Learning and Vision Robotics Mobile Group

DEMO TRACKING

Elena Muñoz España
emunoz@iri.upc.es

1. Presentación

El software tracking realiza el seguimiento de rostros humanos. Detecta que región de la imagen contiene una cara, busca la zona que más se parece a un histograma de referencia previamente definido y dibuja una elipse, a partir de ahí se inicia el proceso de seguimiento, aplica un algoritmo basado en la búsqueda en un área probable y comparación del histograma de referencia eligiendo la zona en la cual hay mayor concordancia.

Se partió de los siguientes programas soporte:

Trackerhist: Clases para realizar el seguimiento de caras. CColor_Histogram, CColorTracker, CHistogram, CList, COutline, CPixelList, CPixelListArray y CSearchStrategy. Desarrollado por Xavier Castell.

SegmentaWin4: Software que realiza el seguimiento de caras, se retomo el código para control del pant tilt y la clase CCamara. Desarrollado por Adrià Tarrida.

Stereo4Anna2: Se tomaron las clases para manejo de las imágenes y captura de ellas mediante las librerías MIL. CImage_unsigned_char, CImage_double, CImage_long y se incluyo matrix.h. Desarrollado por Francesc Moreno.

Ariaproj y Directmotion: Demos incluidos en la documentación de Aria.

Sintetizador: Librería que permite incluir los mensajes audibles.

La versión actual de tracking funciona en un computador externo y requiere de las siguientes conexiones para su operación:

- Conectar el pantilt al COM1 del PC
- Conectar el COM2 del PC al puerto serie del ordenador del robot
- El control del IRIS también esta configurado para el COM2 del PC

2. Descripción de la Interfaz Software

Al ejecutar el programa **tracking**, se inicia con una pantalla de presentación (Figura 1) la cual se visualiza durante dos segundos y se emite un mensaje audible de bienvenida.

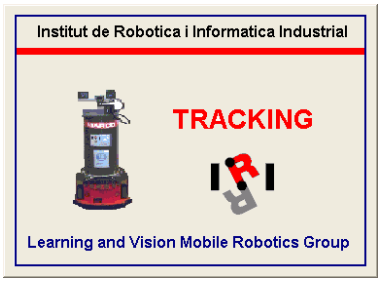


Figura 1. Pantalla de presentación

Después de la pantalla de inicial, se presenta la interfaz de usuario del programa (Figura 2). Consta de un menú de opciones una barra de herramientas una barra de estado y de dos cuadros de imagen (para la imagen capturada y la procesada).

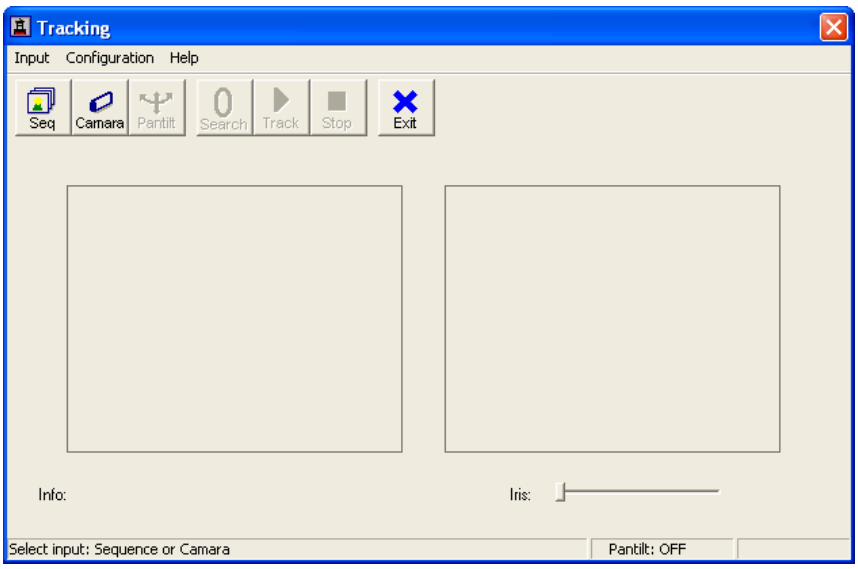


Figura 2. Interfaz de usuario programa tracking

La barra de herramientas posee las siguientes opciones:

Figura 3. Barra de herramientas

Seq: Presenta en el cuadro de imagen de la izquierda la primera imagen de la secuencia definida por defecto. (Se puede cambiar mediante el menú Configuration/Sequence).

Camara: Visualiza la señal proveniente de la cámara.

Pantilt: Conecta o desconecta el *Pantilt* de las cámaras. En la barra de estado ubicada en la parte inferior de la ventana se presenta un mensaje según sea su estado. (Pantilt: ON o Pantilt:OFF)

Search: Inicia la búsqueda de una cara comparándola con el histograma definido previamente en el archivo *Histogram.bin*. Requiere que el pantilt este conectado. Una vez encontrada inicia automáticamente la operación de seguimiento.

Track: Si no se utiliza la opción de Search, se puede indicar mediante un clic con el mouse sobre la imagen la ubicación de la cara y mediante este botón se inicia el seguimiento de la cara a partir de la posición indicada.

Stop: Detiene la operación de búsqueda o de seguimiento iniciadas por los botones *Search* o *Track*.

Exit: Termina la ejecución del programa.

Los botones de la barra de herramientas se habilitan/deshabilitan dependiendo de las acciones que estén permitidas realizar en el momento.

El menú presenta las siguientes opciones:

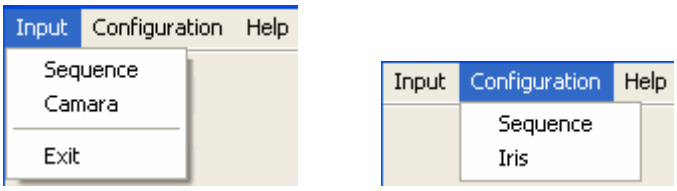


Figura 4. Menú de opciones

Input: Permite seleccionar el modo de operación del programa.
Sequence: Carga por defecto las imágenes que están en el directorio Sec\More1\. Camara: Visualiza la señal proveniente de la cámara en el cuadro de imagen izquierdo.
Exit: Termina la ejecución del programa

Configuration: Permite la configuración de la secuencia de imágenes y del iris de la cámara.
Sequence: Presenta la ventana de la figura 5, en la cual se especifica el directorio base donde se encuentra la secuencia de imágenes, el nombre del primer archivo y el número de imágenes de la secuencia.

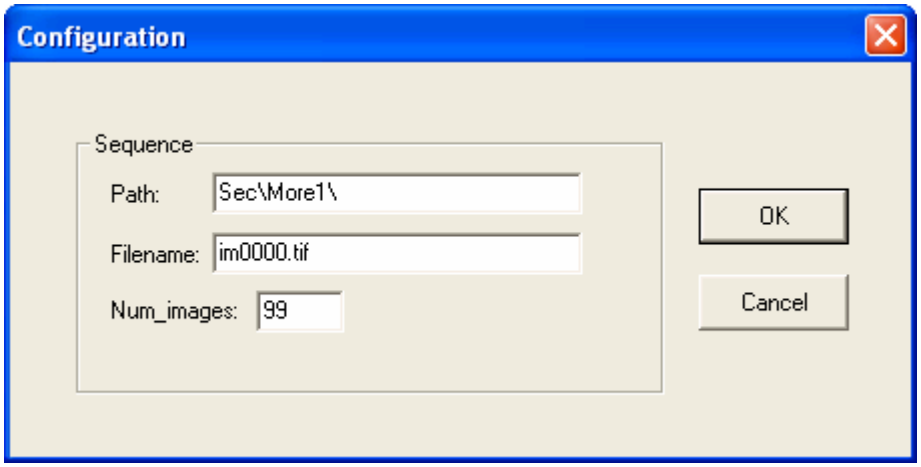


Figura 5. Ventana de configuración de la secuencia

Finalmente en el menú de ayuda en la opción acerca de se presenta una ventana informativa (Figura 6).

