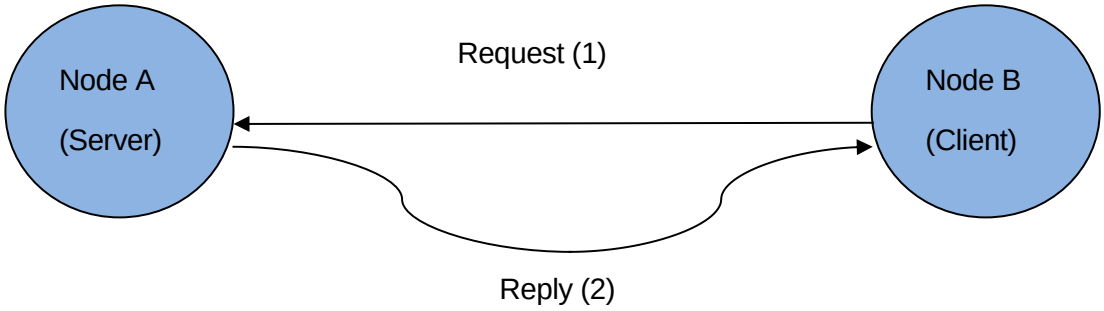
Una interacció de request/response consta de dos missatges: primer es transmet un missatge que funciona com una petició (request) per dur a terme una tasca i seguidament es retorna un missatge (response) que pot servir per informar de si la tasca s'ha complert satisfactòriament o no, o contenir valor de qualsevol altre tipus de variable (Esquema 4).

De la mateixa manera que en els topics els nodes poden ser "publishers" o "subscribers", en els serveis els nodes poden ser "clients", que són aquells nodes que envien una request i esperen una response, o "service", que són els nodes que realitzen una tasca i informen sobre el resultat un cop rebuda una petició.

Els serveis estan definits per un nom únic i s'han de definir en un arxiu .srv que s'ha d'incloure dins del package del node que ha de presentar aquest servei.

A l'igual que els topics, també tenen un tipus, que en aquest cas és simplement el nom del package on està definit aquest arxiu .srv acompanyat del nom de l'arxiu .srv mateix.

Per exemple, si tenim un package anomenat package_A on, en una carpeta anomenada "srv", tenim un arxiu que es diu service_A.srv, el tipus del servei serà: package_A/service_A.



Esquema 4. Diagrama de mostra del funcionament dels serveis.

7.4.1. Rosservice

La comanda que ens permet explorar els serveis del nostre sistema és rosservice. Com als apartats anteriors, es pot introduir al terminal la següent comanda per tal de fer una llista de tots els serveis actius:

```
$ rosservice list
```

També és possible actuar com un client d'un servei i enviar un request des de el terminal a un node que disposa d'un servei. Per fer-ho, cal introduir la comanda següent:

```
$ rosservice call /nom_del_servei argument_1 argument_2
```

on cal especificar el nom que té el servei acompanyat dels arguments que siguin necessaris per cridar aquest servei.

7.4.2. Serveis utilitzats en el projecte

Els serveis que es fan servir al projecte són:

- /delta_haptic_device_driver_node/setForceAndTorque: servei del node del dispositiu hàptic que permet que el dispositiu hàptic apliqui les forces i moments que l'usuari dessitgi.

- /iri_staubli_controller/move_in_cart: servei del robot Stäubli que permet moure'l en l'espai cartesià fins a una posició determinada. Cal notar que aquest servei és bloquejant, és a dir, d'ençà que s'envia la request fins que el robot envia la reply (que s'envia en quant el braç ha assolit la posició desitjada), la resta de funcionalitats de ROS queden en suspens.

- /iri_staubli_controller/move_in_joints: servei del robot Stäubli que permet moure'l en l'espai articular fins a una posició determinada. Cal notar que aquest servei també és bloquejant.

- /ftc_force_sensor_driver_node/set_zero: servei del sensor de forces i moments que permet designar la força i moment que el sensor estigui llegint en el moment de la crida com el nou 0.

7.5. rqt

Quan es treballa en ROS, moltes vegades és útil recórrer al package anomenat 'rqt'. rqt és un framework per software de ROS que implementa diferents eines gràfiques en forma de plugins.

Per exemple, rqt té un plugin anomenat Message Publisher, que permet a l'usuari fer publicacions a un topic de manera senzilla.

Aquesta eina és molt útil per poder visualitzar de manera fàcil molts aspectes de l'arquitectura de ROS.

Té la capacitat de poder executar més d'un plugin alhora, de manera que es pugui visualitzar diferent informació simultàniament, com es pot veure a la Figura 19. Mostra d'una sessió de rqt.Figura 19.

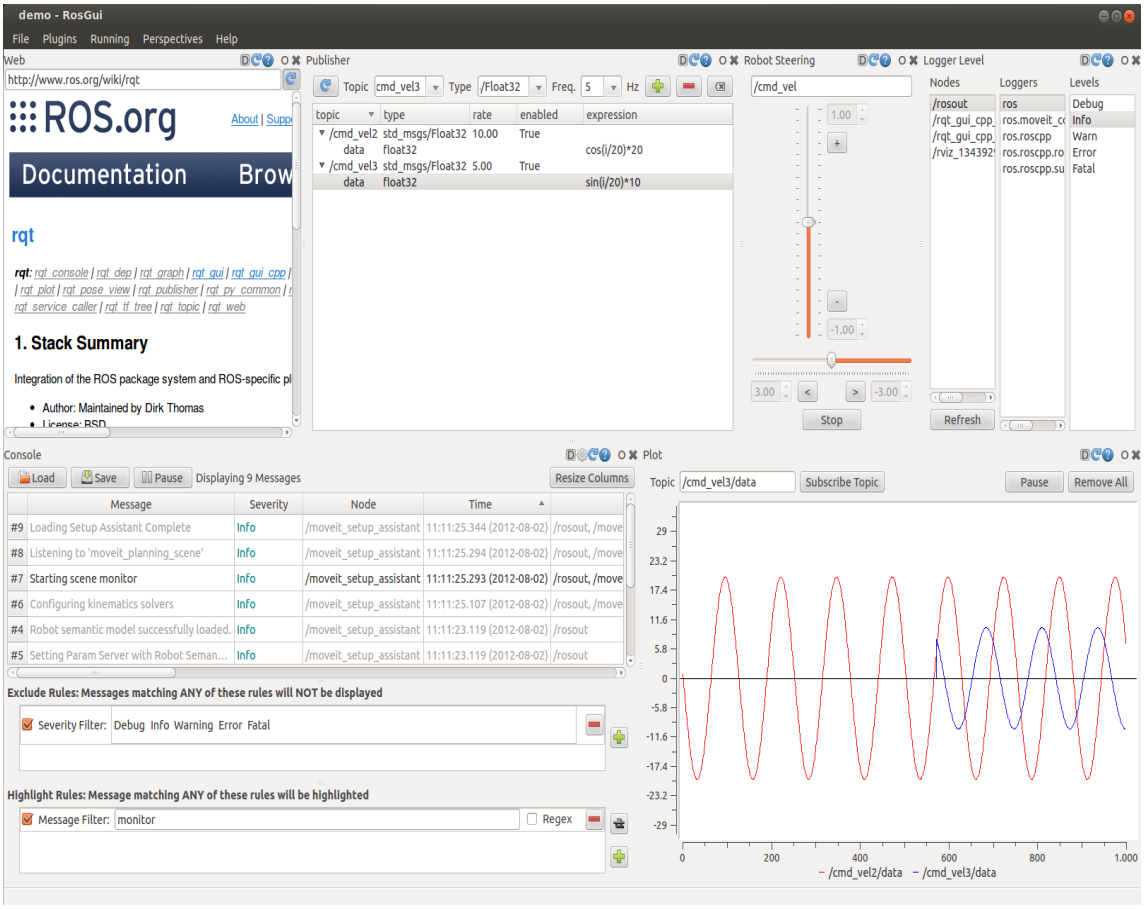


Figura 19. Mostra d'una sessió de rqt.

En aquesta Figura es poden veure alhora els plugins Publisher (part superior central), Robot Steering (part superior dreta), Console (part inferior esquerra) i Plot (part superior dreta). Sense entrar en detall en què fa cadascun d'aquests plugins, es pot veure que ofereixen una informació que pot ser molt interessant per a l'usuari.

Per fer-ho servir, s'ha d'introduir la comanda 'rqt' al terminal des de qualsevol directori un cop s'hagi executat un roscore en un altre terminal, com es veu a la Figura 20. Després, a l'apartat 'Plugins', s'ha de seleccionar el plugin que es vulgui executar.

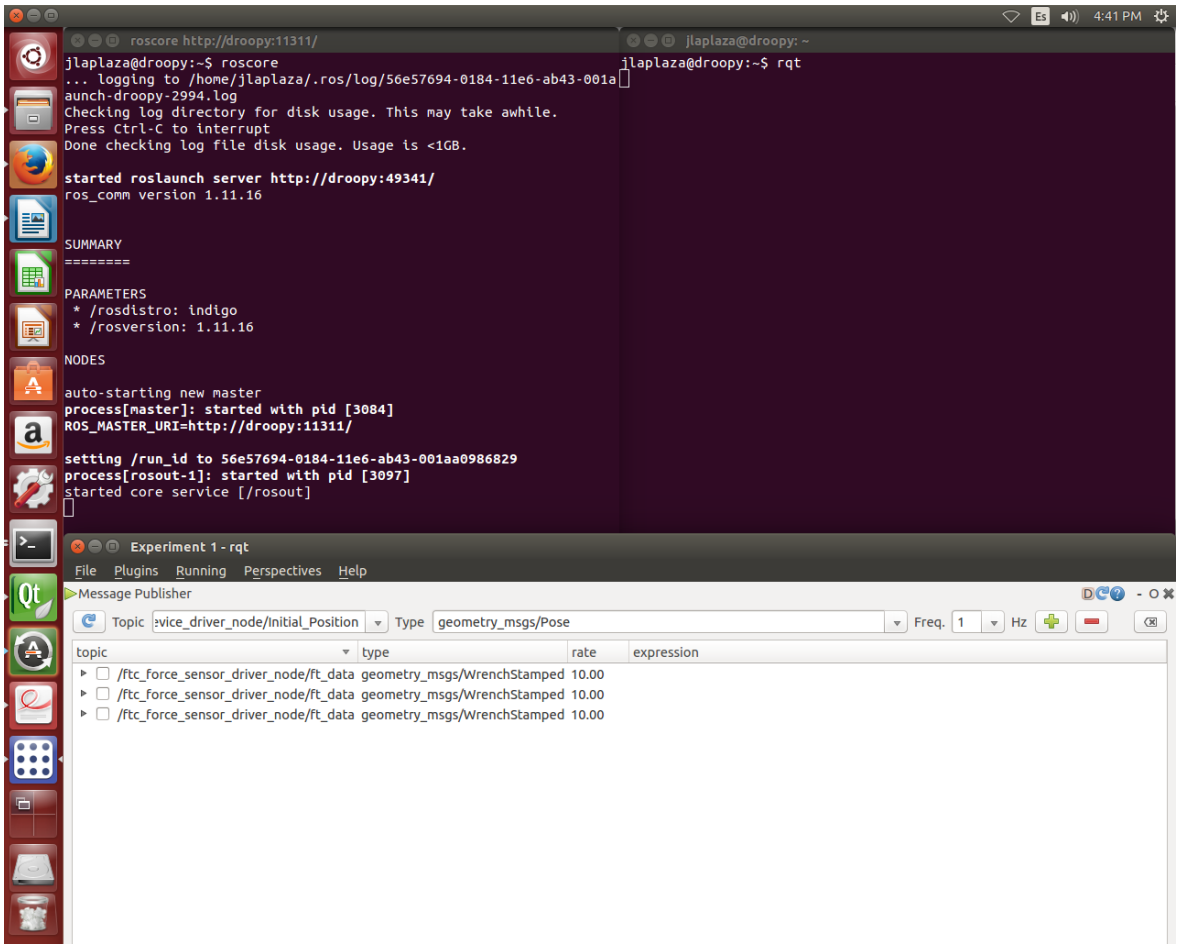


Figura 20. Exemple d'execució de l'eina rqt.

7.5.1. Plugins que s'han fet servir durant el projecte

- Message Publisher: Plugin molt útil per publicar informació a un topic. S'ha fet servir, per exemple, per publicar forces al topic en què publica el sensor de forces, de manera que el dispositiu hàptic llegeixi aquestes forces conegudes i veure com respon.

- Rviz: Plugin que permet visualitzar diferents tipus de topics en un sistema de coordenades. Molt útil per veure si la informació de posició que envia el dispositiu hàptic és l'esperada.

- Node Graph: Eina que mostra en un mapa els diferents nodes i topics actius que hi

ha al sistema. Molt útil per veure còm es relacionen els nodes entre ells.

- Plot: Plugin que mostra un gràfic amb l'evolució temporal d'una variable determinada d'un topic. Útil per veure l'evolució temporal de les velocitats que envia l'End Effector del dispositiu hàptic.

7.6. Log Messages

ROS té el seu propi mecanisme, anomenat rosout, per guardar els log messages que envien els nodes. Els log messages són missatges fàcilment interpretables pels humans que donen informació sobre l'estatus d'un node. Hi ha cinc tipus de log messages, de menys crític a més crític:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

Un exemple senzill d'utilització d'aquests missatges en el codi del node seria el següent:

```
ROS_INFO("Node ready.");
```

D'aquesta manera, quan el codi arriba a aquesta línia, mostra per la pantalla del terminal el missatge "Node ready.".

De fet, no només mostra el missatge per la pantalla, sinó que el publica al topic que s'ha vist anteriorment anomenat "rosout". Alhora, el topic "rosout_agg" torna a publicar aquests missatges per si algun node els vol rebre.

7.7. Rosbag

Rosbag és un package de ROS que permet guardar i repetir els missatges que es publiquen a un topic.

Per fer servir aquesta eina al codi, cal crear un objecte de tipus rosbag::Bag, utilitzar el mètode "open" i concretar el nom del fitxer que s'utilitzarà i el mode en què s'obre aquest fitxer (read o write).

En aquest projecte, s'utilitzarà únicament en mode write, ja que servirà per escriure en un fitxer els missatges que es publiquin a diferents topics.

Per escriure en el fitxer, s'utilitza el mètode "write", especificant el nom del topic, el temps d'escriptura i l'objecte que conté el missatge.

Sempre s'ha de recordar, al final, d'utilitzar el mètode "close".

Els arxius resultants d'aquest procés són desats a la carpeta del workspace de manera predeterminada.

8. Node del Stäubli Robot

Primer, cal recordar els objectius marcats per al braç robòtic:

- Capacitat del braç robòtic de seguir adequadament les posicions que el dispositiu hàptic envia.
- Permetre a l'usuari moure el braç robòtic tant a l'espai articular com a l'espai cartesià i d'enviar les dades de la posició de l'extrem del braç o Tool Centre Point (TCP) a l'aplicació gràfica.

Per poder comunicar el dispositiu hàptic amb el braç robòtic, com especifica el primer objectiu, cal primer analitzar de quina manera aquest pot seguir les consignes de posició enviades pel dispositiu hàptic.

Per poder assolir el segon objectiu, cal veure de quina manera el node del braç robòtic comunica a altres nodes la posició del TCP

Bàsicament, la implementació consisteix a fer que cada cop que s'executi un callback del topic pertinent a la posició del dispositiu hàptic, el TCP es mogui seguint aquesta consigna.

La Figura 10 mostra el mapa de grafs conceptual del funcionament del braç robòtic aprofitant el plugin Node Graph de rqt.

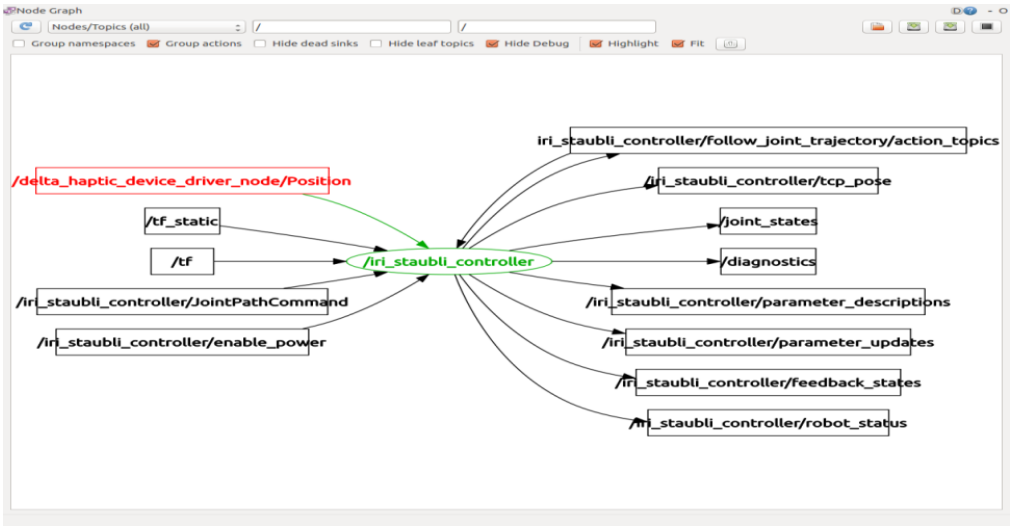


Figura 21. Mapa que mostra el node de ROS del braç robòtic de color verd i tots els topics que publica i als quals està subscrit. En vermell hi ha el topic per on rep consignes de posició del dispositiu hàptic.