Figura 6. Ventana Acerca de Tracking

3. Secuencia de Operación

Tracking puede trabajar con archivos de secuencias de imágenes o con la señal de la cámara, según se seleccione el botón de la barra de herramientas: *Sequence* o *Camara*

Si selecciono el botón *Sequence*:

a. Se cargará la primera imagen de la secuencia por defecto que están en el directorio Sec\More1\.
b. Haga clic sobre el centro de la cara. Inmediatamente se habilitará el botón de *track*.
c. Haga click sobre el botón *track*. Se iniciará el seguimiento a partir del las coordenadas especificadas, se presentaran las diferentes imágenes de la secuencia. Mediante una elipse de color rojo en la imagen de la izquierda se mostrará el resultado de aplicar el algoritmo de seguimiento; mientras esto sucede se habilitará sólo el botón de stop.
d. Click en el botón *stop*.
e. Si desea repetir, seleccione nuevamente el botón *sequence*.

En la parte inferior de la imagen izquierda, se dispone de una mensaje de texto en el cual se especifica: x, y: coordenadas del centro de la elipse, vpan y vtilt: velocidades para el pantilt (solo cuando se ha seleccionado cámara), best_score: puntaje asignado a la región que ha obtenido el mejor resultado de comparación con el histograma de referencia, y fps: cuadro por segundo empleados en el procesamiento.

Si desea cambiar la secuencia de imágenes ir al menú *Configuration / Sequence*.

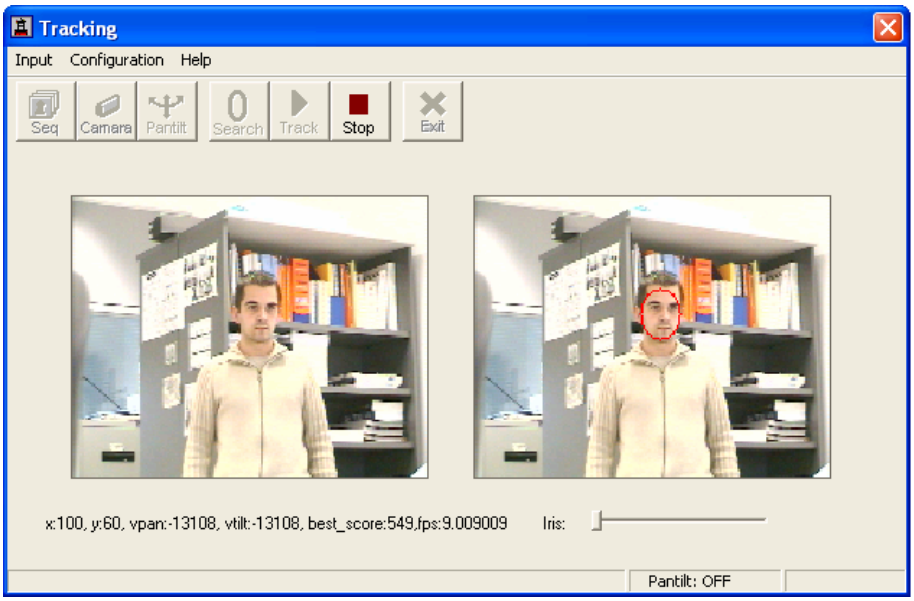


Figura 7. Tracking

Si selecciono el botón *Camara*:

Es necesario inicializar el sistema definiendo un histograma de referencia, para ello primero haga click sobre la cara que se desea que el robot encuentre y siga, así se almacenará en el archivo "*Histogram.bin*" el histograma de referencia que el programa utilizará. Luego haga click en *stop* y siga lo siguientes pasos.

a. Al seleccionar *Camara*, en la imagen de la izquierda se visualiza la señal de video captada por la cámara. Se habilitan los botones de *Pantilt* y de *Stop*.
b. Conectar el *pantilt* haciendo click en el botón correspondiente. En la barra de mensajes se mostrara "Pantilt:ON". Se habilitará el botón *Search*.
c. Se tiene dos opciones:
   c1. Si se hace click sobre el botón *Search*, la cámara se moverá haciendo un barrido en forma horizontal y girará el cuerpo del robot en la misma dirección. Cada vez que se detiene buscará en la imagen la zona que supere un nivel de umbral definido por THRESHOLD_HISTO que esta fijo a 700. Se mantendrá la búsqueda hasta tanto no se supere el nivel de umbral. Este nivel puede depender de las condiciones de iluminación y de la calidad de la imagen que se tenga, para las pruebas realizadas se observo que con este valor y las condiciones de iluminación del laboratorio se presentaban resultados aceptables.
   c2. Se puede especificar directamente la ubicación de la cara de la persona que se desea que el robot siga; esto se logra realizando click sobre la cara de la persona, inmediatamente se habilita el botón *track* y al hacer click sobre este se inicia el seguimiento de caras.
d. Presionar *stop* para detener el seguimiento de caras.

4. Detalles de la Implementación

4.1 Tracking

La captura de la señal de video de la cámara se realiza mediante el uso de las librerías MIL, para ello en la sección de OnInitDialog() se inicializan las variables M y pgen, que especifican los parámetros necesarios para las librerías MIL y los parámetros generales de las imágenes.

```
M=(Mil_parametres*)calloc(1,sizeof(Mil_parametres));
pgen=(Parametres_generals*)calloc(1,sizeof(Parametres_generals));
```

Para realizar la tarea de tracking es necesario definir un objeto de la clase ColorTracker en la sección de variables globales.

```
CColorTracker Tracker;
```