Com es pot veure a la Figura, hi ha un topic anomenat /delta_haptic_device_driver_node/Position al qual està subscrit el node del braç robòtic.

També és interessant notar que el mateix node publica tot una sèrie de topics, entre els quals hi ha els topics /joint_states i /iri_staubli_controller/tcp_pose que envien informació sobre el valor de les articulacions del braç i la posició del TCP a l'espai cartesià respectivament. Aquests topics seran utilitzats més endavant.

9. Node del FTC Force Sensor

Cal recordar els objectius proposats per al sensor de forces i moments:

- Capacitat del sensor de forces d'enviar adequadament les forces i moments que està registrant al dispositiu hàptic i a l'aplicació gràfica.
- Permetre a l'usuari que fa servir l'aplicació gràfica designar com a zero la força i moment que el sensor estigui enregistrant en aquell moment.

Per assolir el primer objectiu i poder desenvolupar el node del dispositiu hàptic, es va haver d'analitzar el node del sensor de forces i moments per veure de quina manera aquest compartia informació en ROS.

Executant el node i fent servir el rqt_graph, s'obté el següent diagrama (Figura 22):

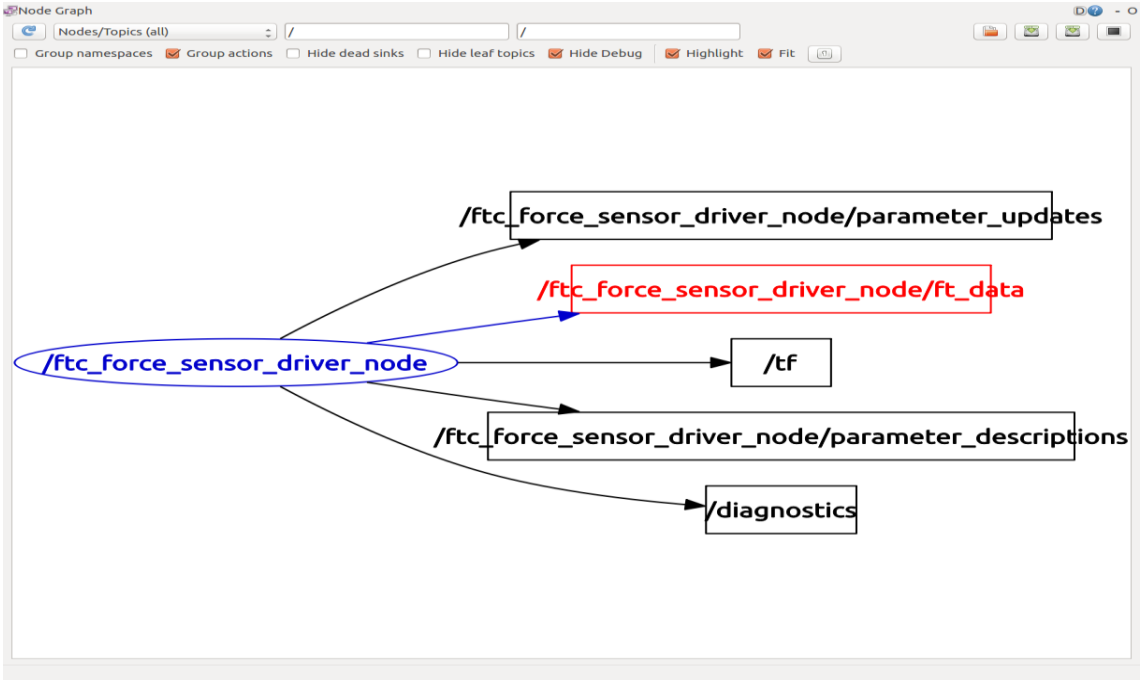


Figura 22. Mapa que mostra el node de ROS del sensor de forces i els topics que aquest publica. En vermell es mostra el topic on publica les forces i moments que registra.

S'hi pot veure que el node del sensor de forces i moments publica un topic anomenat 'ftc_force_sensor_driver_node/ft_data'.

Dintre del codi del node, es pot observar el següent:

```
// [init publishers]

  this->ft_data_publisher_ = this-
>public_node_handle_.advertise<geometry_msgs::WrenchStamped>("ft_data",
10);
```

on es pot veure que s'inicialitza un publisher que publica missatges de tipus WrenchStamped, un tipus de topic utilitzat per representar forces i moments.

També es pot veure el següent codi:

```
// [publish messages]
  this->driver_.get_force_torque(force_torque,sec,nsec);
  this->ft_data_WrenchStamped_msg_.header.frame_id=this-
>driver_.get_sensor_frame();
  this->ft_data_WrenchStamped_msg_.header.stamp=ros::Time(sec,nsec);
  this->ft_data_WrenchStamped_msg_.wrench.force.x=force_torque[0];
  this->ft_data_WrenchStamped_msg_.wrench.force.y=force_torque[1];
  this->ft_data_WrenchStamped_msg_.wrench.force.z=force_torque[2];
  this->ft_data_WrenchStamped_msg_.wrench.torque.x=force_torque[3];
  this->ft_data_WrenchStamped_msg_.wrench.torque.y=force_torque[4];
  this->ft_data_WrenchStamped_msg_.wrench.torque.z=force_torque[5];
  this->ft_data_publisher_.publish(this->ft_data_WrenchStamped_msg_);
```

on es veu que, efectivament, aquest topic conté la informació sobre les forces que està registrant el sensor.

També, executant el node i manipulant el sensor, gràcies a l'eina 'rostopic echo', es va poder comprovar que el node del sensor publica al topic aquestes forces i moments. Per tant, tant el dispositiu hàptic com l'aplicació gràfica hauran d'estar subscrits a aquest topic per tal de rebre la informació relativa a les forces i moments.

Per poder assolir el segon objectiu és interessant comprovar quins serveis proporciona el node del sensor hàptic.

Es comproven els serveis disponibles del node del sensor amb l'eina 'rosservice list', obtenint la següent llista (Figura 23):

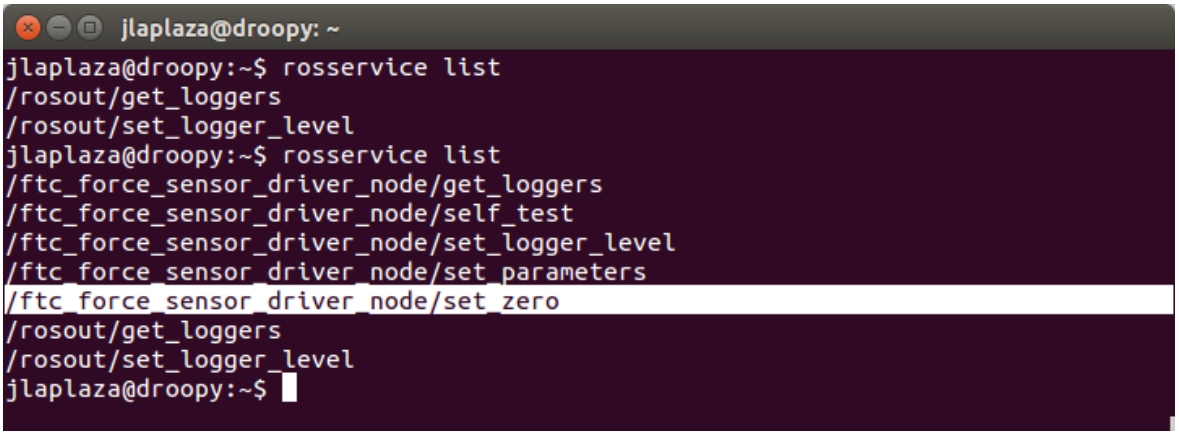


Figura 23. Resultat de la cerca de serveis amb la comanda "rosservice list".

Es pot veure un servei anomenat ' /ftc_force_sensor_driver_node/set_zero'. En el codi del node, es pot trobar el següent codi:

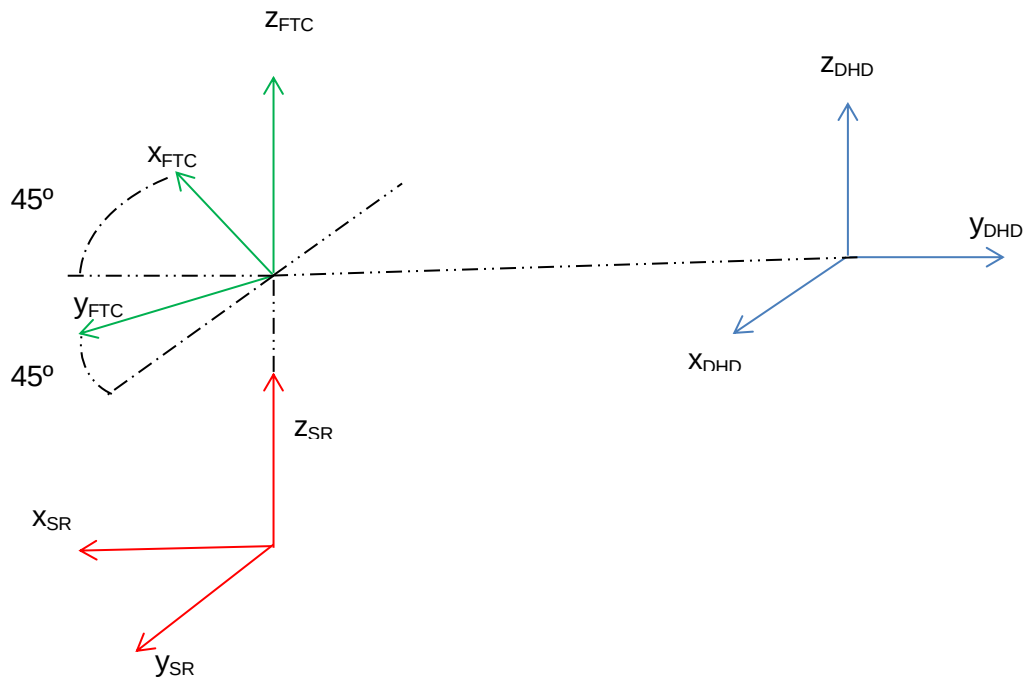
```
// [init services]
  this->set_zero_server_ = this-
>public_node_handle_.advertiseService("set_zero",
&FtcForceSensorDriverNode::set_zeroCallback, this);
/*  [service callbacks] */
bool FtcForceSensorDriverNode::set_zeroCallback(std_srvs::Empty::Request
&req, std_srvs::Empty::Response &res)
{
  ROS_INFO("FtcForceSensorDriverNode::set_zeroCallback: New Request
Received!");
  this->driver_.set_zero();
```

En cridar el servei "set_zero", es fa una crida a una funció anomenada 'set_zero()' del driver de baix nivell. És un servei que no necessita cap paràmetre d'entrada.

Per tant, s'utilitzarà aquest servei per la funcionalitat de l'aplicació gràfica d'executar un 'set zero' al sensor de forces i moments.

10. Conciliació dels sistemes de referència

És molt important tenir present la orientació del sistema de referència de cadascun dels dispositius fets servir. Es pot veure un croquis de la disposició d'aquests sistemes de referència a l'Esquema 5:



Esquema 5. Croquis dels sistemes de referència del DHD, el braç robòtic i el sensor de forces.

Els eixos de color blau representen el sistema de referència del dispositiu hàptic, els eixos de color vermell els del braç robòtic i els de color verd els del sensor de forces.

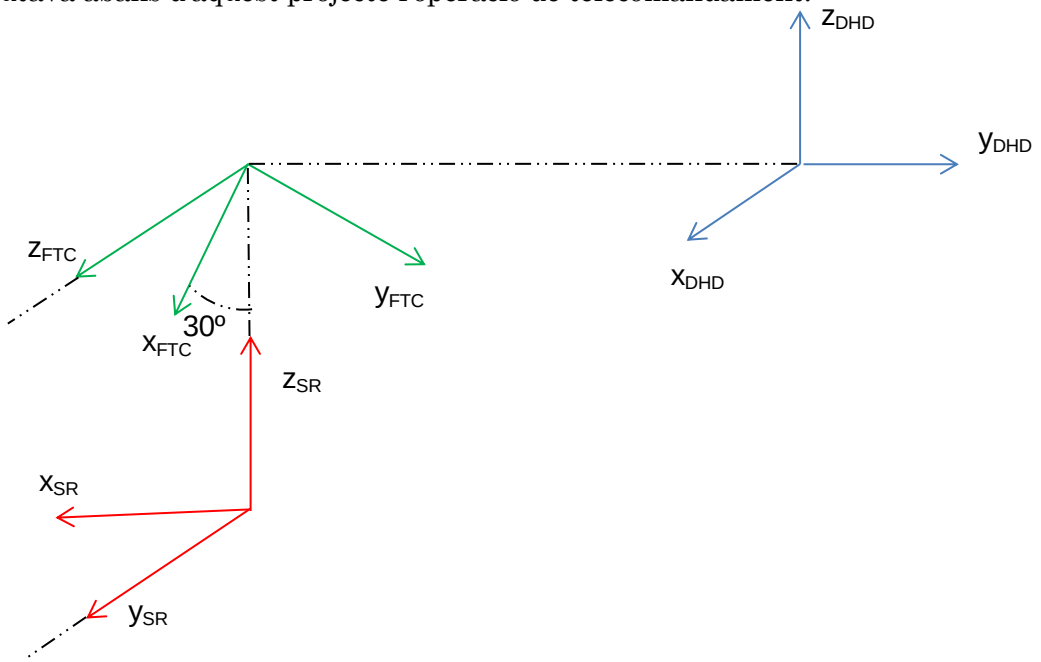
Com es pot veure, tots tres dispositius presenten un eix z amb la mateixa direcció i sentit. No obstant això, el sistema de referència del braç robòtic és equivalent al sistema de referència del dispositiu hàptic girat -90º entorn l'eix z. De la mateixa manera, el sistema de referència del sensor de forces és equivalent al sistema de referència del braç robòtic girat -45º entorn l'eix z.

Aquestes diferències seran un factor a tenir en compte, ja que l'objectiu és que el braç robòtic i el dispositiu hàptic es moguin de manera simultània i que les forces que enviï el sensor de forces siguin útils a l'usuari que fa servir el dispositiu hàptic.

És a dir, s'ha d'evitar situacions en què el DHD i el braç robòtic es moguin de manera antiparal·lela als eixos x i y o que l'usuari no pugui reconèixer correctament la direcció en la qual s'està detectant una força o moment.

Per evitar aquest fenomen, s'ha modificat el codi de cada node que rep la seva consignes pertinent, ja que el més natural és que cada node enviï consignes en el sistema de referència natural del dispositiu, i que el node que rep aquestes consignes processi aquesta informació segons el topic per on la rep.

L'estudi de tots els diferents casos en què es pot posicionar el sensor de forces, canviat per tant la seva referència, no entra dins de l'abast del projecte, però s'ha definit la següent posició de la referència del sensor com a posició de treball estàndard per provar el sistema de telecomunicació (Esquema 6), ja que és semblant a la posició en la qual s'executava abans d'aquest projecte l'operació de telecomandament.



Esquema 6. Esquema on es mostra la posició de treball escollida pel sensor de forces.

També es pot veure la Figura 24 cóm és aquesta situació posicionant els diferents dispositius del laboratori com correspon.

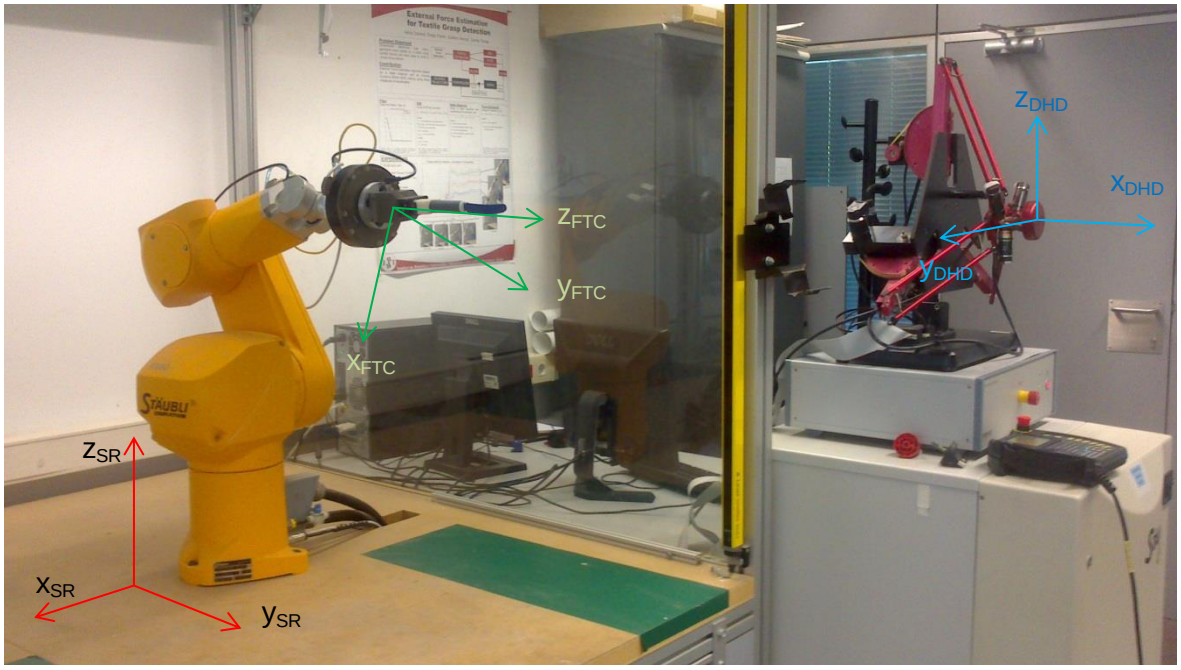


Figura 24. Disposició dels elements adoptada com a posició estàndard per l'operació de telecomandament.

Per tant, en el node del braç robòtic s'ha implementat la funció de seguiment de posicions enviades pel DHD intercanviant els eixos x i y. Per exemple, quan el dispositiu hàptic es mogui positivament en el seu eix x, el braç robòtic es mourà positivament en el seu eix y.

Al node del DHD s'ha d'implementar el codi de manera que quan, per exemple, el sensor registri una força o moment al seu eix x, el DHD apliqui la força corresponent en els eixos y i z, projectant les forces com correspongui. Quan el sensor registri una força o moment al seu eix z, aquest/a s'aplicarà a l'eix x del DHD.

Val a dir que, com també es veurà a l'apartat 13, la funcionalitat de definir noves posicions de treball per al sensor de forces i pel DHD es deixarà com a treball futur (de fet, ja s'ha començat a implementar).

11. Software del Delta Haptic Device

Gràcies a l'anàlisi dels apartats anteriors, queda clar que la situació prèvia a aquest projecte és la següent (Figura 25):

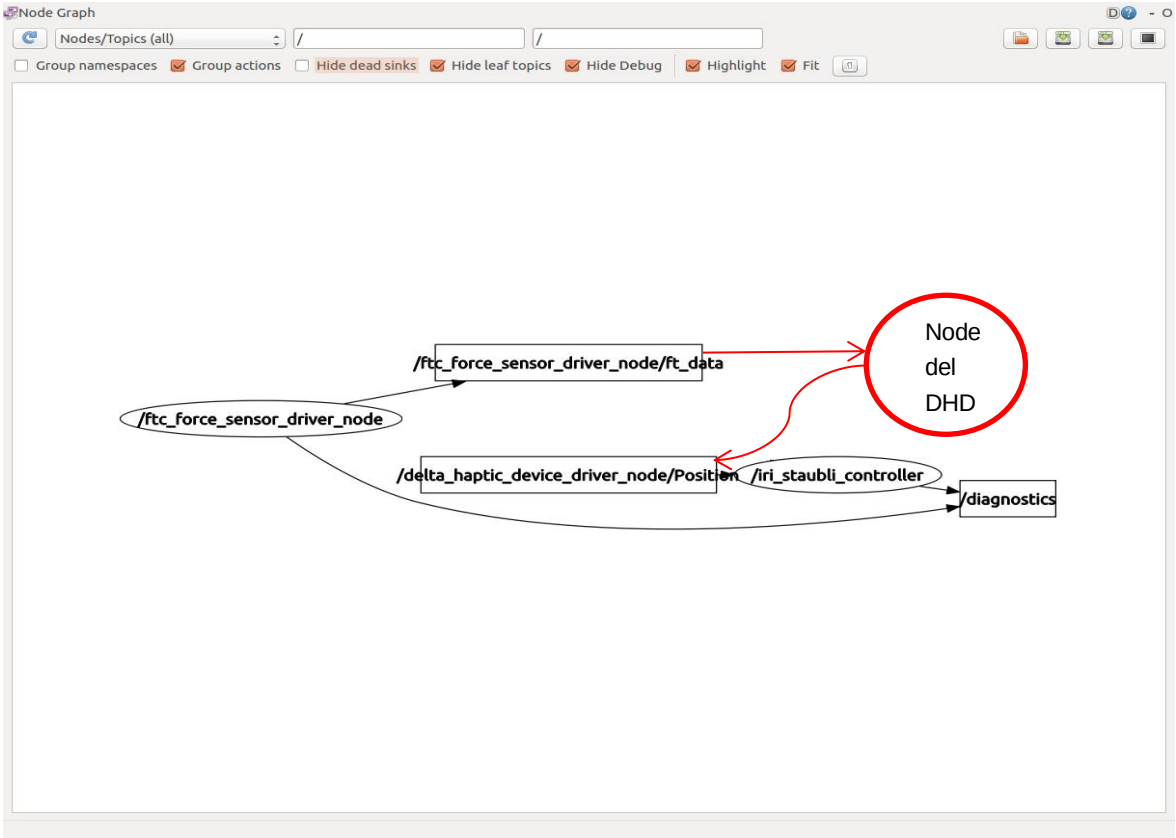


Figura 25. Mapa de les connexions entre nodes i topics del sistema abans de la creació del node del DHD.

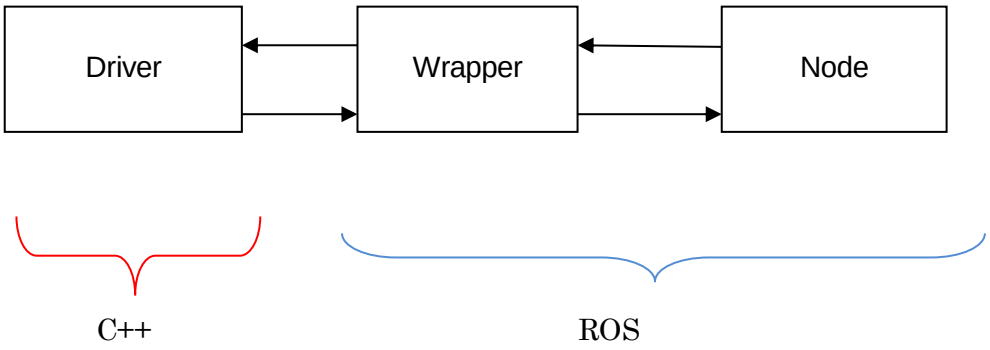
Com es pot veure a la figura, existeixen dos nodes, un pel sensor de forces i altre pel braç robòtic. El primer publica un topic amb les forces i moments que mesura el dispositiu i el segon està subscrit a un topic on s'hi publiquen posicions i orientacions. En vermell s'indica el resultat esperat de l'arquitectura de comunicacions amb el node que es desenvolupa en aquest projecte (es pot comparar amb el resultat obtingut un cop desenvolupat el projecte a la Figura 33)Figura 32.

A continuació s'exposarà tot el software desenvolupat per la integració del Delta Haptic Device.

11.1.Driver

11.1.1. Mètode de creació

A continuació es descriuen els passos que s'han seguit per crear el driver i el node de ROS del dispositiu hàptic. Aquests passos segueixen la filosofia de treball pel desenvolupament d'aquests tipus de projectes de l'IRI, com es pot veure a [6].



Esquema 7. Diagrama que mostra el procés de comunicació entre el driver i el node de ROS.

Com es veu a l'Esquema 7, l'objectiu final és la creació d'un node de ROS. Aquest node treballarà fent crides a funcions definides al wrapper, que alhora haurà d'executar una o vàries funcions definides al driver. Per tant, el sistema treballa en cadena: el node acudeix al wrapper per saber què fa una funció, el wrapper acudeix al driver per saber què fa aquesta funció, el driver dóna una resposta al wrapper que alhora respon també al node.

Aquest sistema s'utilitza per tal d'aprofitar la gran velocitat d'execució del llenguatge C++.

11.1.2. API del dispositiu hàptic

Abans de començar, cal recordar quines tasques ha de realitzar el dispositiu hàptic:

- Ser capaç d'identificar la posició, orientació i velocitat lineal de l'End Effector.
- Ser capaç d'aplicar d'aplicar forces i moments a l'End Effector.

Per tant, es van identificar quines funcions de l'API del dispositiu hàptic permetrien realitzar aquestes tasques.

A l'arxiu dhd.h de l'API original del DHD es troben totes les funcions que el dispositiu hàptic és capaç de realitzar. Les funcions que s'han seleccionat per aconseguir els objectius inicials són les següents:

1. Detecció de posició, orientació i velocitat lineal

- **int __SDK dhdGetPositionAndOrientationDeg(double *px, double *py, double *pz, double *oa, double *ob, double *og, char ID = -1)**
  Funció que fa que les variables px, py i pz prenguin els valors de la posició x, y i z en el moment de la crida.
  També fa que les variables oa, ob i oc prenguin els valors de l'orientació dels angles d'Euler oa, ob i oc de l'End Effector en el moment de la crida. Els angles s'expressen en graus.

- **int __SDK dhdGetLinearVelocity (double *vx, double *vy, double *vz,char ID= -1)**
  Funció que fa que les variables vx, vy i vz prenguin el valor de les velocitats x, y i z de l'End Effector en el moment de la crida.

2. Aplicació de la força i moment

- **int __SDK dhdEnableForce (uchar val, char ID = -1)**
  Funció que serveix per activar els motors de forces i de moments.
- **int __SDK dhdSetForceAndTorque(double fx, double fy, double fz, double tx, double ty, double tz, char ID = -1)**
  Funció que aplica a l'End Effector unes forces i moments equivalents al valor numèric que hi ha a les variables fx, fy, fz, tx, ty, tz.

3. Altres funcions necessàries pel funcionament del dispositiu hàptic

- **int __SDK dhdOpen ()**

  Funció que inicia la connexió entre el dispositiu hàptic i l'ordinador.

- **int __SDK dhdEnableExpertMode()**

  Funció que habilita la utilització de certes funcions del dispositiu catalogades com a funcions de mode expert, ja que utilitzades de manera incorrecta podrien arriba a ser perilloses.

- **void __SDK dhdGetSDKVersion (int *major, int *minor, int *release, int *revision)**

  Funció que permet obtenir la versió del dispositiu hàptic que s'està utilitzant.

- **void __SDK dhdGetSerialNumber (ushort* sn, char ID)**

  Funció que retorna el número de sèrie del dispositiu.

- **int __SDK dhdGetVersion (double* ver, char ID)**

  Funció que retorna la versió del controlador.

- **const char* __SDK dhdGetSystemName(char ID)**

  Retorna el tipus de dispositiu hàptic (com a variable de tipus string).

- **int __SDK dhdGetSystemType (char ID)**

  Retorna el tipus de dispositiu hàptic (com a variable de tipus int).

- **int __SDK dhdClose ()**

  Funció que finalitza la connexió entre el dispositiu hàptic i l'ordinador.

- **const char* __SDK dhdErrorGetLastStr ()**

  Funció que retorna un string que explica l'últim error que ha experimentat el dispositiu hàptic.

- **int __SDK dhdStop (char ID = -1)**

  Funció que desactiva els motors del dispositiu hàptic.

- **int __SDK dhdDisableExpertMode ()**

  Funció que desactiva el mode expert.

11.1.3. Creació de la llibreria de C++ delta_haptic_device

En primer lloc, es crea l'estructura de carpetes:

```
$ mkdir iri-lab
$ cd iri-lab
$ svn checkout https://devel.iri.upc.edu/labrobotica/ --depth
immediates
$ cd labrobotica
$ svn update --set-depth infinity algorithms/ drivers/ tools/
```

La llibreria s'estableix a la carpeta de drivers i es descarreguen tots els scripts de l'IRI, que permetran mecanitzar la creació de les carpetes del driver, crear el package del node de ROS, afegir publishers o subscribers al node, etc:

```
$ cd ~/iri-lab/labrobotica/drivers
$ mkdir delta_haptic_device
$ cd delta_haptic_device
$ wget http://www.iri.upc.edu/people/shernand/template.tar.gz
$ tar -zxvf template.tar.gz
$ ./new_project.sh -t driver -p "Delta Haptic Device Driver" -
n delta_haptic_device
```

S'accedeix a la carpeta trunk i després a la carpeta src:

```
$ cd trunk/src
```

En aquesta carpeta, s'han de definir el header i la implementació de totes les funcionalitats del dispositiu hàptic.

11.1.4. Header

El header és un manifest de totes les classes, funcions, atributs, etc que presentarà el driver. És un arxiu .h anomenat en aquest cas delta_haptic_device.h que es pot consultar a l'Annex B 2.1.

El primer que s'especifica al header del driver del dispositiu hàptic (delta_haptic_device.h) són els includes.

Els includes serveixen per poder utilitzar funcions que estan incloses a altres llibreries.

```
#include <iostream>
#include "dhdc.h"
```

En primer lloc, s'inclou la llibreria de C++ iostream, que agilitza l'input i output de variables.

També s'inclou el header de la llibreria del dispositiu hàptic, l'arxiu dhdc.h, per tal de poder utilitzar les funcions d'aquesta.

```
class CDelta_Haptic_Device
{

  public:
    CDelta_Haptic_Device();
    ~CDelta_Haptic_Device();
    bool ON;
    bool status_ok;
```

El següent és definir el nom de la classe que utilitzarà el driver. En aquest cas,el nom de la classe és CDelta_Haptic_Device. També es defineix directament el constructor, que és una funció que s'executa automàticament en crear un objecte d'un classe i s'utilitza per inicialitzar els valors de les diferents variables internes, i el destructor, que és la funció que s'executa al eliminar un objecte creat a una classe.

També es crearan dos atributs de tipus booleans anomenats ON i status_ok, que prendran el valor de true quan l'ordinador hagi pogut establir comunicacions amb el dispositiu hàptic (amb la funció dhdOpen()) i quan s'hagi comprovat que la versió del SDK, el Serial Number, la versió i el nom del sistema són els esperats, respectivament.

```
int getPositionAndOrientation (double& px, double& py, double& pz, double&
oa, double& ob, double& oc);
```

Es defineix la funció getPositionAndOrientation, que necessita 6 arguments, 3 de posició i 3 d'orientació. Permetrà actualitzar el valor de les variables de posició i orientació.

```
int getLinearVelocity (double& vx, double& vy, double& vz);
```

Es defineix la funció getLinearVelocity, que necessita 3 arguments, la velocitat lineal en cada direcció de l'espai. Permetrà actualitzar les variables de velocitat lineal.

```
int open();
```

Funció que serà cridada en el moment de voler fer servir el dispositiu hàptic. Contempla possibles excepcions, com ara intentar fer un open() mentre el dispositiu hàptic no està endollat o intentar fer dues vegades la funció open().

```
int setForceAndTorque (double fx, double fy, double fz, double tx,
double ty, double tz);
```

Funció que rep com arguments les forces i els moments en els 3 eixos de coordenades. Permetrà aplicar unes forces i moments iguals al valor numèric de les variables fx, fy, fz, tx, ty i tz. Es limita tota força a un valor absolut de 10 Newtons i tot moment a un valor absolut de 0.2 N·m.

Totes les funcions retornen els possibles error que poden experimentar per pantalla mitjançant excepcions.

La implementació de totes aquestes funcions es pot consultar a l'Annex B 2.2.

11.1.5. Implementació de l'exemple del driver

Per comprovar que el driver de baix nivell funciona, s'ha creat un exemple d'aplicació del dispositiu hàptic.

L'exemple s'encarrega de mantenir l'End Effector en suspensió (força i moments nuls) de manera que l'usuari pugui moure'l sense dificultat. Alhora, es mostra tot un seguit d'instruccions a l'usuari pel terminal (Figura 26).

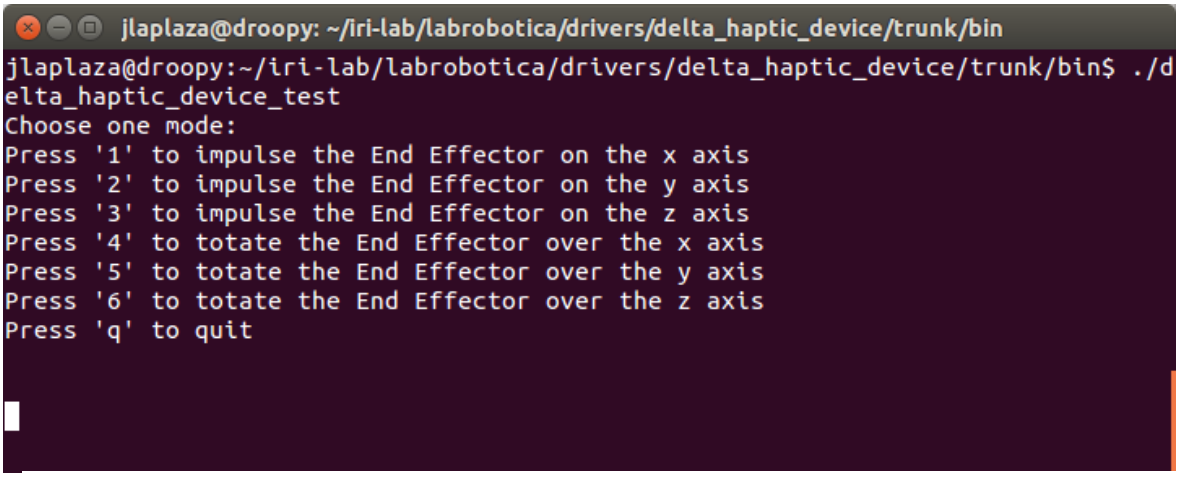


Figura 26. Menú que ofereix l'exemple dissenyat.