En el thread especificado por Myfunction las líneas que realizan la función del tracking para el caso de la señal de video de la cámara, son esencialmente las siguientes:

```
1   COutline Ellip;
2   while (!pgen->parar_el_thread_del_tracker) {
3       if (pgen->input==0) { //Camara
4           MbufCopyColor(pxccam->obtieneimagenrapida(), M->MilImatge0,
                    M_ALL_BAND);
5           MbufGetColor(M->MilImatge0,M_RGB24+M_PACKED,M_ALL_BAND,
                    I0->data);
6           MimResize(M->MilImatge0, M->MilImatge0_disp, pgen->
                ZoomX_Im_to_Disp, pgen->ZoomY_Im_to_Disp,M_DEFAULT);
7       }

        if (pgen->track==TRUE || pgen->input==1) {
8           Tracker.SearchForHead(I0->data,pgen->curr_x,pgen->curr_y,pgen->curr_z,
                                  &newx,&newy,&newz,&best_score);
9           if (newz>=7 && newz<17)newz=pgen->curr_z;
10          pgen->curr_x=newx;
11          pgen->curr_y=newy;
12          pgen->curr_z=newz;
13          pgen->best_score=best_score;

14          if (pt->JA_CONNECTAT==1) {
            //instruccions pel pan & tilt
15          consigna_vel_pan = PTU_PARM_PTR(Kp*(center_x-pgen->curr_x) +
                        Kdp*(pgen->curr_x-previous_state_x));
16          consigna_vel_tilt = PTU_PARM_PTR(-Kt*(center_y-pgen->curr_y) –
                        Kdt*(pgen->curr_y-previous_state_y));
17          if((consigna_vel_pan<tol_pan)&&(consigna_vel_pan>-tol_pan))
                consigna_vel_pan=0;
18          if((consigna_vel_pan!=0)&&(consigna_vel_pan!=previous_consign a_vel_
                pan)) set_desired(PAN,SPEED,& consigna_vel_pan,ABSOLUTE);
            if(consigna_vel_pan==0) halt(PAN);
19          if((consigna_vel_tilt<tol_tilt)&&(consigna_vel_tilt>-tol_tilt))
                consigna_vel_tilt=0;
20          if((consigna_vel_tilt!=0)&&(consigna_vel_tilt!=previous_consigna_vel_tilt))
                set_desired(TILT,SPEED,&consigna_vel_tilt,ABSOLUTE);
21          if(consigna_vel_tilt==0) halt(TILT);

22          previous_state_x = pgen->curr_x;
23          previous_state_y = pgen->curr_y;
24          previous_consigna_vel_pan = consigna_vel_pan;
25          previous_consigna_vel_tilt = consigna_vel_tilt;
26          }

27          Ellip.MakeEllipticalOutline(newz,int(1.2*newz));
28          if (newx>15 && newx<I0->nrows-15 && newy>15 && newy<I0->ncols-15)
                Ellip.PrintEllipInImame(I0->data,newx,newy,I0->nrows,I0->ncols);
29          MbufPutColor(M->MilImatge1,M_RGB24+M_PACKED, M_ALL_BAND,
                    I0->data);
30          MimResize(M->MilImatge1, M->MilImatge1_disp, pgen->ZoomX_
                    Im_to_Disp, pgen->ZoomY_Im_to_Disp, M_DEFAULT);
31      }
```

En las líneas 4 a 6 se dan las instrucciones utilizando las librerías MIL para la adecuación y presentación de la señal de video en el cuadro de imagen.

En la línea 8 se llama al método *SearchForHead* del objeto *tracker* de la clase *CcolorTracker*. Los parámetros que se le pasan son la matriz de la imagen (I0->data), las coordenadas del centro de la elipse y su tamaño (pgen->curr_x,pgen->curr_y,pgen->curr_z). Regresa la nueva ubicación de la elipse y su tamaño en las variables newx, newy y newz, además de best_score que indica el puntaje obtenido de la zona de la nueva elipse tonel histograma de referencia.

En la línea 9 se comprueba que el nuevo tamaño de la elipse este dentro del rango aceptado por las listas creadas para la zona de búsqueda (referirse a la sección 4.2).

En las líneas 10 a 13 se actualizan en las variables de parámetros generales (pgen) los valores obtenidos por *SearchForhead*.

Las líneas de la 15 a la 25 definen un control tipo PD para el pantilt.

En la línea 27, se llama al método *MakeEllipticalOutline* para que cree una elipse del tamaño newz.

En la línea 28, se dibuja la elipse modificando la variable que contiene la imagen (I0->data), pero antes de ello se comprueba que nos encontremos dentro de la zona de la imagen dejando un margen de 15 pixeles; con ello garantizamos que al dibujar la elipse no nos salgamos del área de la imagen, lo cual ocasionaría errores al invocar a *MakeEllipticalOutline*.

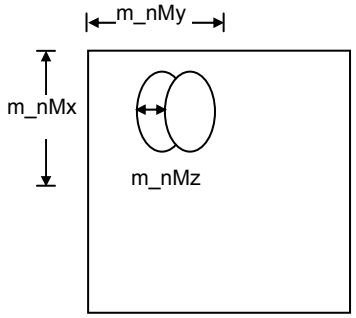
Con las líneas 29 y 30 se muestran los resultados obtenidos en el cuadro de imagen derecho.

4.2 Crear Listas

Para Crear las listas se corren las siguientes líneas:

```
// Clist list(m_nMx,m_nMy,m_nZinc,m_nSizeEllipMin,m_nSizeEllipMax);
   CList list(20,20,1,2,25);
  list.CalculateAndSaveLists("2_25AddList_20_20_1_new.bin","2_25SubList_20_20_1_new
                                .bin",I0->nrows,I0->ncols);
```

En el código de tracking estas líneas se encuentran comentadas y no se dejo un botón que las ejecute directamente.



Para este ejemplo se define un área de búsqueda con m_nMx y m_nMy igual a 20 como se indica en la figura, m_nZinc define el paso entre una elipse y otra. nSizeEllipMin y m_nSizeEllipMax definen el tamaño máximo y mínimo de la elipse.

Las listas generadas son almacenadas en los archivos: 2_25AddList_20_20_1_new.bin y 2_25SubList_20_20_1_new.bin. Estos archivos deben cargarse al seleccionar secuencia o cámara, en la línea correspondiente.
Tracker.InitLists("2_25AddList_20_20_1_new.bin","2_25SubList_20_20_1_new.bin");

El proceso de crear las listas puede tomar tiempo de cálculo pero sólo se generan una vez y luego se utilizan los archivos. No es recomendable utilizar los valores extremos por que al parecer da inconvenientes en los límites del área de búsqueda, para este caso se podría utilizar entre 5 y 22 por ejemplo.

4.3 Búsqueda de una cara

La búsqueda de una cara se realiza mediante el thread que se corre con la función Myfunction_search_head. Se realiza una exploración del ambiente realizando movimientos del pantilt y del cuerpo del robot. Se requiere que se defina un histograma inicial (referirse a la sección 3).

En esta implementación se observó que afecta bastante el cambio de iluminación del ambiente y hasta tanto no se implemente un algoritmo que permita solucionar esto, es necesario siempre inicializar el histograma de referencia. También puede variarse el nivel de umbral definido mediante la constante THRESHOLD_HISTO que se fijo en 700, este nivel depende también de las condiciones de iluminación y de la calidad de la imagen que se obtiene.