Les indicacions demanen a l'usuari seleccionar un mode de treball prement la tecla indicada. Cadascun d'aquests modes aplicaran una petita força o moment en cadascun dels tres eixos de referència durant un petit interval de temps, com petits impulsos de força o moment, de manera que l'usuari pugui comprovar que tot funciona correctament. Per sortir d'aquest mode, és suficient amb prémer la tecla 'q'.

Aquest exemple es crea dintre del directori anomenat 'examples' a la carpeta 'src'. Per executar-ho, cal entrar al directori '~/iri-lab/labrobotica/drivers/delta_haptic_device/trunk/bin' i executar la comanda './delta_haptic_device_test' un cop s'hagi compilat. El seu codi es troba a l'Annex B 2.3.

11.1.6. Compilar la llibreria, l'exemple i la documentació

Per compilar la llibreria i l'exemple, cal entrar a la carpeta 'build' del driver i executar les següents comandes.

```
$ cmake ..
$ make
```

Si no hi ha hagut cap problema amb la compilació, hauria d'aparèixer un arxiu anomenat 'libdelta_haptic_device.a' (la llibreria) a la carpeta 'lib' del driver i un executable anomenat 'delta_haptic_device_test' (l'executable del test) a la carpeta anomenada 'bin'.

Per instal·lar la llibreria al sistema (pas **CRÍTIC** per a la utilització del node del dispositiu hàptic ja que si no, el node de ROS no serà capaç de trobar les funcions definides al driver) cal anar al directori 'build' i introduir la següent comanda:

```
$ sudo make install
```

El terminal demanarà la contrasenya de l'usuari per realitzar aquesta operació i un cop introduïda s'instal·larà la llibreria.

Per compilar la documentació, es pot introduir al mateix directori la comanda:

```
$ make document
```

Aquesta compilació es fa gràcies al programa Doxygen, que permet crear una sèrie de documents de text que contenen informació sobre el driver.

11.2.Node de ROS

11.2.1. Creació del package

Per poder treballar amb ROS, el primer pas és crear un workspace. Per fer-ho, es pot introduir al terminal les següents operacions:

```
$ mkdir -p ~/iri-lab/iri_ws/src
$ cd ~/iri-lab/iri_ws/src
$ catkin_init_workspace
$ cd ~/iri-lab/iri_ws
$ catkin_make
$ echo "source ~/iri-lab/iri_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ cd ~/iri-lab/iri_ws/src
$ wstool init
$ wstool update
$ roscd
$ cd ..
$ catkin_make
```

Després, s'ha d'introduir com a mínim el package 'iri_core'.

```
$ roscd
$ cd ../src
$ wstool set iri_core --svn
https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/metapackages/iri_core
```

I després instal·lar els scripts de ROS de LabRobòtica:

```
$ sudo apt-get install realpath
$ svn checkout https://devel.iri.upc.edu/labrobotica/ros/iri-ros-
pkg_hydro/scripts-catkinscripts
$ echo "source `pwd`/scripts/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Per crear el package, es va executar el següent script al terminal:

```
$ create_driver_package.sh -n delta_haptic_device
```

A continuació es van haver de completar els arxius Cmakelists.txt i package.xml. Aquest és un pas molt important quan es treballa en ROS, ja que aquests arxius contenen la informació que ROS necessita per poder compilar els packages, com ara dades sobre la llibreria del driver o les dependències del package amb altres packages.

Els dos arxius es poden trobar a l'Annex C 3.1 i 3.2.

11.2.2. Wrapper

Dintre de la carpeta d'include del package 'iri_delta_haptic_device', es defineix l'arxiu 'delta_haptic_device_driver.h', el header del wrapper que es troba a l'Annex C 3.3.

El header i la majoria del codi que conté es forma amb els scripts de l'IRI. Per tant, es comentaran a continuació les parts que s'han afegit o modificat del codi original proporcionat pel script, ja que són les que afecten específicament al dispositiu hàptic.

```
#include <iri_base_driver/iri_base_driver.h>
#include <iri_delta_haptic_device/DeltaHapticDeviceConfig.h>
#include "delta_haptic_device.h"
```

En primer lloc, s'ha d'incloure el header del driver per tal de poder utilitzar en la implementació les funcions definides en el driver.

```
int getPositionAndOrientation (double& px, double& py, double& pz, double&
oa, double& ob, double& oc);
```

De la mateixa manera que en el driver, aquesta funció rep com a paràmetres tres variables que representen posicions i tres variables que representen angles i canvia el valor numèric d'aquestes variables donant-les-hi el valor de la posició i orientació de l'End Effector del dispositiu hàptic en el moment de la crida de la funció.

```
int getLinearVelocity (double& vx, double& vy, double& vz);
```

El mateix succeeix amb la següent funció, que rep com a paràmetres tres variables que representen les velocitats lineals, el valor de les quals és actualitzat amb el valor de la velocitat lineal en cadascun dels tres eixos de coordenades del dispositiu hàptic en el moment de la crida.

```
int setForceAndTorque (double fx, double fy, double fz, double tx, double
ty, double tz);
```

Per últim, aquesta funció rep sis paràmetres, tres variables que representen forces i tres variables que representen moments, i aplica sobre l'End Effector del dispositiu hàptic unes forces i moments equivalents al valor numèric d'aquestes variables.

La implementació de totes aquestes funcions es pot veure a l'arxiu 'delta_haptic_device_driver.cpp', a l'Annex C 3.4.

11.2.3. Node

11.2.3.1.Header

Aquest arxiu es troba a l'Annex C 3.5.

```
#include <iri_base_driver/iri_base_driver_node.h>
#include "delta_haptic_device_driver.h"
```

Primer de tot, cal incloure el header del wrapper per poder utilitzar les funcions definides al wrapper.

```
class DeltaHapticDeviceDriverNode : public
iri_base_driver::IriBaseNodeDriver<DeltaHapticDeviceDriver>
{
  double px, py, pz;
  double oa, ob, oc;
  double vx, vy, vz;
  double fx, fy, fz;
  double tx, ty, tz;
  double pxi, pyi, pzi;
  double oai, obi, oci;
  double dsf, asf;
  bool pospos, force_feedback, torque_feedback
```

A continuació, és necessari declarar tot un seguit de variables internes de posició, orientació, força, moment, velocitat, factor d'escala i mode de treball.

```
#include <geometry_msgs/Pose.h>
#include <geometry_msgs/WrenchStamped.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/PoseStamped.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Bool.h>
```

També s'han d'incloure els headers de cadascun dels tipus de topics que s'utilitzaran, que són:

- Un missatge de tipus geometry_msgs/Pose que publicarà la posició inicial del dispositiu hàptic al iniciar el node.
- Un missatge de tipus geometry_msgs/WrenchStamped per poder subscriure el

node del dispositiu hàptic a les forces i moments que envia el sensor de forces.

- Un missatge de tipus geometry_msgs/Vector3 per poder publicar les tres velocitats lineals de l'End Effector del dispositiu hàptic.
- Un missatge de tipus geometry_msgs/PoseStamped per poder enviar la posició i orientació de l'End Effector del dispositiu hàptic.
- Un missatge de tipus std_msgs/Float64 per poder subscriure el node del dispositiu hàptic al factor d'escala de posicions i orientacions de l'aplicació gràfica.
- Un missatge de tipus std_msgs/Bool per poder subscriure al dispositiu hàptic a diferents topics que permetin modificar el comportament del mateix mitjançant l'aplicació.

```
// [publisher attributes]
ros::Publisher Initial_Position_publisher_;
geometry_msgs::Pose Initial_Position_Pose_msg_;

ros::Publisher Velocity_publisher_;
geometry_msgs::Vector3 Velocity_Vector3_msg_;

ros::Publisher Position_publisher_;
geometry_msgs::PoseStamped Position_PoseStamped_msg_;


// [subscriber attributes]
ros::Subscriber ft_data_subscriber_;
Void ft_data_callback(const geometry_msgs::WrenchStamped::ConstPtr& msg);
```

També s'han de definir els publishers i els subscribers. En els publishers s'ha de concretar de quin tipus és el missatge que s'ha de publicar, i en els subscribers quina funció serà cridada cada cop que el topic rebi un nou missatge.

```
//[subscriber attributes]
ros::Subscriber dsf_subscriber_;
void dsf_callback(const std_msgs::Float64::ConstPtr& msg);

ros::Subscriber asf_subscriber_;
void asf_callback(const std_msgs::Float64::ConstPtr& msg);

ros::Subscriber state_subscriber_;
void state_callback(const std_msgs::Bool::ConstPtr& msg);

ros::Subscriber force_feedback_subscriber_;
void force_feedback_callback(const std_msgs::Bool::ConstPtr& msg);

ros::Subscriber torque_feedback_subscriber_;
void torque_feedback_callback(const std_msgs::Bool::ConstPtr& msg);
```

També és necessari declarar tot un seguit de missatges que permetin a l'aplicació interactuar amb el node de ROS.

Per una banda, es declaren els subscribers dsf_subscriber_ i asf_subscriber_, ambdós del tipus std_msgs/Float64. La funció d'aquests és que el node pugui llegir quin és valor del factor d'escala de les distàncies (distance scale factor) i del factor d'escala dels angles (angle scale factor) que l'aplicació permet modificar.

Per altra banda, es declaren tres subscribers anomenats state_subscriber_, force_feedback_subscriber_ i torque_feedback_subscriber_, que són de tipus std_msgs/Bool i permeten seleccionar des de l'aplicació gràfica el mode de treball del dispositiu hàptic amb el qual es vol treballar (amb retroalimentació de forces i/o moments o sense).

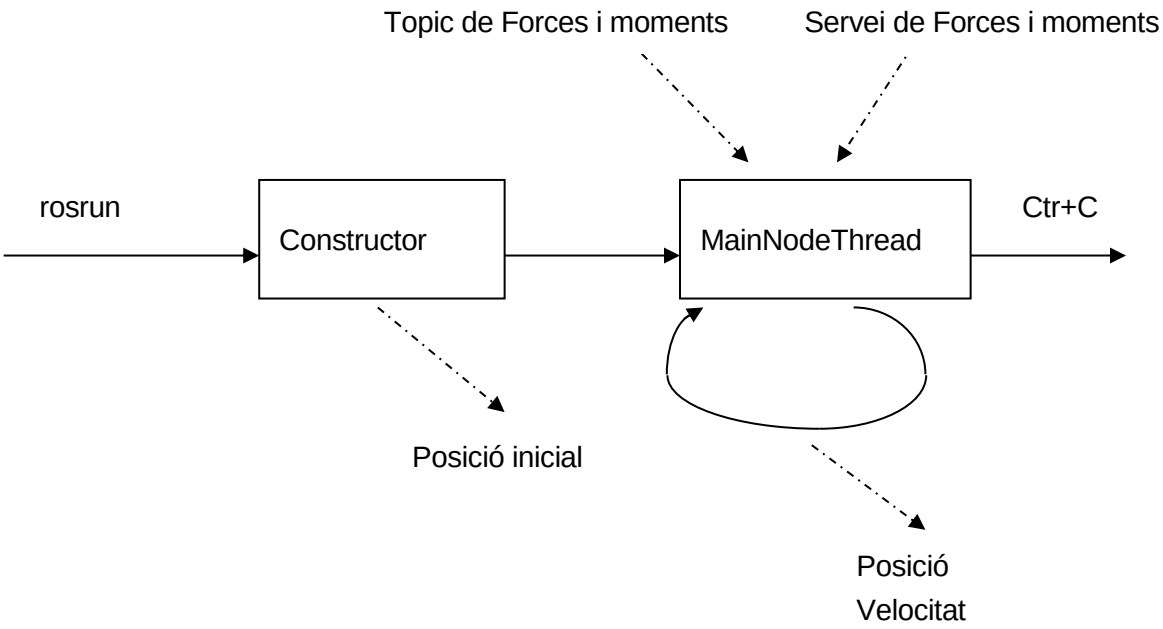
```
//[service attributes]
ros::ServiceServer setForceAndTorque_server_;
bool setForceAndTorqueCallback
(iri_delta_haptic_device::ForceAndTorque::Request &req,
iri_delta_haptic_device::ForceAndTorque::Response &res);
```

Per acabar, es declara un servei anomenat setForceAndTorque_server_, que permetrà aplicar una determinada força i un determinat moment al dispositiu hàptic. En aquest projecte, aquest servei serà utilitzat des de l'aplicació gràfica.

11.2.3.2.Implementació

És interessant analitzar més profundament la implementació del node de ROS (que es pot consultar a l'Annex C 3.6) del dispositiu hàptic, ja que té un funcionament particular.

Per fer l'explicació del codi més entenedora, s'utilitzarà el següent diagrama (Esquema 8):



Esquema 8. Esquema orientatiu sobre el funcionament del node de ROS del DHD.

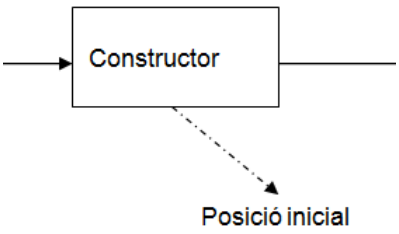
```
#include "delta_haptic_device_driver_node.h"
#include <math.h>
#include <unistd.h>

const double pi = 3.14159265;
```

Evidentment, és necessari incloure el header del node (delta_haptic_device_driver_node.h) per tal de poder fer referència a les funcions definides en ell.

També serà necessari incloure la llibreria "math" de C++ i definir la constant pi ja que serà necessari fer conversions d'angles en el codi.

Per últim, s'inclou també la llibreria unistd, que permet cridar a la funció sleep, la qual serà necessària, com s'explica més endavant.

En l'entorn de ROS, s'acostumen a definir els subscribers, publishers, serveis, ... en definitiva, tots els elements de ROS al constructor. El constructor analitzat a continuació representa aquesta part del diagrama:



Esquema 9. Part del diagrama original corresponent a la inicialització del node.

El codi és el següent:

```
DeltaHapticDeviceDriverNode::DeltaHapticDeviceDriverNode(ros::NodeHandle
&nh) :
  iri_base_driver::IriBaseNodeDriver<DeltaHapticDeviceDriver>(nh)
{

  ROS_INFO("Delta Haptic Device Node STARTED");

  //init class attributes if necessary
  this->loop_rate_ = 10;//in [Hz]
  driver_.setForceAndTorque(0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
  usleep(2500000);
  driver_.getPositionAndOrientation(this->pxi, this->pyi, this->pzi, this-
>oai, this->obi, this->oci);
  this->fx = this->fy = this->fz = this->tx = this->ty = this->tz = 0.0;
  this->dsf = this->asf = 1.0;
  this->pospos = false;
  this->force_feedback = this->torque_feedback = true;
```

Si s'analitza el constructor del node, es pot veure que en primer lloc utilitza un Log Message de ROS per comunicar que el node del dispositiu hàptic ha començat a actuar (Figura 27).

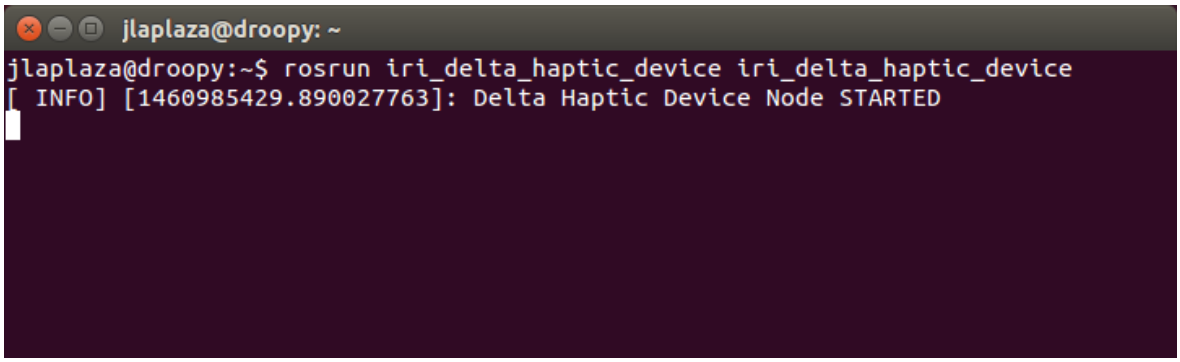


Figura 27. Missatge que es mostra per pantalla al iniciar el node del DHD.

Seguidament, es fixa un loop_rate que es tindrà en compte a la següent funció i es crida a la funció setForceAndTorque i se li dóna un valor igual a 0 a totes les forces i moments. Això es fa per tal que l'End Effector del dispositiu hàptic quedi en suspensió i sigui més fàcil per a la persona que l'utilitzarà moure'l, ja que no haurà de fer força per compensar el pes de l'End Effector.

Abans de veure el que es fa quan es crida a la funció usleep, és interessant veure la comanda de la línia just a sota, on es crida a la funció getPositionAndOrientation i rep com a paràmetres les variables pxi, pyi, pzi, oai, obi i oci. Aquestes variables representen les posicions i orientacions inicials de l'End Effector, i en cridar la funció aquestes variables prenen el valor de les posicions i orientacions que té l'End Effector en iniciar el node.

En haver aplicat unes forces i moments iguals a zero, l'End Effector es mou de manera automàtica gràcies als motors encarregats de transmetre moments fins a assolir una orientació on els tres angles oa, ob i oc són iguals a zero. Això podria no ser així si l'usuari manté l'End Effector en una posició arbitraria fent força amb la mà, però en general serà més habitual que l'usuari vulgui que els tres angles inicials siguin iguals a zero.

En principi, cridant la funció getPositionAndOrientation després de la funció setForceAndTorque, l'orientació que queda guardada com a inicial no és la posició amb els tres angles nuls, ja que l'End Effector triga un cert interval de temps en assolir aquesta posició, per tant, quan la funció getPositionAndOrientation és cridada, l'End Effector encara no ha assolit l'orientació amb els tres angles nuls.

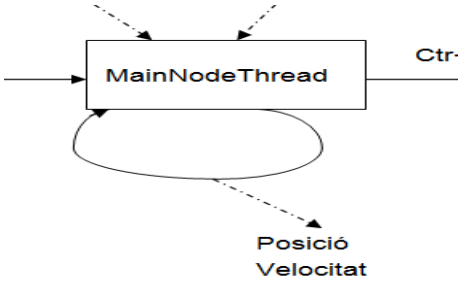
Per evitar aquest problema, entre la funció setForceAndTorque i getPositionAndOrientation es crida a la funció usleep, funció que fa que l'ordinador esperi 2,5 segons (2.500.000 microsegons) abans de continuar executant el codi, donant temps a l'End Effector per a assolir la posició desitjada abans d'executar la funció getPositionAndOrientation.

Després de fer això, cadascun dels objectes destinats a ser publishers, subscribers i serveis definits al header han de ser inicialitzats amb el seu topic corresponent. A continuació, es detalla un exemple de cada:

```
 // [init publishers]
this->Initial_Position_publisher_ = this-
>public_node_handle_.advertise<geometry_msgs::Pose>("Initial_Positi
on", 1);

  // [init subscribers]
  this->ft_data_subscriber_ = this-
>public_node_handle_.subscribe("/ftc_force_sensor_driver_node/ft_da
ta", 10, &DeltaHapticDeviceDriverNode::ft_data_callback, this);

  // [init services]
  this->setForceAndTorque_server_ = this-
>public_node_handle_.advertiseService("setForceAndTorque",
&DeltaHapticDeviceDriverNode::setForceAndTorqueCallback, this);
```

Com es pot veure, en el cas dels subscribers i els serveis és necessari especificar el nom de la funció que serà executada cada cop que es publiqui en el topic en qüestió (en el cas dels subscribers) o cada cop que el node rebi una request del servei en qüestió (en el cas dels serveis).

Un cop acabat el constructor, s'executa la següent funció, que representa aquesta part del diagrama (Esquema 10):



Esquema 10. Part del diagrama original que mostra el funcionament de la funció MainNodeThread.

La funció és la següent:

```
void DeltaHapticDeviceDriverNode::mainNodeThread(void)
```

Aquesta funció és la funció principal del node. Un node no deixa de ser un codi que s'executa i realitza una funció, i en l'àmbit de la robòtica és habitual que aquest codi hagi d'executar-se de manera reiterada durant un període de temps. Per fer-ho, és necessari executar alguna mena de loop.

Amb els algoritmes proporcionats per l'IRI, aquest loop es dóna en aquesta funció. L'arquitectura del codi és tal que fa que s'executi aquesta funció amb una freqüència fixada per l'usuari en el constructor (amb la variable loop_rate) indefinidament, fins que l'usuari prem la combinació de tecles Ctr+C.

El primer que fa aquesta funció és determinar en quin mode de treball es troba el dispositiu hàptic. Aquest mode de treball és el que es fixa mitjançant l'aplicació gràfica. Un cop determinat el mode de treball, aplica les forces i/o moments determinats.

Seguidament, s'actualitzen els valors de les variables internes de posicions, angles i orientacions amb els valors de l'End Effector en el moment d'execució de la funció.

Per últim, la funció publica a cada iteració als topics de posició i orientació, posició i orientacions inicials (les que tenia en executar el constructor) i velocitat amb els valors obtinguts durant la iteració.

El cas particular de la publicació dels angles d'orientació és especialment interesant. Els topics de tipus Pose i PoseStamped treballen amb angles en forma de quaternió, que és la representació d'angles utilitzant números complexos, mentre que les funcions del dispositiu hàptic utilitzen els seus angles d'Euler.

Per fer la conversió d'angles d'Euler a quaternió, s'ha utilitzat la següent transformació ([Eq. 1], extret de [7] i [8]):

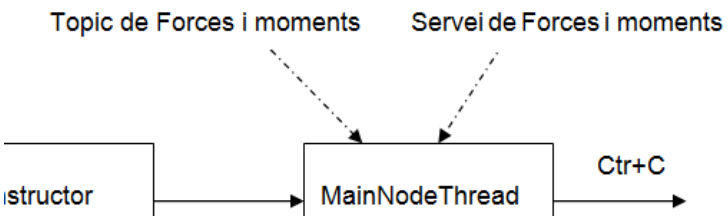
$$q = qz \cdot qy \cdot qx = \begin{bmatrix} \cos(\frac{\Psi}{2}) \\ 0 \\ 0 \\ sin(\frac{\Psi}{2}) \end{bmatrix} \cdot \begin{bmatrix} \cos(\frac{\Theta}{2}) \\ 0 \\ sin(\frac{\Theta}{2}) \\ 0 \end{bmatrix} \cdot \begin{bmatrix} \cos(\frac{\phi}{2}) \\ sin(\frac{\phi}{2}) \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\left(\frac{\phi}{2}\right) \cdot \cos\left(\frac{\Theta}{2}\right) \cdot \cos\left(\frac{\Psi}{2}\right) + sin\left(\frac{\phi}{2}\right) \cdot sin\left(\frac{\Theta}{2}\right) \cdot sin(\frac{\Psi}{2}) \\ \sin\left(\frac{\phi}{2}\right) \cdot \cos\left(\frac{\Theta}{2}\right) \cdot \cos\left(\frac{\Psi}{2}\right) - cos\left(\frac{\phi}{2}\right) \cdot sin\left(\frac{\Theta}{2}\right) \cdot sin(\frac{\Psi}{2}) \\ \cos\left(\frac{\phi}{2}\right) \cdot \sin\left(\frac{\Theta}{2}\right) \cdot \cos\left(\frac{\Psi}{2}\right) + sin\left(\frac{\phi}{2}\right) \cdot cos\left(\frac{\Theta}{2}\right) \cdot sin(\frac{\Psi}{2}) \\ \cos\left(\frac{\phi}{2}\right) \cdot \cos\left(\frac{\Theta}{2}\right) \cdot \sin\left(\frac{\Psi}{2}\right) - sin\left(\frac{\phi}{2}\right) \cdot sin\left(\frac{\Theta}{2}\right) \cdot cos(\frac{\Psi}{2}) \end{bmatrix}$$

[Eq. 1] Transformació d'angles d'Euler a quaternió. És el resultat de multiplicar successivament un quaternió unitari que representa una rotació entorn l'eix z,un altre a l'eix y i un altre a l'eix x.

Després d'aquesta funció, hi ha tot un seguit de funcions anomenades callbacks que permeten, com ja s'ha dit, actualitzar el valor de les variables afectades en els subscriptors i serveis.

L'espai que ocupen al diagrama és el següent (Esquema 11):



Esquema 11. Part del diagrama original on s'analitza la interacció del node amb un topic al qual està subscrit i amb un servei.

Com a exemple de callback d'un subscriptor, s'analitzarà el callback del topic de forces i moments:

```
void DeltaHapticDeviceDriverNode::ft_data_callback(const
geometry_msgs::WrenchStamped::ConstPtr& msg)
{
  this->fx = msg->wrench.force.z;
  this->fy = -0.5*(msg->wrench.force.x)+0.866*(msg->wrench.force.y);
  this->fz = -0.866(msg->wrench.force.x)-0.5(msg->wrench.force.y);
  this->tx = msg->wrench.torque.z;
  this->ty = -0.5*(msg->wrench.torque.x)+0.866*(msg->wrench.torque.y);
  this->tz = -0.866(msg->wrench.torque.x)-0.5(msg->wrench.torque.y);
}
```

Com es pot veure, aquesta funció el que fa és actualitzar les variables internes de força i moment amb els valors que han estat publicats al topic, tenint en compte les diferències entre els sistemes de referència entre els dos aparells (com s'ha vist al Capítol 10).

Per al correcte funcionament del servei, és necessari crear primer una carpeta nova al package iri_delta_haptic_device anomenada "srv", on hi haurà un fitxer anomenat "ForceAndTorque.srv". En aquest fitxer, s'han d'especificar quines variables d'entrada i de sortida tindrà el servei. Les variables d'entrada seran les que faran servir els requests, i les de sortida seràn les que faràn servir les replys. Aquestes variables estaran separades per tres guions. L'arxiu es pot trobar a Annex C 3.7.

Com a exemple de callback d'un servei, s'analitzarà el callback del servei setForceAndTorque:

```
bool
DeltaHapticDeviceDriverNode::setForceAndTorqueCallback(iri_delta_haptic_device::ForceA
ndTorque::Request &req, iri_delta_haptic_device::ForceAndTorque::Response &res)
{
  ROS_INFO("DeltaHapticDeviceDriverNode::setForceAndTorqueCallback: New Request
Received!");

  if (this->fx != req.new_fx || this->fy != req.new_fy || this->fz != req.new_fz ||
this->tx != req.new_tx || this->ty != req.new_ty || this->tz != req.new_tz)
  {
    this->fx = req.new_fx;
    this->fy = req.new_fy;
    this->fz = req.new_fz;
    this->tx = req.new_tx;
    this->ty = req.new_ty;
    this->tz = req.new_tz;

    res.success = true;
  }

  else
    res.success = false;