A continuación algunos comentarios sobre esta implementación, que falta depurar más.

```
    void Myfunction_search_head(void *resderes)
    {
1       double fps;
2       int x,y,z,score;
3       int pos_pan=0, pos_pan_max=900, pos_pan_min=-900, step=300;
4       int step_turn=15;
5       int borde_img=30;

6       std::string str;
7       ArTime start;
8       clock_t t1,t2;

        //Posición inicial pantilt
9       if (pt->JA_CONNECTAT ==1) {
10          close_host_port(pt->COMstream);
11          free(pt);
12          pt=(pantilt*)malloc(sizeof(pantilt));
13          pt->JA_CONNECTAT=1;
14          pt->status=NO_ERROR;
15          pt->status=LoadParam(".\\","param_PTU.txt",pt->param_pt,pt->status);
16          strcpy(pt->COMportName,COM_PANTILT); //COM1

17          pt->COMstream = open_host_port(pt->COMportName);
```

```
18          if (pt->COMstream != PORT_NOT_OPENED) {
19             pt->status=Settings(pt->param_pt,pt->status);
20             pos_pan=-300;
21             MoveToPositionPan(pt->COMstream, pt->param_pt, pos_pan, pt->status);
22             MoveToPositionTilt(pt->COMstream, pt->param_pt, -400, pt->status);
23          }
24       }

         // the serial connection (robot)
25       ArSerialConnection serConn;
         // robot
26       ArRobot robot;
         // mandatory init
27       Aria::init();

         //modify the next line if you're not using the first serial port to talk to your robot
28       serConn.setPort(COM_ROBOT); //COM2
29       robot.setDeviceConnection(&serConn);

         //try to connect, if we fail exit
30       if (!robot.blockingConnect()) {
31          MessageBox(0,"Could not connect to robot... exiting\n","",0);
32          Aria::shutdown();
33       }

34       else {
35         Tracker.LoadReFromFile("Histogram.bin");

36         robot.comInt(ArCommands::ENABLE, 1);
37         robot.comInt(ArCommands::SOUNDTOG, 0);

           // run the robot in its own thread, so it gets and processes packets and such
38         robot.runAsync(false);

39         while (!pgen->parar_thread_search) {
40            t1=clock();
41            for (int k=0; k<=7; k=k+1) { //Movimiento del pantilt
42               pgen->best_score=0;
43               MoveToPositionPan(pt->COMstream, pt->param_pt, pos_pan,pt->status);
44               robot.lock();
45               robot.setDeltaHeading(step_turn);
46               robot.unlock();

47               for (int i =borde_img; i<I0->nrows - borde_img; i=i+15) {
48                  for (int j =borde_img; j<I0->ncols - borde_img ; j=j+15) {
49                     Tracker.SearchForHead(I0->data,i,j,11,&x,&y,&z,&score);
50                     if  (score >  pgen->best_score) {
51                        pgen->best_score=score;
52                        pgen->curr_x=x; pgen->curr_y=y; pgen->curr_z=z;
53                     }
54                  }
55               }
56               if (pgen->best_score>THRESHOLD_HISTO) {
60                  halt(ALL);
61                  set_mode(SPEED_CONTROL_MODE,PTU_PURE_VELOCITY_
                              SPEED_CONTROL_MODE);
```

```
62                      pgen->parar_thread_search = 1;
63                      pgen->track=TRUE;
64                      tts_salle("T'he trobat","Lenta","");
65                      break;
66                  }
67                  if (pgen->parar_thread_search==1) break;
68                  if (step>0 && pos_pan==pos_pan_max){ step=-300; step_turn=-15;}
69                  if (step<0 && pos_pan==pos_pan_min) { step=300; step_turn=+15;}
70                  pos_pan=pos_pan+step;
71                  t2=clock();
72                  fps=CLOCKS_PER_SEC/(double(t2-t1));
73              }
74          }
75      }
76      ArUtil::sleep(2000);
77      Aria::shutdown();
78  }
```

En la línea 3 las variables pos_pan, pos_pan_min, pos_pan_max y step; definen la posición actual del pan, el rango mínimo y máximo y el paso de avance. Con estas posiciones se hace un barrido de una buena parte de la zona en la que se encuentre el robot.

En la línea 4, step_turn define el giro del cuerpo del robot en cada paso.

En las líneas 9 a 24 se repite el código para inicializar el pantilt y define la posición inicial. Fue necesario incluir este código para que el pantilt se reinicializará desde este thread y no presentará conflicto con instrucciones similares presentes en myfunction.

Desde las línea 25 hasta la 33 y en 36 a 38, se presentan apartes tomados de los programas demos de aria para inicializar la comunicación con el robot.

En la línea 35 se carga el histograma de referencia que previamente debe ser almacenado en el archivo Histogram.bin.

A partir de la línea 39 hasta 75 se da el proceso de búsqueda mediante el ciclo while. Por lo tanto el robot se mantendrá buscando la cara de referencia hasta que encuentre una puntuación que supere a THRESHOLD_HISTO o se presione el botón stop.

En las líneas 43 a 46 se posiciona el robot para iniciar la búsqueda.

En las líneas 47 a 55 se implementan dos ciclos for para realizar la búsqueda en un marco interno de la imagen (borde_img). Se calcula el best_score más alto que resulte y si este es superior al THRESHOLD_HISTO se detiene el thread y se habilita el inicio del tracking fijando pgen->track=true en la línea 63. En caso contrario se prepararía para avanzar a la siguiente posición de búsqueda mediante las instrucciones de las líneas 67 a 70.

```
// TrackingDlg.cpp : implementation file
//
//Conexiones necesarias para el funcionamiento de la demo tracking:
//Conectar el pantilt al COM1 del PC
//Conectar el COM2 del PC al puerto serie del ordenador del robot
//Si se desea control del IRIS esta configurado para el COM2

#include "stdafx.h"
#include "Tracking.h"
#include "TrackingDlg.h"

#include <math.h>
#include <time.h>

#include "Aria.h"
#include "definicions.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//*******definició de les variables globals********

Mil_parametres *M;
Parametres_generals *pgen;
CImage_unsigned_char *I0;
camara *pxccam;
pantilt *pt;
CCamara *control;
CColorTracker Tracker;
Sequences *seq;

BOOL CTrackingDlg::OnInitDialog()
{
	// TODO: Add extra initialization here

	//Inicializando sintetizador
	hinstLib_tts_salle = LoadLibrary("tts_salle");
	tts_salle = (TTS_SALLE)GetProcAddress(hinstLib_tts_salle, "tts_salle");

	// Ventana de presentación
	CInicio  dlgInicio(this);
	dlgInicio.DoModal();

	tts_salle("Bienvenidos a Tracking de caras","Lenta","");

	//Inicializar la barra de herramientas y de estado
	Incializar_Toolbar();
	Inicializar_Satus_Bar();
```

```
	//Inicializar variables
	pt=(pantilt*)malloc(sizeof(pantilt));
	pt->JA_CONNECTAT = 0;

	M=(Mil_parametres*)calloc(1,sizeof(Mil_parametres));
	pgen=(Parametres_generals*)calloc(1,sizeof(Parametres_generals));
	pgen->input=2;
	seq=(Sequences*)calloc(1,sizeof(Sequences));

	_tcscpy(seq->Path,"Sec\\More1\\");
	_tcscpy(seq->Filename,"im0000.tif");
	seq->NumImages =99;

	tts_salle("Seleccione secuencia o camara","Lenta","");

	return TRUE;  //return TRUE unless you set the focus to a control
}

void CTrackingDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
	CDialog::OnLButtonDown(nFlags, point);
	pgen->XDisplay=point.x;
	pgen->YDisplay=point.y;

	if (Out_Limits_Image(pgen->XDisplay,pgen->YDisplay) & (pgen->input < 2)){
		MessageBox("Fuera de los límites de la imagen!", "Error",
		MB_ICONINFORMATION);
	}
	else if (pgen->input < 2){

		pgen->parar_el_thread_del_tracker=1;

		pgen->XFinestra=pgen->XDisplay-XMIN;
		pgen->YFinestra=pgen->YDisplay-YMIN;
		pgen->XRedim=long(double(pgen->XFinestra)/pgen->ZoomX_Im_to_Disp);
		pgen->YRedim=long(double(pgen->YFinestra)/pgen->ZoomY_Im_to_Disp);

		pgen->curr_x=pgen->XRedim;
		pgen->curr_y=pgen->YRedim;
		pgen->curr_z=11; //Tamaño inicial de la elipse

		// Inicializar histograma
		Tracker.SetReferenceHistogram(pgen->curr_x,pgen->curr_y,pgen-
		>curr_z,I0->data);
		//Guardar histograma en archivo
		Tracker.SaveRefHistogramToFile("Histogram.bin");

		//Enabled button track
		CToolBar* pBar = &m_wndToolBar;
		UINT iButtonID, iButtonStyle;
		int  iButtonImage;
		pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage); //Button  track
```

```
                pBar->SetButtonInfo(5, iButtonID, 0 ,iButtonImage);
                m_statusbar.SetPaneText(0,"Click in track button",TRUE);
        }
}

void CTrackingDlg::OnMouseMove(UINT nFlags, CPoint point)
{

        CDialog::OnMouseMove(nFlags, point);
        char text[100];
        long XFinestra, YFinestra, XRedim,YRedim;
        if (Out_Limits_Image(point.x,point.y)) {
                m_statusbar.SetPaneText(2," ",TRUE);
        }
        else if (pgen->input < 2) {
                XFinestra=point.x-XMIN;
                YFinestra=point.y-YMIN;
                XRedim=long(double(XFinestra)/pgen->ZoomX_Im_to_Disp);
                YRedim=long(double(YFinestra)/pgen->ZoomY_Im_to_Disp);
                sprintf (text,"  X=%ld   Y=%ld" ,XRedim,YRedim);
                m_statusbar.SetPaneText(2,text,TRUE);
        }
}

BOOL CTrackingDlg::Out_Limits_Image(long X, long Y)
{
        if(X<XMIN+15 || X>XMAX-15 || Y<YMIN+15 || Y>YMAX-15) return(TRUE);
        else return (FALSE);
}

void CTrackingDlg::Incializar_Toolbar()
{
//  Create the Toolbar and attach the resource
        if(!m_wndToolBar.Create(this) || !m_wndToolBar.LoadToolBar(IDR_TOOLBAR))
        {
                 TRACE0("Failed to Create Dialog Toolbar\n");
                 EndDialog(IDCANCEL);
        }
        CRect  rcClientOld; // Old Client Rect
        CRect  rcClientNew; // New Client Rect with Tollbar Added
        GetClientRect(rcClientOld); // Retrive the Old Client WindowSize

        // Called to reposition and resize control bars in the client rea of a window.
        // The reposQuery FLAG does not really traw the  Toolbar.  It only does the
        // calculations.
        // And puts the new ClientRect values in rcClientNew so we can do the rest of the
        // Math.
        RepositionBars(AFX_IDW_CONTROLBAR_FIRST,
        AFX_IDW_CONTROLBAR_LAST,0,reposQuery,rcClientNew);

        // All of the Child Windows (Controls) now need to be moved so the Tollbar does
        // not cover them up.
```

```
	// Offest to move all child controls after adding Tollbar
	CPoint ptOffset(rcClientNew.left-rcClientOld.left,rcClientNew.top-rcClientOld.top);
	CRect  rcChild;

	// Handle to the Dialog Controls
	CWnd*pwndChild = GetWindow(GW_CHILD);
	while(pwndChild) // Cycle through all child controls
	{
		pwndChild->GetWindowRect(rcChild); // Get the child control RECT
		ScreenToClient(rcChild);
		// Changes the Child Rect by the values of the claculated offset
		rcChild.OffsetRect(ptOffset);
		pwndChild->MoveWindow(rcChild,FALSE); // Move the Child Control
		pwndChild = pwndChild->GetNextWindow();
	}

	CRect  rcWindow;
	GetWindowRect(rcWindow); // Get the RECT of the Dialog

	// Increase width to new Client Width
	rcWindow.right += rcClientOld.Width() - rcClientNew.Width();

	// Increase height to new Client Height
	rcWindow.bottom += rcClientOld.Height() - rcClientNew.Height();
	MoveWindow(rcWindow,FALSE); // Redraw Window

	// Now we REALLY Redraw the Toolbar
	RepositionBars(AFX_IDW_CONTROLBAR_FIRST,
	AFX_IDW_CONTROLBAR_LAST,0);

	// Disabling button controls
	CToolBar* pBar = &m_wndToolBar;
	UINT iButtonID, iButtonStyle;
	int  iButtonImage;
	pBar->GetButtonInfo(4, iButtonID, iButtonStyle,iButtonImage);     //Button search
	pBar->SetButtonInfo(4, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage);     //Button track