return true;
}
```

Quan s'activa el callback, en primer lloc s'envia un Log Message, i després, si les forces i moments requerits són diferents de les que ja estan aplicades, s'actualitzen les variables internes de forces i moments amb els valors de les forces i moments de la request.

11.2.3.3. Comprovacions

Un cop es va terminar el node, es va comprovar si enviava i rebia la informació correctament.

- Comprovació dels missatges de posició i orientació (Figura 28)

Per comprovar si el topic /iri_delta_haptic_device_driver_node/Position enviava correctament les posicions, es va utilitzar un plugin de rqt anomenat RViz, que mostra tres eixos de coordenades que es mouen dins d'un sistema de referència fix.

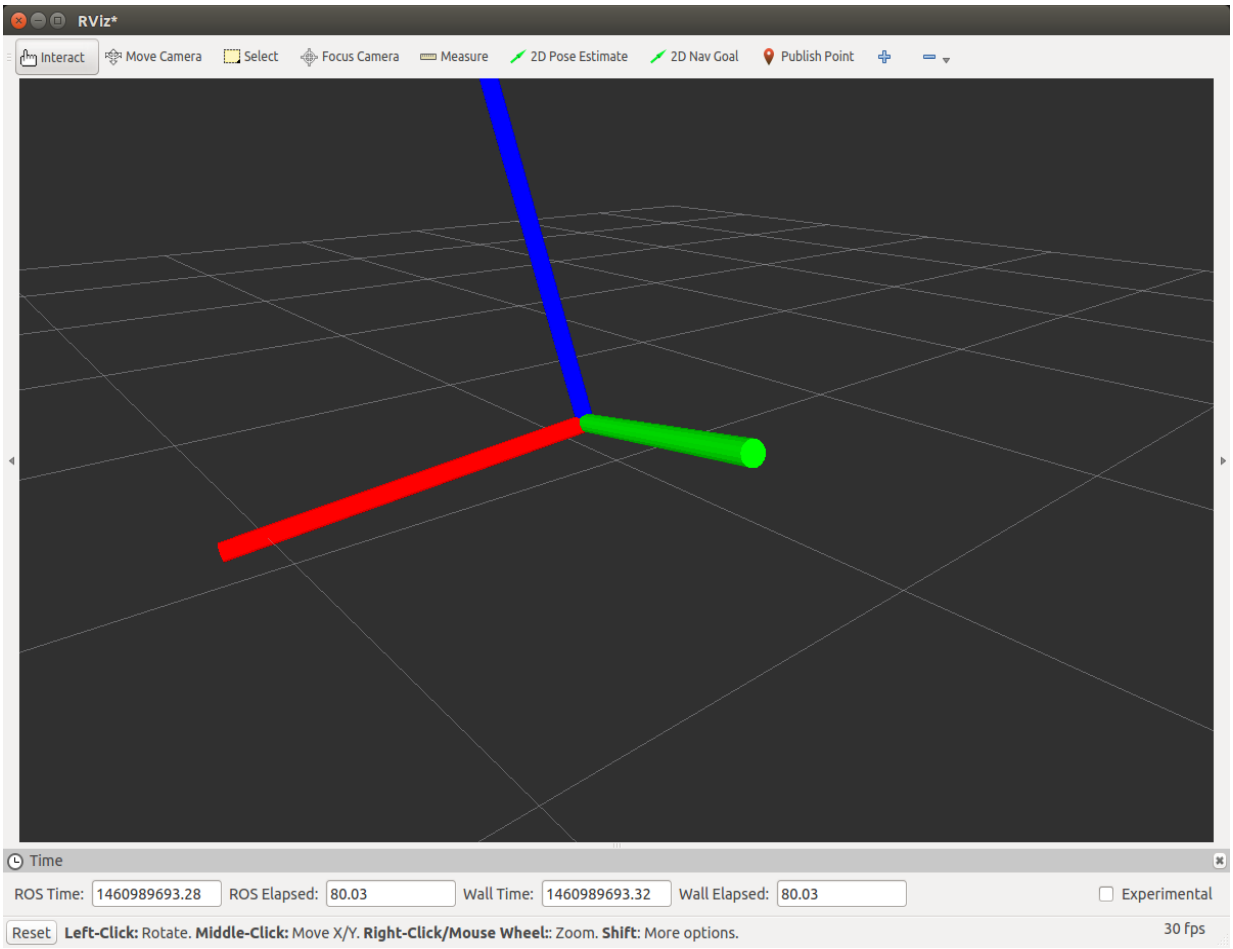


Figura 28. Representació del missatge de posició enviat pel DHD en el plugin de rqt RViz.

- Comprovació dels missatges de velocitat (Figura 29)

  Per comprovar si les velocitats s'enviaven correctament, es va utilitzar un plugin de rqt anomenat Plot que mostra un gràfic amb l'evolució temporal de cadascuna de les velocitats lineals del topic /iri_delta_haptic_device_driver_node/Velocity.

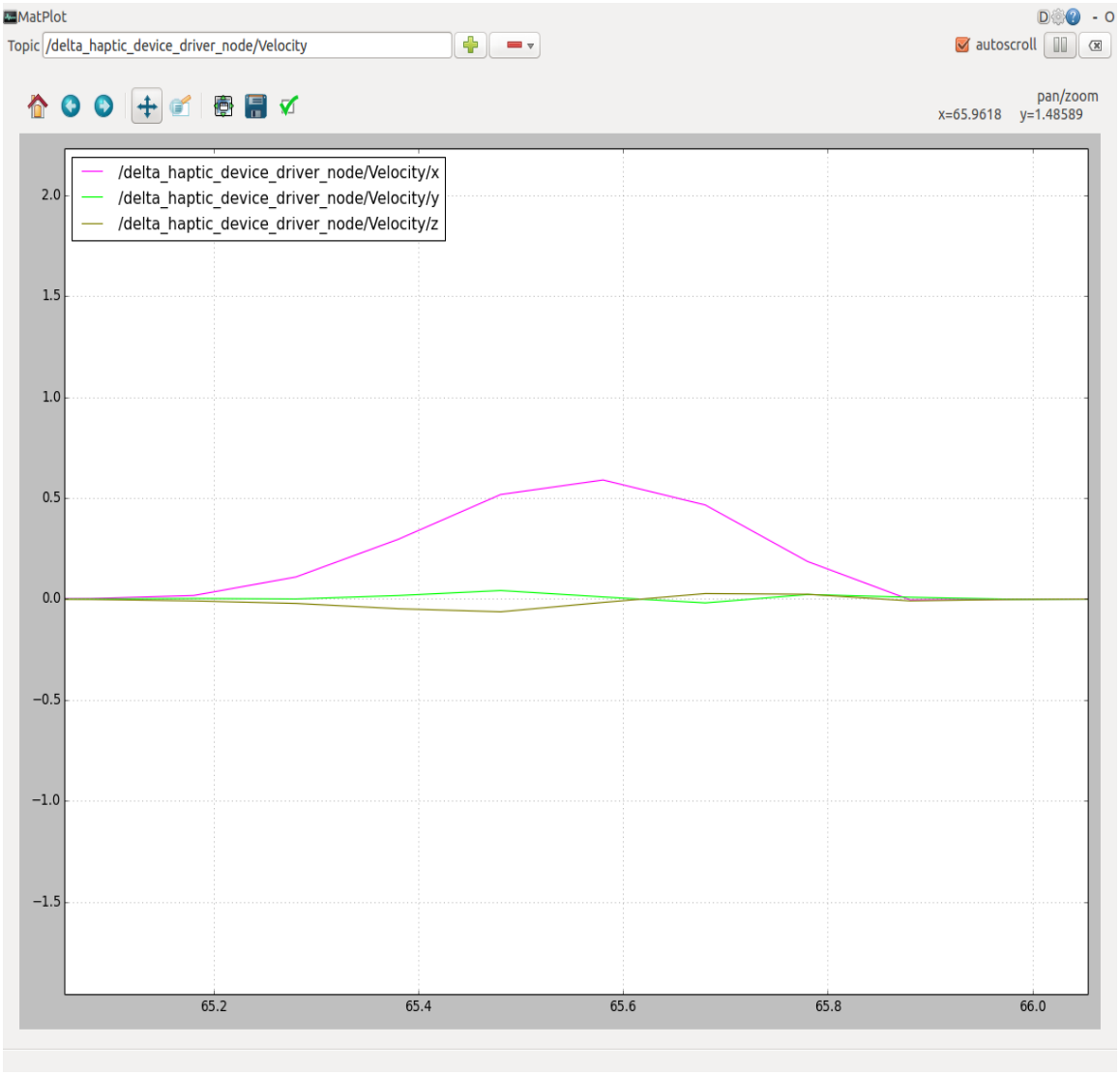


Figura 29. Representació de l'evolució temporal de les velocitats lineals de l'End Effector del DHD amb el plugin de rqt Plot quan és mogut en l'eix x.

- Comprovació de l'aplicació de les forces i moments

  A l'End Effector del dispositiu hàptic, s'hi poden aplicar forces de dues maneres diferents: mitjançant el topic /iri_ftc_force_sensor_driver_node/ft_data on publica el sensor de forces i moments o mitjançant el servei setForceAndTorque.

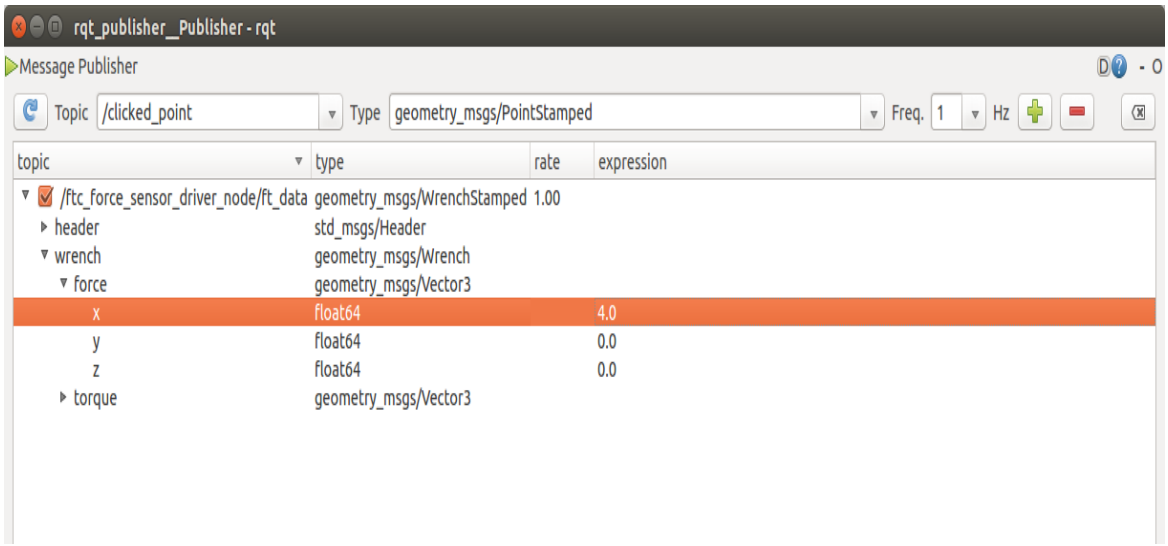


Figura 30. Figura del plugin de rqt Publisher quan s'envia una consigna de 4 N a l'eix x al DHD. Es comprova que el DHD reacciona a aquesta comanda.

Per comprovar si el topic funciona correctament, s'utilitza el plugin de rqt anomenat Publisher, que permet a l'usuari publicar la informació que vulgui a un topic, com es mostra a la Figura 30.

Per comprovar que el funcionament del servei sigui correcte, es va fer servir el plugin de rqt anomenat Service Caller, que permet a l'usuari fer una crida d'un determinat servei.

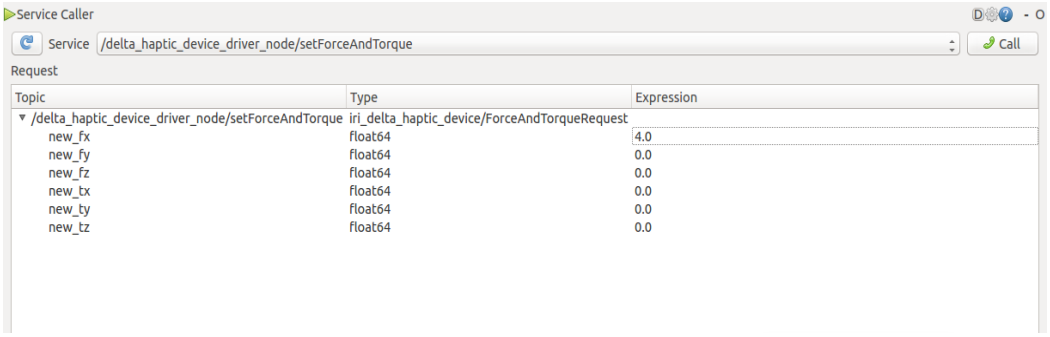


Figura 31. Figura del plugin de rqt Service Caller quan es fa una rquest de 4 N a l'eix x al DHD. Es comprova que el DHD reacciona a aquesta comanda.

Totes les comprovacions van resultar satisfactòries.

La integració final del node del dispositiu hàptic presenta l'estructura següent, vista amb el plugin Graph de rqt a la Figura 32:

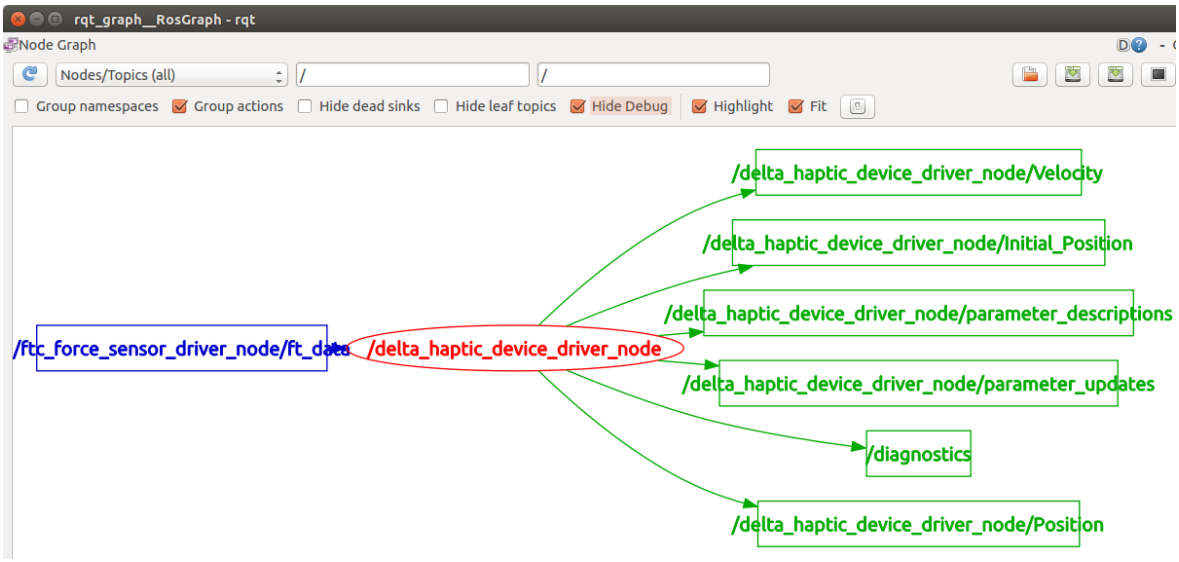


Figura 32. Representació gràcies al plugin rqt Graph del node del DHD.

Es pot veure, en vermell el node del dispositiu hàptic, en blau el topic al qual està subscrit (les forces i moments del sensor de forces) i en verd els topics als quals publica (principalment, la velocitat, la posició inicial i la posició arbitraria).

Aquest node, integrat amb els nodes del sensor de forces i el node del braç robòtic, presenta l'estructura següent, que es pot comparar amb la Figura 25 d'aquest capítol (Figura 33).

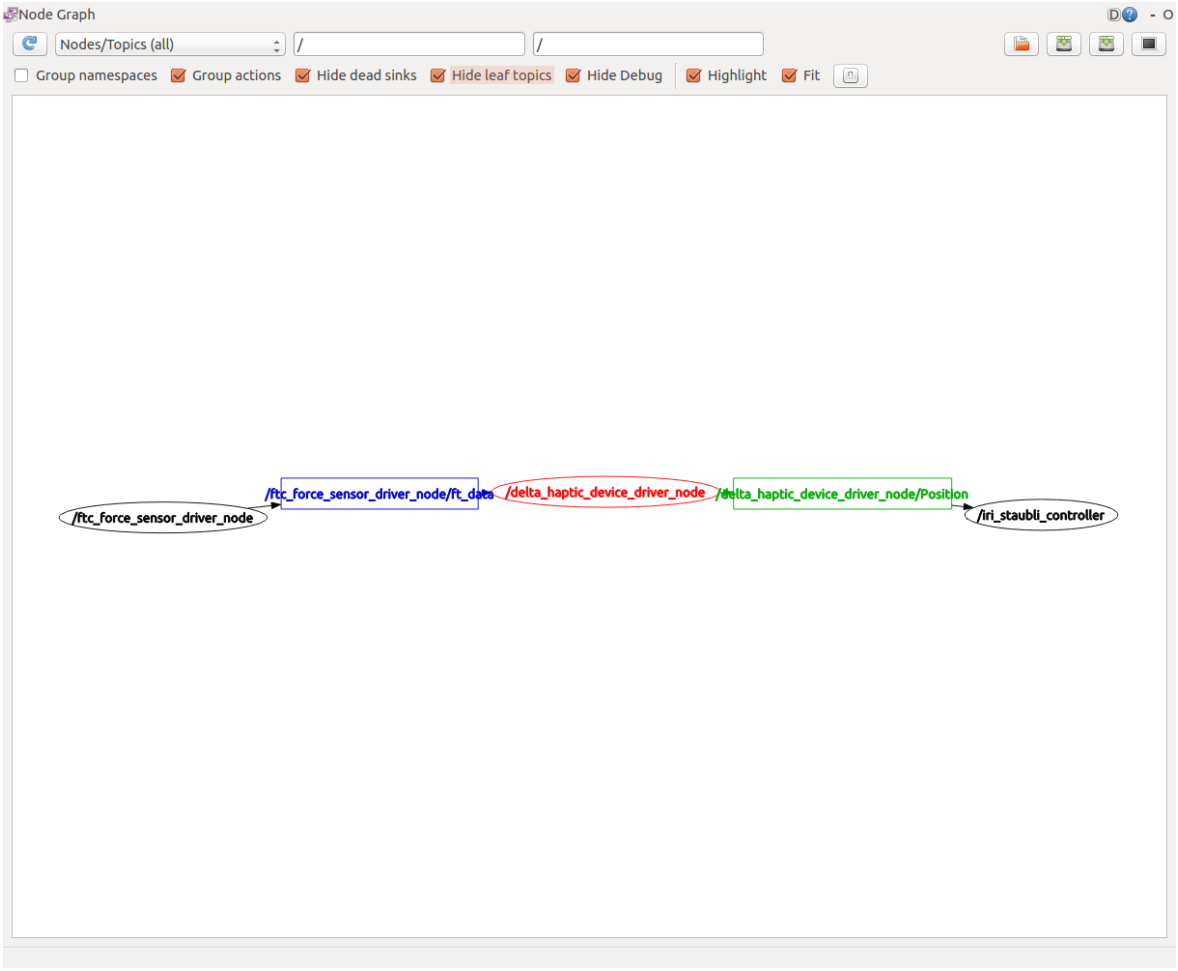


Figura 33. Representació amb el plugin de rqt Graph de tots els nodes que conformen el sistema de telecomunicació.

A la Figura es pot veure com el node del dispositiu hàptic completa el llaç de comunicació entre tots els dispositius.

12. Aplicació gràfica

12.1. Introducció

Per desenvolupar l'aplicació gràfica, es va decidir crear un plugin de rqt. El motiu d'aquesta decisió ha estat el fet que un dels requeriments de l'aplicació era que es pogués ampliar en el futur.

Com s'ha vist al capítol 7.5, els plugins de rqt es poden executar de manera simultània. D'aquesta manera, si es vol ampliar l'aplicació amb una funcionalitat que ja incorpora un altre plugin, només cal executar el plugin de l'aplicació i l'altre plugin alhora. Si es vol ampliar l'aplicació amb una funcionalitat que no ofereix cap altre plugin, es pot crear un plugin nou o ampliar el que s'ha fet en aquest projecte.

Per tant, seguint el tutorial de ROS per desenvolupar plugins de rqt [9], s'ha creat un plugin nou de rqt anomenat iri_gui_delta_haptic_device.

L'aspecte d'aquest plugin és el següent (Figura 34 i Figura 35):

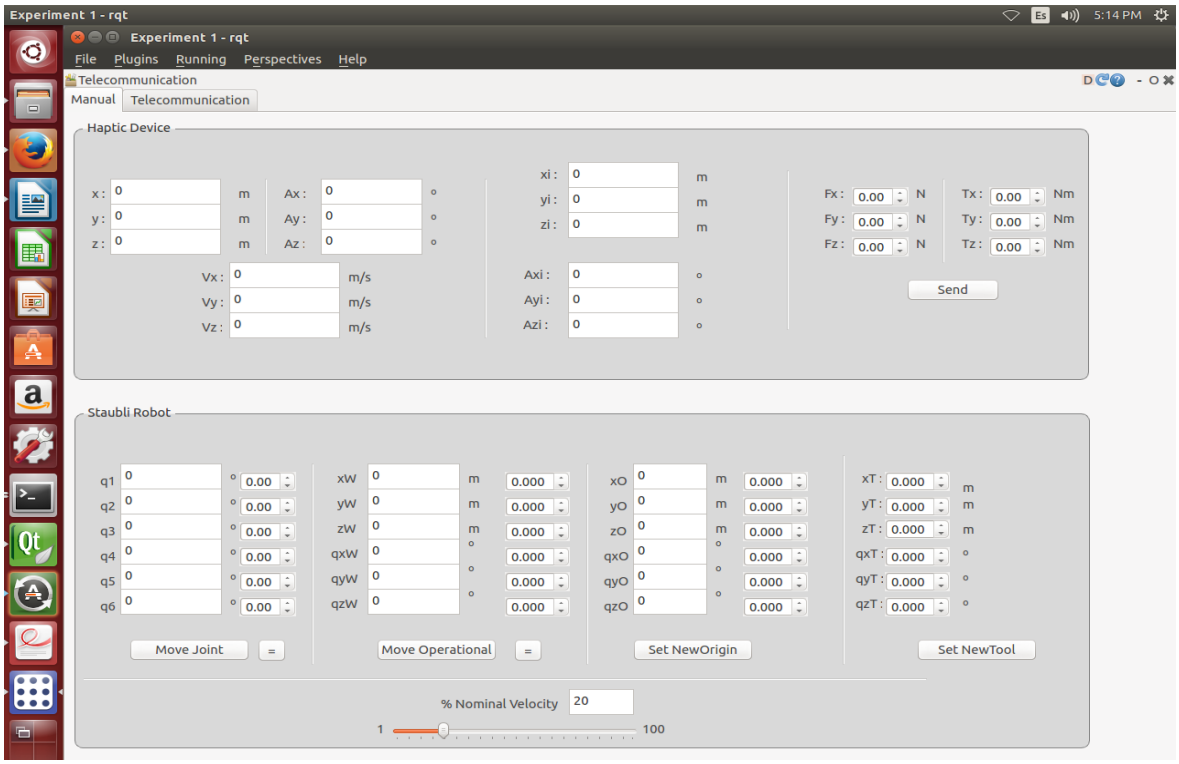


Figura 34. Disseny de la pestanya Manual de la aplicació desenvolupada al projecte.

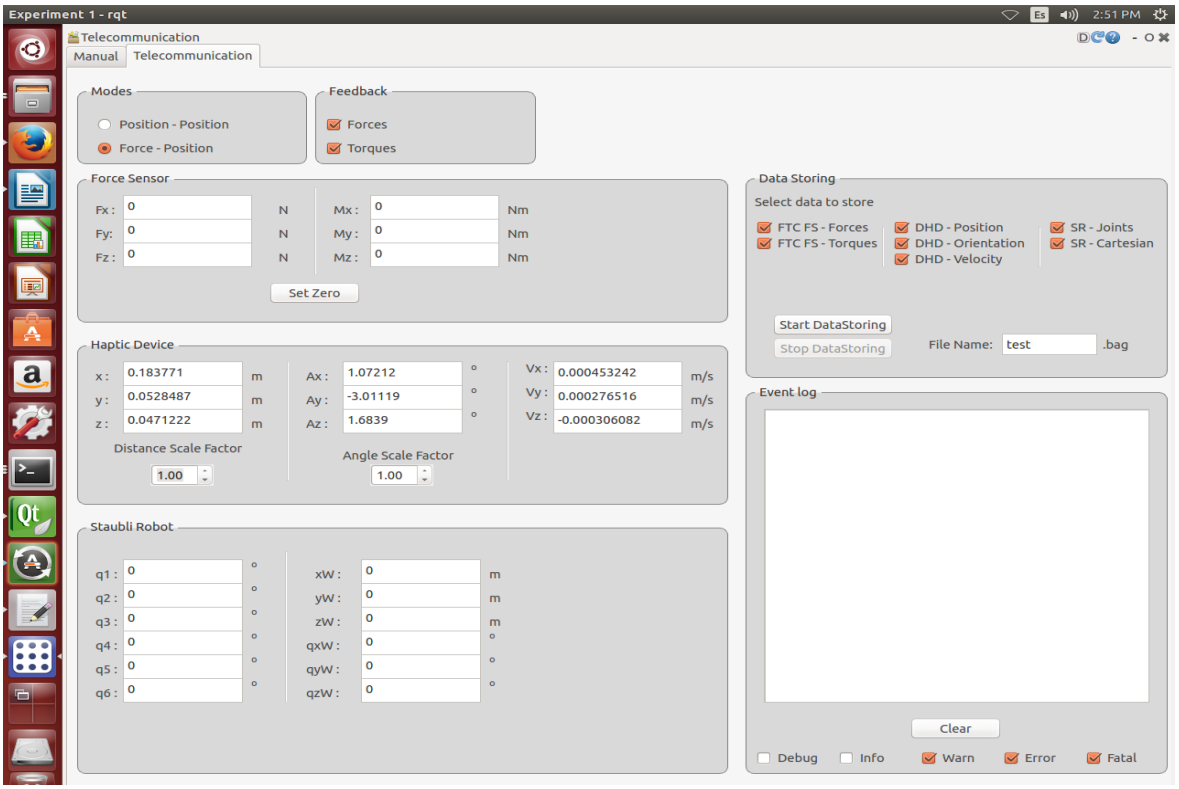


Figura 35. Disseny de la pestanya Manual de la aplicació desenvolupada al projecte.

Com es pot veure, l'aspecte visual d'aquesta aplicació és molt semblant a l'aspecte de l'aplicació antiga, però s'incorporen certs canvis interessants. Per veure aquests canvis, es detallen a continuació totes les opcions que incorpora aquesta aplicació:

- L'aplicació consta de dues pestanyes principals anomenades Manual i Telecommunication, igual que l'aplicació antiga (tot i que a l'aplicació antiga, el nom de la segona pestanya era Teleoperation).

- La pestanya Manual és pràcticament idèntica a l'aplicació antiga, amb l'excepció que ja no es mostren les dades del Sensor de Forces, ja que en ser una pestanya pensada per interactuar directament amb cada aparell es va considerar que ja que l'únic aspecte del sensor amb el que es pot interactuar és amb la funcionalitat del Set Zero, era millor deixar-ho només a la pestanya de Telecommunication.

  Pel que fa al dispositiu hàptic i al braç robòtic, no hi ha cap gran canvi, excepte el fet que el dispositiu hàptic mostra ara la posició inicial on s'ha arrancat el node.

- Pel que fa a la pestanya Telecommunication, els canvis són més significatius.

  Continuen havent-hi dues àrees de selecció de mode de treball (per controlar el feedback de forces i moments) i tres àrees on es mostren les dades de cadascun dels tres dispositius, amb la novetat de l'aparició al menú del dispositiu hàptic d'uns factors d'escala per les distàncies i per les orientacions configurables per l'usuari.

  Però més destacable encara és l'aparició de dues àrees noves, Data Storing i Event Log.

  L'àrea Data Storing permet a l'usuari en primer lloc seleccionar quines dades vol emmagatzemar, seleccionar un nom per al fitxer on s'emmagatzemarà aquesta informació i prémer el botó Start DataStoring per tal que tots els missatges seleccionats que s'estan publicant als topics s'escriguin en aquest fitxer fins que el botó Stop DataStoring s'acciona.

  En segon lloc, l'àrea Event Log dispossa d'un quadre de text on s'escriuen tots els missatges de la consola de ROS. Només s'escriuran els missatges que siguin del tipus que l'usuari hagi seleccionat a la part inferior d'aquesta zona. Els missatges de tipus DEBUG i INFO s'escriuen de color negre, els missatges de tipus WARN s'escriuen de color taronja, els missatges de tipus ERROR s'escriuen de color vermell fosc i els missatges de tipus FATAL s'escriuen de color vermell viu. També hi ha un botó Clear per esborrar tots els missatges que hi ha a la finestra.

Encara que alguns aspectes sembli que no hagin estat modificats, l'estructura interna de la informació i el seu tractament sí que ha canviat considerablement, a causa de la utilització de ROS.

12.2. Construcció

12.2.1. Package iri_gui_delta_haptic_device

Per crear aquest plugin, s'ha seguit el tutorial de ROS de creació d'un plugin de rqt [9]. Per aquest motiu, no s'utilitzen els scripts de construcció de l'IRI, ja que s'intenta ser el més fidel possible a aquest tutorial. Per seguir el tutorial, el primer pas és crear un package nou on guardar el plugin.

```
$ cd ~/iri-lab/iri_ws/src
$ catkin_create_pkg iri_gui_delta_haptic_device roscpp rqt_gui
rqt_gui_cpp message_generation rosbag std_msgs geometry_msgs
```

Seguidament, s'ha de preparar l'arxiu 'package.xml', incloent-hi sobretot a l'apartat dels exports la línia següent, ja que si no, el plugin no serà reconegut per rqt:

```
    <rqt_gui plugin="${prefix}/plugin.xml"/>
```

Després s'ha de crear l'arxiu 'plugin.xml', que es pot veure a l'Annex D 4.1 que conté informació referida concretament al plugin (com en quin submenú de rqt ha de mostrar-se el plugin) i omplir el document CmakeLists.txt, que es pot veure a Annex.

12.2.2. Creació de l'estructura visual de l'aplicació

El següent pas, és crear un arxiu .ui. Els arxius .ui són arxius que contenen informació sobre interficies gràfiques per usuaris. Contenen informació referent a la posició dels requadres que es mostren, el color d'aquests requadres, la forma, ...

Per crear aquest arxiu, s'ha fet servir una biblioteca de C++ anomenada Qt. Qt és una biblioteca multiplataforma àmpliament utilitzada per desenvolupar aplicacions amb interfícies gràfiques d'usuari. Qt es desenvolupa com a software lliure i de codi obert.

Concretament, per crear l'arxiu .ui de l'aplicació, s'ha fet servir un programa anomenat Qt Creator (Figura 36). Qt Creator és un editor on, de manera gràfica i senzilla, es poden disposar diferents elements de Qt, com ara barres lliscants, botons, quadres de text, … i generar el codi que els representa, com es veu a la Figura 37.

Creant un projecte de Qt Creator anomenat iri_gui_delta_haptic_device, se situen els diferents elements (anomenats Widgets) que conformen l'aplicació.

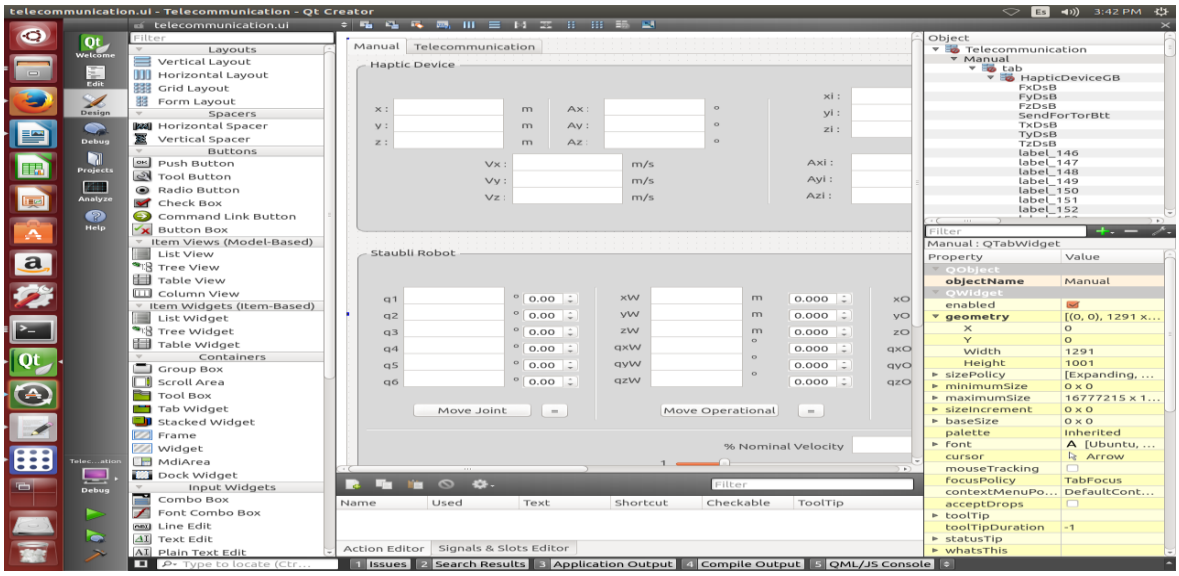


Figura 36. Editor Qt Creator.

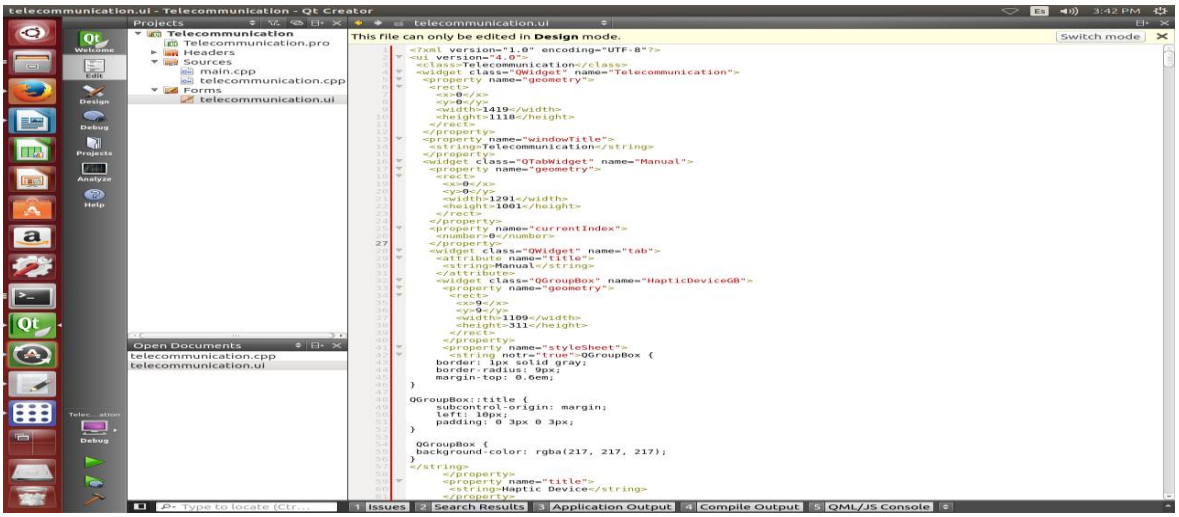


Figura 37. Aspecte del codi de l'arxiu .ui que defineix posicions i característiques dels elements de l'aplicació.

Un cop creat l'esquema visual de l'aplicació, Qt Creator permet prémer el botó Run (o Ctrl+R), de manera que es generin unes carpetes al directori home amb els fitxers Iri_gui_delta_haptic_device.ui, moc_iri_gui_delta_haptic_device.cpp i ui_iri_gui_delta_haptic_device.h. És necessari copiar aquests fitxers dins del package

iri_gui_delta_haptic_device, a una carpeta anomenada 'resource' que s'ha de crear.

12.2.3. Base teòrica del plugin

Una de les característiques principals de l'aplicació és que s'ha de programar per esdeveniments. És a dir, al contrari que els nodes de ROS on el codi es va executant amb una freqüència fixa, en aquest cas hi haurà parts del codi que només s'executaran quan succeeixi un determinat esdeveniment, com per exemple que s'accioni un botó o es modifiqui un valor d'algun Widget.

De fet, hi ha parts del codi dels nodes de ROS que sí que funcionen per esdeveniments: els subscribers i els serveis. Els subscribers només s'executen quan algú publica qualsevol cosa al topic, mentre que el servei només s'executa quan algú crida al servei.

Per fer aquest enllaç, Qt ofereix una eina anomenada "connect". Fent servir la comanda "connect", es pot fer que una funció sigui cridada només quan es compleix determinada situació. El mètode d'utilització és:

```
connect(Objecte1, SIGNAL(funció de l'objecte 1), Objecte 2, SLOT(funció de l'objecte 2))
```

on Objecte1 és l'objecte que envia un senyal, el senyal és una funció de l'objecte1 que ha sigut cridada, Objecte2 és l'objecte que rep el senyal que envia l'objecte 1 i, en rebre aquest senyal de l'objecte 1, executa la funció de l'objecte 2 especificada.

Per exemple, el botó Clear de l'àrea Event Log, és un widget que té una funció anomenada Clicked(), que actua com a senyal. Quan l'usuari clica aquest botó, aquest senyal és executat. Això comporta l'execució d'una funció del widget del quadre de text on es mostren els missatges de la consola anomenada "clear()", que esborra tot el que hi ha escrit al quadre.

Cal tenir en compte un aspecte molt important quan es treballa en Qt: els threads [10].

Els threads s'utilitzen per realitzar tasques en paral·lel, com els processos. La diferència entre els dos és subtil, però es pot entendre amb un exemple:

Es pot utilitzar un ordinador per treballar amb un editor de textos mentre s'escolta música alhora utilitzant un altre programa. Això són dos processos diferents i es coneix

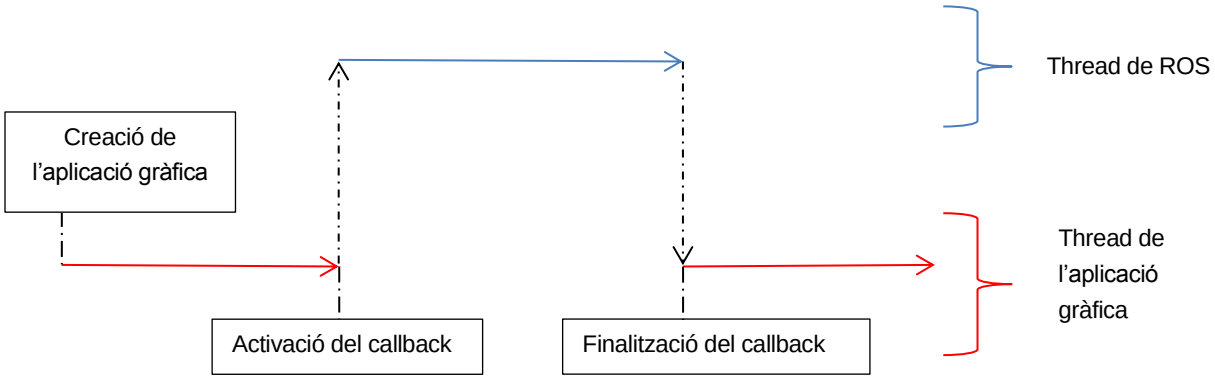
com multitasking.

Per altra banda, si s'analitza el comportament del programa reproductor de música, es pot veure que també hi ha operacions en paral·lel dins del mateix procés: mentre el reproductor de música envia senyals al driver de l'àudio, la interfície gràfica del programa s'està actualitzant constantment. Per això serveixen els threads, per realitzar diferents tasques dins del mateix procés.

Això afecta l'aplicació gràfica d'aquest projecte de manera crítica. El motiu és la interacció de l'aplicació amb ROS.

Quan s'inicia l'aplicació, es crea un objecte que conté tota la informació sobre l'aplicació en un thread. Més endavant, es poden crear altres threads, però està rotundament prohibit modificar un objecte que ha sigut creat en un thread des d'un altre thread.

Això afecte, per exemple, a l'hora de mostrar les dades que publiquen els altres nodes. El procediment és crear un subscriber al codi de l'aplicació i que cada cop que algú publiqui alguna informació en aquest topic, el callback d'aquest subscriber sigui cridat. El problema succeeix ja que, com es mostra a l'Esquema 12, aquest callback s'executa en un altre thread, different del thread on s'ha creat l'aplicació.



Esquema 12. Funcionament dels threads quan s'executa un callback.

Per tant, si es vol modificar la informació que mostra l'aplicació quan un callback és cridat, no es pot modificar aquesta informació des de dins del propi callback. La manera canònica d'evitar aquest problema, és crear un signal i fer que aquest signal sigui cridat en el callback. Aquest signal ha d'enllaçar-se amb una funció del thread principal que s'encarregui d'actualitzar les dades mostrades per l'aplicació. A més, quan els signals i

slots són a threads diferents, és necessari modificar lleugerament l'estructura del "connect":