	pBar->SetButtonInfo(5, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(6, iButtonID, iButtonStyle,iButtonImage);     //Button stop
	pBar->SetButtonInfo(6, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(2, iButtonID, iButtonStyle,iButtonImage);     //Button pantilt
	pBar->SetButtonInfo(2, iButtonID, TBBS_DISABLED ,iButtonImage);
}

void CTrackingDlg::Inicializar_Satus_Bar()
{
	static UINT BASED_CODE indicators[] =
	{
		ID_STATUS_TEXT,
		ID_STATUS_XY
	};
	//Inicializar status bar
```

```
        m_statusbar.Create(this); //We create the status bar
        m_statusbar.SetIndicators(indicators,3); //Set the number of panes
        CRect rect;
        GetClientRect(&rect);
        //Size the panes
        m_statusbar.SetPaneInfo(0,ID_STATUS_TEXT,SBPS_NORMAL,rect.Width()-200);
        m_statusbar.SetPaneInfo(1,ID_STATUS_PANTILT,SBPS_NORMAL,100);
        m_statusbar.SetPaneInfo(2,ID_STATUS_XY,SBPS_NORMAL,100);

        //This is where we actually draw it on the screen
        RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAS
        T,ID_STATUS_XY);
}

void CTrackingDlg::Alocatar_Buffers_Mil_Seq()
{
        char name1[50];

        //Alocatacions de l'aplicació i sistemes (VGA i Meteor)
        MappAlloc(M_DEFAULT, &M->MilApplication);
        MsysAlloc(M_SYSTEM_VGA, M_DEV0, M_NO_DDRAW, &(M->MilSystem));

        //Alocatacions dels displays
        MdispAlloc(M->MilSystem, M_DEFAULT, M_DEF_DISPLAY_FORMAT,
        M_DEFAULT, &M->MilDisplay0);
        MdispAlloc(M->MilSystem, M_DEFAULT, M_DEF_DISPLAY_FORMAT,
        M_DEFAULT, &M->MilDisplay1);

        //Es redimensiona la imatge al tamany de la finestra (només per la visualització, i
        //no per al tractament, que es fa amb el tamany definit per l'usuari)
        m_image0.GetWindowRect(pgen->dimensiones_imagen0);

        //Capturo el tamany de les finestres
        pgen->Amplada_Finestra=abs(pgen->dimensiones_imagen0.left-pgen-
        >dimensiones_imagen0.right);
        pgen->Altura_Finestra=abs(pgen->dimensiones_imagen0.top-pgen-
        >dimensiones_imagen0.bottom);

        //Alocato els buffers de display del tamany de les finestres de l'entorn
        MbufAllocColor(M->MilSystem,3,pgen->Amplada_Finestra,pgen-
        >Altura_Finestra,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP, &M-
        >MilImatge0_disp);
        MbufAllocColor(M->MilSystem,3,pgen->Amplada_Finestra,pgen-
        >Altura_Finestra,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP, &M-
        >MilImatge1_disp);

        //Carrego imatge originial per veure'n el tamany
        strcpy(name1,seq->Path);
        strcat(name1,seq->Filename);
        MbufRestore(name1, M->MilSystem, &M->MilImatge0);
        MbufInquire(M->MilImatge0,M_SIZE_X,&pgen->SizeX);
        MbufInquire(M->MilImatge0,M_SIZE_Y,&pgen->SizeY);
```

```
	//Alocatació de la resta d'imatges al tamany original
	MbufAllocColor(M->MilSystem,3,pgen->SizeX,pgen-
	>SizeY,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP,&M->MilImatge1);

	//coeficients  per la reduccio a pantalla
	pgen->ZoomX_Im_to_Disp=double(double(pgen->Amplada_Finestra)/double(pgen-
	>SizeX));
	pgen->ZoomY_Im_to_Disp=double(double(pgen->Altura_Finestra)/double(pgen-
	>SizeY));

	//imagen adecuada a la ventana
	MdispSelectWindow(M->MilDisplay0, M->MilImatge0_disp,m_image0.m_hWnd);
	MdispSelectWindow(M->MilDisplay1, M->MilImatge1_disp,m_image1.m_hWnd);

	I0=new CImage_unsigned_char(pgen->SizeX,pgen->SizeY,3);

	//Se carga la primera imagen
	MbufLoad(name1, M->MilImatge0);
	MbufGetColor(M->MilImatge0,M_RGB24+M_PACKED,M_ALL_BAND,I0->data);
	MimResize(M->MilImatge0, M->MilImatge0_disp, pgen->ZoomX_Im_to_Disp, pgen-
	>ZoomY_Im_to_Disp, M_DEFAULT);
}

void CTrackingDlg::Alocatar_Buffers_Mil_Camara()
{
	//Alocatacions de l'aplicació i sistemes (VGA i Meteor)
	//MappAlloc(M_DEFAULT, &M->MilApplication);
	MsysAlloc(M_SYSTEM_VGA, M_DEV0, M_NO_DDRAW, &(M->MilSystem));

	//Alocatacions dels displays
	MdispAlloc(M->MilSystem, M_DEFAULT, M_DEF_DISPLAY_FORMAT,
	M_DEFAULT, &M->MilDisplay0);
	MdispAlloc(M->MilSystem, M_DEFAULT, M_DEF_DISPLAY_FORMAT,
	M_DEFAULT, &M->MilDisplay1);
	MgraAlloc(M->MilSystem, &M->MilGrafic);

	//Es redimensiona la imatge al tamany de la finestra (només per la visualització, i
	//no per al tractament, que es fa amb el tamany definit per l'usuari)
	m_image0.GetWindowRect(pgen->dimensiones_imagen0);

	//Capturo el tamany de les finestres
	pgen->Amplada_Finestra=abs(pgen->dimensiones_imagen0.left-pgen-
	>dimensiones_imagen0.right);
	pgen->Altura_Finestra=abs(pgen->dimensiones_imagen0.top-pgen-
	>dimensiones_imagen0.bottom);

	//Alocato la resta de buffers MIL
	MbufAllocColor(M->MilSystem,3,pgen->Amplada_Finestra,pgen-
	>Altura_Finestra,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP, &M-
	>MilImatge0_disp);
```

```
        MbufAllocColor(M->MilSystem,3,pgen->Amplada_Finestra,pgen-
        >Altura_Finestra,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP, &M-
        >MilImatge1_disp);

        //Alocatació de la resta d'imatges al tamany original

        MbufAllocColor(M->MilSystem,3L,pxccam->SizeX,pxccam-
        >SizeY,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP,&M->MilImatge0);
        MbufAllocColor(M->MilSystem,3L,pxccam->SizeX,pxccam-
        >SizeY,8+M_UNSIGNED,M_IMAGE+M_PROC+M_DISP,&M->MilImatge1);

        //coeficients  per la reduccio a pantalla
        pgen->ZoomX_Im_to_Disp=double(double(pgen-
        >Amplada_Finestra)/double(pxccam->SizeX));
        pgen->ZoomY_Im_to_Disp=double(double(pgen->Altura_Finestra)/double(pxccam-
        >SizeY));

        //imagen adecuada a la ventana
        MdispSelectWindow(M->MilDisplay0, M->MilImatge0_disp,m_image0.m_hWnd);
        MdispSelectWindow(M->MilDisplay1, M->MilImatge1_disp,m_image1.m_hWnd);

        I0=new CImage_unsigned_char(pxccam->SizeX,pxccam->SizeY,3);
}

/*void CTrackingDlg::OnButtonList()
{
        // TODO: Add your control notification handler code here
        // Crear listas. list(m_nMx,m_nMy,m_nZinc,m_nSizeEllipMin,m_nSizeEllipMax);
        CList list(12,12,1,2,18);
        list.CalculateAndSaveLists("2_18AddList_12_12_1_new.bin","2_18SubList_12_12_
        1_new.bin",I0->nrows,I0->ncols);

}*/

void CTrackingDlg::OnButtonCamara()
{
        pgen->parar_el_thread_del_tracker=1;

        if (pgen->input<2){
                Liberar_Buffers();
                //Inicialización parametros MIL
                M=(Mil_parametres*)calloc(1,sizeof(Mil_parametres));
                pgen=(Parametres_generals*)calloc(1,sizeof(Parametres_generals));
        }

        pgen->input=0; //Señal desde la camara
        pgen->track = FALSE;
        pxccam = new camara(&m_image0);
        Alocatar_Buffers_Mil_Camara();

        //Definición de listas
```

```
        Tracker.InitLists("2_18AddList_10_10_1_new.bin","2_18SubList_10_10_1_new.bin"
        );
        Tracker.ColorImagesInit(I0->nrows,I0->ncols);

        //Barra de herramientas
        CToolBar* pBar = &m_wndToolBar;
        UINT iButtonID, iButtonStyle;
        int  iButtonImage;
        pBar->GetButtonInfo(0, iButtonID, iButtonStyle,iButtonImage);     //Buttonsequence
        pBar->SetButtonInfo(0, iButtonID, TBBS_DISABLED ,iButtonImage);
        pBar->GetButtonInfo(1, iButtonID, iButtonStyle,iButtonImage);     //Button camara
        pBar->SetButtonInfo(1, iButtonID, TBBS_DISABLED ,iButtonImage);
        pBar->GetButtonInfo(2, iButtonID, iButtonStyle,iButtonImage);     //Button pantilt
        pBar->SetButtonInfo(2, iButtonID, 0 ,iButtonImage);
        pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage);     //Button track
        pBar->SetButtonInfo(5, iButtonID, TBBS_DISABLED ,iButtonImage);
        pBar->GetButtonInfo(6, iButtonID, iButtonStyle,iButtonImage);     //Button stop
        pBar->SetButtonInfo(6, iButtonID, 0 ,iButtonImage);

        pgen->parar_el_thread_del_tracker=0;
        _beginthread(Myfunction,0,&m_text);
}

void CTrackingDlg::OnMenuInputCamara()
{
        CTrackingDlg::OnButtonCamara();
}

void CTrackingDlg::OnButtonSequence()
{
        pgen->parar_el_thread_del_tracker=1;
        if(pt->JA_CONNECTAT==1)CTrackingDlg::OnButtonPantilt();
        if (pgen->input<2){
                Liberar_Buffers();
                //Inicialización parametros MIL
                M=(Mil_parametres*)calloc(1,sizeof(Mil_parametres));
                pgen=(Parametres_generals*)calloc(1,sizeof(Parametres_generals));
        }
        pgen->input=1; //Secuencia
        Alocatar_Buffers_Mil_Seq();

        //Definición de listas
        Tracker.InitLists("2_18AddList_10_10_1_new.bin","2_18SubList_10_10_1_new.bin"
        );
        Tracker.ColorImagesInit(I0->nrows,I0->ncols);

        CToolBar* pBar = &m_wndToolBar;
        UINT iButtonID, iButtonStyle;
        int  iButtonImage;
        pBar->GetButtonInfo(2, iButtonID, iButtonStyle,iButtonImage);     //Button pantilt
        pBar->SetButtonInfo(2, iButtonID, TBBS_DISABLED ,iButtonImage);
        pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage);     //Button track
```

```
	pBar->SetButtonInfo(5, iButtonID, TBBS_DISABLED ,iButtonImage);
	m_statusbar.SetPaneText(0,"Click in the image to define hystogram",TRUE);
}

void CTrackingDlg::OnMenuInputSequence()
{
	CTrackingDlg::OnButtonSequence();
}

void CTrackingDlg::OnButtonTrack()
{
	pgen->parar_el_thread_del_tracker=1;

	CToolBar* pBar = &m_wndToolBar;
	UINT iButtonID, iButtonStyle;
	int  iButtonImage;
	pBar->GetButtonInfo(0, iButtonID, iButtonStyle,iButtonImage);	//Button sequence
	pBar->SetButtonInfo(0, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(1, iButtonID, iButtonStyle,iButtonImage);	//Button camara
	pBar->SetButtonInfo(1, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(2, iButtonID, iButtonStyle,iButtonImage);	//Button pantilt
	pBar->SetButtonInfo(2, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(4, iButtonID, iButtonStyle,iButtonImage);	//Button search
	pBar->SetButtonInfo(4, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage);	//Button track
	pBar->SetButtonInfo(5, iButtonID, TBBS_DISABLED ,iButtonImage);
	pBar->GetButtonInfo(6, iButtonID, iButtonStyle,iButtonImage);	//Button stop
	pBar->SetButtonInfo(6, iButtonID, 0 ,iButtonImage);
	pBar->GetButtonInfo(8, iButtonID, iButtonStyle,iButtonImage);	//Button exit
	pBar->SetButtonInfo(8, iButtonID, TBBS_DISABLED ,iButtonImage);
	m_statusbar.SetPaneText(0,"",TRUE);

	if (pgen->input==0)	pgen->track=TRUE;
	else pgen->track=FALSE;

	pgen->parar_el_thread_del_tracker=0;
	_beginthread(Myfunction,0,&m_text);
}

void CTrackingDlg::OnButtonStop()
{
	pgen->parar_el_thread_del_tracker=1;
	pgen->parar_thread_search = 1;

	pgen->track=FALSE;

	CToolBar* pBar = &m_wndToolBar;
	UINT iButtonID, iButtonStyle;
	int  iButtonImage;
	if (pgen->track == 1){
		pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage);//Button track
		pBar->SetButtonInfo(5, iButtonID, 0 ,iButtonImage);
```

```
        }
        else{
                pBar->GetButtonInfo(5, iButtonID, iButtonStyle,iButtonImage);  //Button track
                pBar->SetButtonInfo(5, iButtonID, TBBS_DISABLED ,iButtonImage);
        }
        pBar->GetButtonInfo(6, iButtonID, iButtonStyle,iButtonImage);     //Button stop
        pBar->SetButtonInfo(6, iButtonID, TBBS_DISABLED ,iButtonImage);
        pBar->GetButtonInfo(0, iButtonID, iButtonStyle,iButtonImage);     //Button sequence
        pBar->SetButtonInfo(0, iButtonID, 0 ,iButtonImage);
        pBar->GetButtonInfo(1, iButtonID, iButtonStyle,iButtonImage);     //Button camara
        pBar->SetButtonInfo(1, iButtonID, 0 ,iButtonImage);
        pBar->GetButtonInfo(4, iButtonID, iButtonStyle,iButtonImage);     //Button search
        pBar->SetButtonInfo(4, iButtonID, 0 ,iButtonImage);
        pBar->GetButtonInfo(8, iButtonID, iButtonStyle,iButtonImage);     //Button exit
        pBar->SetButtonInfo(8, iButtonID, 0 ,iButtonImage);
        if (pgen->input==0) { //Camara
                pBar->GetButtonInfo(2, iButtonID, iButtonStyle,iButtonImage); //Button pantilt
                pBar->SetButtonInfo(2, iButtonID, 0 ,iButtonImage);
        }
}

void CTrackingDlg::OnMenuConfigurationSequence()
{
        int r;
        CString Filename;
        Cconfig_sequence dlgConfig(this);
        //Valores iniciales
        dlgConfig.m_path = seq->Path;
        dlgConfig.m_filename = seq->Filename;
        dlgConfig.m_num_images = seq->NumImages;
        r=dlgConfig.DoModal ();
        _tcscpy(seq->Path,dlgConfig.m_path);
        _tcscpy(seq->Filename,dlgConfig.m_filename);
        seq->NumImages = dlgConfig.m_num_images;

        CTrackingDlg::OnButtonSequence();
}

void CTrackingDlg::Connect_Pant_Tilt()
{
        pt->status=NO_ERROR;
        pt->status=LoadParam(".\\","param_PTU.txt",pt->param_pt,pt->status);
        strcpy(pt->COMportName,COM_PANTILT);  //COM1

        pt->COMstream = open_host_port(pt->COMportName);

        if ( pt->COMstream == PORT_NOT_OPENED )     {
                MessageBox("\nSerial Port setup error.\n",NULL,MB_OK);
                pt->status=PTU_CABLE_DISCONNECTED;
        }

        if (pt->COMstream != PORT_NOT_OPENED) {