```
connect(Objecte1, SIGNAL(funció de l'objecte 1), Objecte 2, SLOT(funció de l'objecte 2), Qt::BlockingQueuedConnection)
```

Aquest paràmetre al final del "connect" s'encarrega de fer que, quan comenci el segon thread, no es pugui sortir d'aquest thread fins que s'executi fins al final.

12.2.4. Funcionament del codi

En crear-se l'objecte que conté l'aplicació, s'inicialitzen tots els connects necessaris per al funcionament de l'aplicació. També es declaren tots els publishers, subscribers i clients necessaris, que són:

- **Publishers**
  - dsf_publisher_: Publisher del factor d'escala de les distàncies. Aquest publisher és de tipus std_msgs/Float64, és un número que rebrà el dispositiu hàptic i el multiplicarà per les distàncies que publica.
  - asf_publisher_: Publisher del factor d'escala dels angles. Aquest publisher és de tipus std_msgs/Float64, és un número que rebrà el dispositiu hàptic i el multiplicarà pels angles de les orientacions que publica.
  - state_publisher_: Publisher que s'encarrega de gestionar l'àrea "Mode" de l'aplicació. Aquest publisher és de tipus std_msgs/Bool, és una variable booleana que, quan el seu valor és false, fa que el sistema treballi en el mode "Position-Position", i quan el seu valor és true, fa que el sistema treballi en el mode "Force-Position".
  - force_feedback_publisher_: Publisher que s'encarrega de controlar la realimentació de forces. Aquest publisher és de tipus std_msgs/Bool, és una variable booleana que, quan el seu valor és false, desactiva la realimentació de forces, i quan el seu valor és true permet aquesta realimentació.

- torque_feedback_publisher_: Publisher que s'encarrega de controlar la realimentació de moments. Aquest publisher és de tipus std_msgs/Bool, és una variable booleana que, quan el seu valor és false, desactiva la realimentació de moments, i quan el seu valor és true permet aquesta realimentació.

- **Subscribers**

  - dhd_position_subscriber_: Subscriber que s'encarrega de llegir les posicions que envia el dispositiu hàptic. Aquest subscriber és de tipus geometry_msgs/PoseStamped.

  - dhd_initial_position_subscriber_: Subscriber que s'encarrega de llegir la posició inicial que envia el dispositiu hàptic. Aquest subscriber és de tipus geometry_msgs/Pose.

  - dhd_velocity_subscriber_: Subscriber que s'encarrega de llegir la velocitat lineal de l'End Effector del dispositiu hàptic. Aquest subscriber és de tipus geometry_msgs/Vector3.

  - ft_data_subscriber_: Subscriber que s'encarrega de llegir les forces i moments que capta el sensor de forces. Aquest subscriber és de tipus geometry_msgs/WrenchStamped.

  - staubli_joints_subscriber_: Subscriber que s'encarrega de llegir el valor de les articulacions del braç robòtic. Aquest subscriber és de tipus sensor_msgs/JointState.

  - staubli_tcp_subscriber_: Subscriber que s'encarrega de llegir la posició a l'espair cartesià del TCP del braç robòtic. Aquest subscriber és de tipus geometry_msgs/PoseStamped.

  - log_subscriber_: Subscriber que s'encarrega de llegir tots els missatges de la consola de ROS. Aquest subscriber és de tipus rosgraph_msgs/Log.

- **Clients**
    - setForceAndTorque_client_: Client que s'encarrega d'aplicar la força i moment designada per l'usuari a l'end effector del dispositiu hàptic.
    - setZero_client_: Client que s'encarrega de designar com a 0 el valor de les forces i moments del moment en què és cridat el client.
    - move_in_joints_client_: Client que s'encarrega de moure el braç robòtic en l'espai articular.
    - move_in_cart_client_: client que s'encarrega de moure el TCP del braç robòtic en l'espai cartesià.

Quan s'executa el plugin, aquest mostra les dades que es publiquen als topics als quals està subscrit. Quan l'usuari acciona el botó Start DataStoring, s'obre un bag amb rosbag.

Cada cop que s'executa un callback d'un dels subscribers emmagatzemables, en primer lloc l'aplicació actualitza el valor de les variables internes de posició, força, moment, velocitat, etc.

En segon lloc, el codi comprova si s'ha premut el botó Start DataStoring i, en cas afirmatiu, comprova si l'usuari ha seleccionat que vol guardar en el fitxer especificat les dades d'aquest topic. En cas afirmatiu, s'escriu al fitxer la informació de cada callback repetidament fins que l'usuari prem el botó Stop DataStoring, que tanca el bag de rosbag.

Finalment, cada callback emet un senyal que activa una funció que fa un "update" de la informació que es mostra a la interficie gràfica.

En el cas concret dels missatges de posició (geometry_msgs/Pose i geometry_msgs/PoseStamped) els angles que rep l'aplicació estan expressats en forma de quaternió.

Per passar de quaternió a angle, s'ha utilitzat la següent transformació ([Eq. 2], extret de [7]):

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}\arctan\dfrac{2\,(q_0q_1+q_2q_3)}{1-2(q_1^2+q_2^2)}\\ \arcsin(2(q_0q_2-q_3q_1)\\ \arctan\dfrac{2(q_0q_3+q_1q_2)}{1-2(q_2^2+q_3^2)}\end{bmatrix}$$

[Eq. 2] Transformació per passar de quaternió a angles d'Euler.

No obstant això, en llenguatge de programació, la funció arctan només retorna resultats compresos entre –π/2 i π/2, per tant és necessari utilitzar en el codi la transformació definida a la [Eq. 3] (extret de [11]):

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}\text{atan2}(2\,(q_0q_1+q_2q_3), 1-2(q_1^2+q_2^2))\\ \arcsin(2(q_0q_2-q_3q_1)\\ \text{atan2}(2(q_0q_3+q_1q_2), 1-2(q_2^2+q_3^2))\end{bmatrix}$$

[Eq. 3] Transformació utilitzada dintre del codi de programació per poder representar angles que no estiguin dintre del rang ±π/2.

En aquest cas, s'utilitza la funció "atan2", que forma part de la llibreria "math" de C++. Aquesta funció rep dos arguments, el numerador i el denominador, i té en compte el signe de cadascun per determinar l'angle resultant.

En el cas de la zona de l'Event Log, quan s'activa el callback es determina de quin tipus és el missatge que s'ha enviat i si l'usuari vol mostrar aquest missatge en el quadre de text. En cas afirmatiu, s'escriu el missatge amb el color corresponent.

Es pot trobar el codi complet a l'Annex D 4.2 i 4.3.

En la Figura 38 es mostra un diagrama de grafs de tot el sistema amb l'aplicació gràfica. Hi apareix destacat en vermell el node de l'aplicació gràfica, en verd tots els topics als quals aquesta hi publica i en blau tots els topics als quals aquesta està subscrita (principalment, per monitorar les magnituds de posició, força, angle de les articulacions, ... que cada dispositiu comparteix).

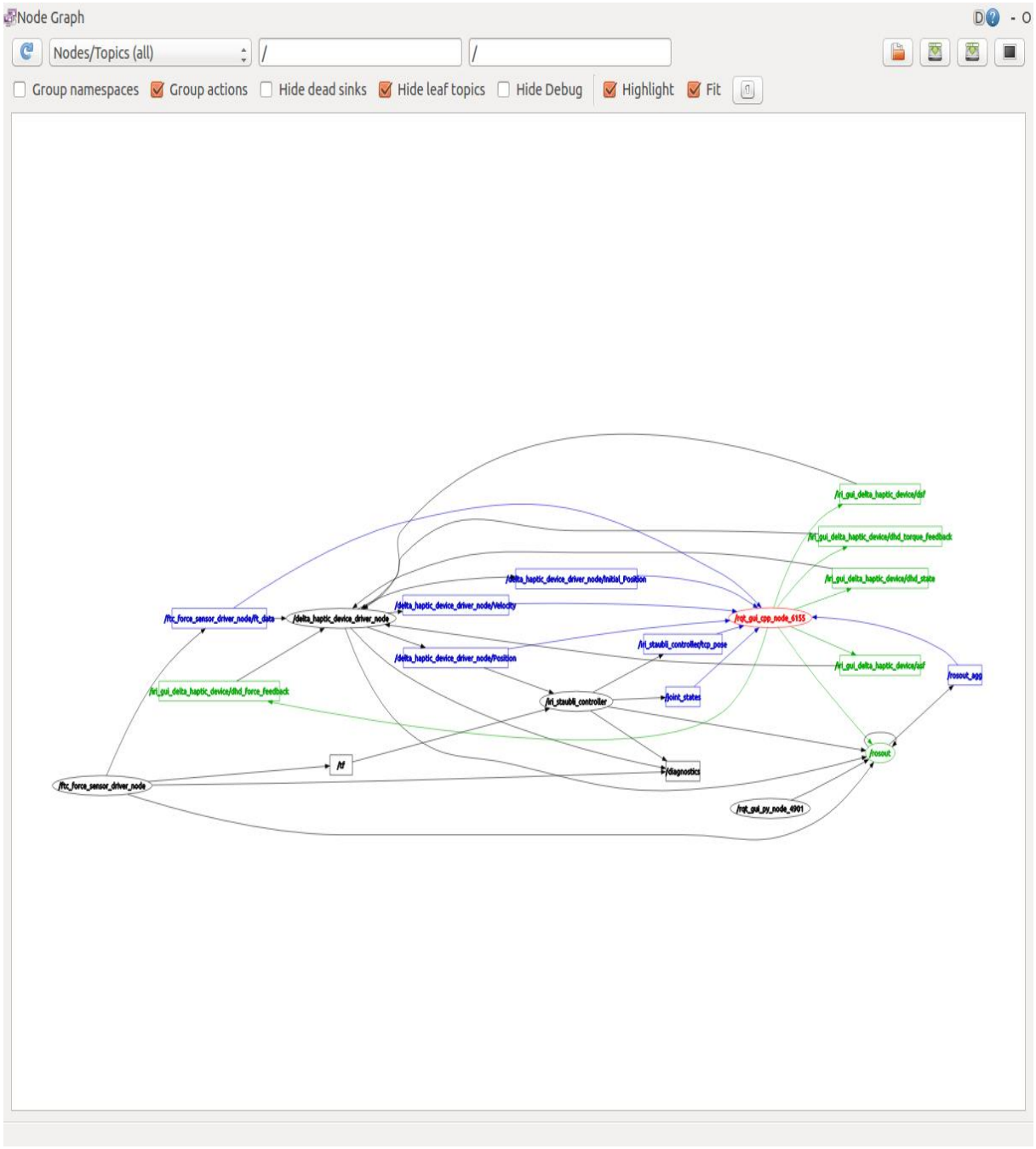


Figura 38. Diagrama del plugin de rqt Graph on es mostra les connexions de tot el sistema.

13. Treball futur

Es proposen tot una sèrie de millores enfocades a millorar o acabar d'afinar alguns aspectes del sistema disenyat:

- Quan succeeix qualsevol excepció en el node del dispositiu hàptic, aquest les gestiona a baix nivell i mostra pel terminal quin ha sigut l'error que ha succeït, però no envia cap missatge amb aquesta informació fent servir missatges de consola de ROS. Seria una bona idea implementar aquesta característica amb missatges de nivell WARN, per exemple.
- Seria interessant que l'aplicació pogués seleccionar el nom dels topics als quals ha d'estar subscrita, ja que en el futur el nom d'alguns dels topics podria variar.
- El sensor de forces i moments necessita alguna reparació, ja que el sistema electrònic dóna problemes de tant en tant.
- S'ha de millorar el sistema de gestió de posicions no assolibles pel braç robòtic, ja que en el sistema actual, quan el braç és enviat a alguna posició no assolible el node del braç robòtic deixa de funcionar correctament i es queda parat, fent necessàri tornar a executar el node.
- Un aspecte interessant de l'aplicació prèviament existent que no s'ha inclós a la nova aplicació és el fet de poder gestionar quins dispositius es volen fer servir alhora. Seria interesant tornar a implementar aquesta característica de cara a la comprovació del correcte funcionament de cada element de manera aïllada.
- És necessari aprofundir en la naturalesa del retard de comunicació entre el dispositiu hàptic i el braç robòtic, ja que es vital que el braç robòtic executi les consignes de posició i orientació del DHD de manera ràpida i fluida. Actualment s'està treballant en aquest aspecte al laboratori de desenvolupament.

Conclusions

Durant el projecte, s'han assolit diferents objectius:

Gràcies a la facilitat de separació de la informació que proporciona ROS, s'ha pogut utilitzar el sensor de forces i el braç robòtic mirant únicament el codi dels seus nodes, sense la necessitat d'entrar a analitzar els drivers d'aquests dispositius, fet que hagués sigut complexe i costós.

Aquesta anàlisi dels nodes, ha proporcionat una visió clara de com encaixar el node que s'havia de crear del dispositiu hàptic en l'estructura ja existent de ROS.

S'ha desenvolupat un node per al dispositiu hàptic que és capaç d'enviar satisfactòriament les seves dades de posició, orientació i velocitat i de rebre dades de força i moment.

També s'ha desenvolupat una interfície gràfica d'usuari capaç de mostrar les dades que els elements comparteixen entre si i d'interactuar satisfactòriament amb ells.

Juntament amb aquesta memòria, s'inclou un manual d'usuari del sistema (Annex E 5.1 i 5.2).

El sistema ha estat implementat correctament i el seu funcionament es satisfactòri, tenint present que calen certes millores al node del braç robòtic i certes reparacions a nivell de hardware del sensor de forces.

Agraïments

M'agradaria agrair profundament a en Pablo Jiménez i a en Guillem Alenyà per fer-me l'entrada a l'IRI el més fàcil possible i que sempre tinguessin un moment per ajudar-me.

També m'agradaria agrair de tot cor a en Sergi Foix, sense el qual el resultat final d'aquest projecte hagués estat impossible.

De la mateixa manera, en Toni Gabás i l'Alejandro Suárez han sigut peces fonamentals per ajudar-me durant el llarg procés d'aprenentatge.

Per últim, gràcies als meus pares i a la Luana, per donar-me el seu suport incondicional.

Gràcies a tots.

14. Bibliografia

[1] S. I. AG, «Industrial automation in Textile, Connectors, Robotics - Stäubli,» Stäubli, 23 11 2003. [En línia]. Available: http://www.staubli.com/. [Últim accés: 19 04 2016].

[2] S. Grange, «Force Dimension Home,» Force Dimension, 21 04 2016. [En línia]. Available: http://www.forcedimension.com/. [Últim accés: 10 07 2007].

[3] M. Heimgreiter, «Schunk Home,» Schunk, 07 01 2002. [En línia]. Available: http://de.schunk.com/de_de/startseite/#/. [Últim accés: 25 04 2016].

[4] L. Rozo, «Staubli RX-60 teleoperado a través de un DHD con retroalimentación de fuerza,» Institut de Robòtica i Informàtica Industrial, Barcelona, 2012.

[5] D. Thomas, «ROS Tutorials,» ROS, 09 10 1998. [En línia]. Available: http://wiki.ros.org/ROS/Tutorials. [Últim accés: 22 06 2016].

[6] M. Roure, «Labrobotica Tutorial Hello World,» Institut de Robòtica i Informàtica Industrial, 29 03 2001. [En línia]. Available: https://wiki.iri.upc.edu/index.php/LabRobotica_Tutorial_Hello_World. [Últim accés: 20 06 2015].

[7] D. M. Henderson, «Euler Angles, Quaternions and Transformation Matrices,» NASA, Houston, 1977.

[8] K. Joy, «Quaternions,» 19 12 2014. [En línia]. Available: https://www.youtube.com/watch?v=mHVwd8gYLnI. [Últim accés: 22 04 2016].

[9] L. Walter, «Create your rqt plugin package,» ROS, 06 08 2014. [En línia]. Available: http://wiki.ros.org/rqt/Tutorials/Create%20your%20new%20rqt%20plugin. [Últim accés: 29 02 2016].

[10] «Threading Basics,» Qt, 10 11 2011. [En línia]. Available: http://doc.qt.io/qt-5/thread-basics.html. [Últim accés: 04 03 2016].

[11] J. Diebel, «Representing Attitude: Euler Angles, Unit Quaternions, and Rotation,» Stanford University, Stanford, 2006.

[12] «C++ Language,» C++, 14 12 2001. [En línia]. Available: http://www.cplusplus.com/doc/tutorial/. [Últim accés: 10 07 2015].

[13] M. Mahoney, «How to program in C++,» Florida Institute of Technology, 13 01 1989. [En línia]. Available: https://cs.fit.edu/~mmahoney/cse2050/how2cpp.html. [Últim accés: 13 07 2015].

[14] S. R. Aude Bolopion, «A Review of Haptic Feedback Teleoperation Systems for,» Institute of Electrical and Electronics Engineers, Paris, 2013.

[15] C. O. a. M. Sitti, Visual Servoing-Based autonomous 2-D manipulation of microparticles using nanoprobe, Pittsburg: IEEE , 2007.

[16] C. O. a. M. Sitti, «Towards micro-assembly of hybrid MOEMS components on reconfigurables silicon free-space micro-optical bench,» *Journal of Micromechanics and Microengineering,* vol. 20, p. 20, 2010.

[17] B. N. a. B. V. Y. Zhou, «Fusing force and vision feedback for micromanipulation,» de *IEEE International Conference on Robotics and Automation*, Chicago, 1998.

[18] T. N. P. a. F. R. Dauterive, «A comparison of total Laparoscopic Hysterectomy to Robotically Assited Histerectomy: Surgical Outcomes in a Community Practice,» *Journal of Minimally Invasive Gynecology,* vol. 15, pp. 286-91, 2008.