```

```
		pt->status=Settings(pt->param_pt,pt->status);
		MoveToPositionPan(pt->COMstream, pt->param_pt, 0, pt->status);
		MoveToPositionTilt(pt->COMstream, pt->param_pt, -400, pt->status);
	}
}

void CTrackingDlg::Disconnect_Pant_Tilt()
{
	close_host_port(pt->COMstream);
}

void CTrackingDlg::OnButtonPantilt()
{
	CToolBar* pBar = &m_wndToolBar;
	UINT iButtonID, iButtonStyle;
	int  iButtonImage;

	if (pt->JA_CONNECTAT ==0) {	//Alocatació pan&tilt
		Connect_Pant_Tilt();
		if(pt->status==NO_ERROR) pt->JA_CONNECTAT=1;
		m_statusbar.SetPaneText(1,"   Pantilt: ON",TRUE);
		//Enabled button search
		pBar->GetButtonInfo(4, iButtonID, iButtonStyle,iButtonImage);//Button search
		pBar->SetButtonInfo(4, iButtonID, 0 ,iButtonImage);
	}
	else if (pt->JA_CONNECTAT ==1) {	//free pan&tilt
		Disconnect_Pant_Tilt();
		free(pt);
		pt=(pantilt*)malloc(sizeof(pantilt));
		pt->JA_CONNECTAT=0;
		m_statusbar.SetPaneText(1,"   Pantilt: OFF",TRUE);
		//Disabled button search
		pBar->GetButtonInfo(4, iButtonID, iButtonStyle,iButtonImage);//Button search
		pBar->SetButtonInfo(4, iButtonID, TBBS_DISABLED ,iButtonImage);
	}
}

void CTrackingDlg::OnButtonExit()
{
	pgen->parar_el_thread_del_tracker=1;
	pgen->parar_thread_search = 1;
	pgen->track=FALSE;
	Sleep(500);

	if (pgen->input<2) Liberar_Buffers();
	if (pt->JA_CONNECTAT ==1) CTrackingDlg::OnButtonPantilt() ;
	free(pt); //free pan&tilt
	free(seq);
	OnOK();
}

void CTrackingDlg::OnMenuExit()
```

```
{
	CTrackingDlg::OnButtonExit();
}

void CTrackingDlg::Liberar_Buffers()
{
	if (M->MilImatge0>0)MbufFree(M->MilImatge0);
	if (M->MilImatge1>0)MbufFree(M->MilImatge1);
	if (M->MilImatge0_disp>0)MbufFree(M->MilImatge0_disp);
	if (M->MilImatge1_disp>0)MbufFree(M->MilImatge1_disp);

	MdispFree(M->MilDisplay0);
	MdispFree(M->MilDisplay1);
	MsysFree(M->MilSystem);
	if (pgen->input==1)MappFree(M->MilApplication);
	if (pgen->input==0)    pxccam->~camara();
	free(M);
	free(pgen);
}

void CTrackingDlg::OnMenuAbout()
{
	CAboutDlg dlgAbout;
	dlgAbout.DoModal();
}

void CTrackingDlg::OnReleasedcaptureSliderIris(NMHDR* pNMHDR, LRESULT*
pResult)
{
	if (pgen->iris_cx) {
		UpdateData(TRUE);
		pSlideriris=(CSliderCtrl*) GetDlgItem(IDC_SLIDER_IRIS);
		m_iris=pSlideriris->GetPos();
		p_pos_iris=m_iris;

		control->Manual();
		control->IrisDirect(p_pos_iris);
		pos_iris_cam=p_pos_iris;
		*pResult = 0;
	}
	else MessageBox("Do click in Menu Configuration/Iris");
}

void CTrackingDlg::OnMenuConfigIris()
{
	//Iris:COM2, definido en LineaSerie.cpp en CLineaSerie()
	pSlideriris= (CSliderCtrl*) GetDlgItem(IDC_SLIDER_IRIS);
	pSlideriris->SetRange(0,17);
	pSlideriris->SetPos(m_iris);

	control= new CCamara();
	if (control -> Error == LSMAL) {
```

```
                delete control;
                control = NULL;
        }
        control->Manual();
        MessageBox("Iris connected to COM2");
        pgen->iris_cx=TRUE;
}

void Myfunction(void *resderes)
{
        CStatic *button = (CStatic*)resderes;

        char name1[50],name2[50];
        int cont=0;
        int newx, newy,newz,best_score;
        double fps;
        char cad[80];

        clock_t t1,t2;

        //Pan tilt params
        long center_x, center_y;
        PTU_PARM_PTR consigna_vel_pan, consigna_vel_tilt;
        PTU_PARM_PTR previous_consigna_vel_pan, previous_consigna_vel_tilt;
        PTU_PARM_PTR tol_pan, tol_tilt;
        double Kp, Kt, Kdp, Kdt;
        long previous_state_x, previous_state_y;

        //Parametros para el control del pan & tilt
        Kp = 30.0; Kdp = 10.0;
        Kt = 30.0; Kdt = 10.0;
        tol_pan = 10;
        tol_tilt = 10;
        center_x = I0->nrows/2;  //coordenadas del centro de la imagen.
        center_y = I0->ncols/2;
        previous_state_x = I0->nrows/2;   //inicialización
        previous_state_y = I0->ncols/2;

         //Mode control per velocitat
        if (pt->JA_CONNECTAT==1) {
                set_mode(SPEED_CONTROL_MODE,PTU_PURE_VELOCITY_SPEED_
                        CONTROL_MODE);
        }

        COutline Ellip;

        while (!pgen->parar_el_thread_del_tracker) {
                t1=clock();
                if (pgen->input==1){  //Secuencia
                        strcpy(name1,seq->Path);
                        if (cont<10){
                                sprintf(name2,"im000%d.tif",cont);
```

```
                strcat(name1,name2);
            }
            else if (cont>=10 && cont<100){
                sprintf(name2,"im00%d.tif",cont);
                strcat(name1,name2);
            }
            cont++;

            MbufLoad(name1, M->MilImatge0);
            MbufGetColor(M->MilImatge0,M_RGB24+M_PACKED,
                        M_ALL_BAND,I0->data);
            MimResize(M->MilImatge0, M->MilImatge0_disp, pgen->ZoomX_Im
                    _to_Disp, pgen->ZoomY_Im_to_Disp, M_DEFAULT);
}
else if (pgen->input==0) { //Camara
            MbufCopyColor(pxccam->obtieneimagenrapida(), M->MilImatge0,
                        M_ALL_BAND);
            MbufGetColor(M->MilImatge0,M_RGB24+M_PACKED,
                        M_ALL_BAND,I0->data);
            MimResize(M->MilImatge0, M->MilImatge0_disp, pgen->ZoomX_Im
                        _to_Disp, pgen->ZoomY_Im_to_Disp, M_DEFAULT);
}

if (pgen->track==TRUE || pgen->input==1) {
            Tracker.SearchForHead(I0->data,pgen->curr_x,pgen->curr_y,pgen-
                                >curr_z,&newx,&newy,&newz,&best_score);
            if (newz>=7 && newz<17)newz=pgen->curr_z;
            pgen->curr_x=newx;
            pgen->curr_y=newy;
            pgen->curr_z=newz;
            pgen->best_score=best_score;

            if (pt->JA_CONNECTAT==1) {
                    //instruccions pel pan & tilt
                consigna_vel_pan = PTU_PARM_PTR(Kp*(center_x-pgen-
                        >curr_x) + Kdp*(pgen->curr_x-previous_state_x));
                consigna_vel_tilt = PTU_PARM_PTR(-Kt*(center_y-pgen-
                        >curr_y) - Kdt*(pgen->curr_y-previous_state_y));

                if((consigna_vel_pan<tol_pan)&&(consigna_vel_pan>-tol_pan))
                    consigna_vel_pan=0;

                if((consigna_vel_pan!=0)&&(consigna_vel_pan!=previous_consign
                a_vel_pan))
                set_desired(PAN,SPEED,&consigna_vel_pan,ABSOLUTE);
                 if(consigna_vel_pan==0) halt(PAN);

                if((consigna_vel_tilt<tol_tilt)&&(consigna_vel_tilt>-tol_tilt))
                    consigna_vel_tilt=0;

                if((consigna_vel_tilt!=0)&&(consigna_vel_tilt!=previous_consigna_v
                el_tilt)) set_desired(TILT,SPEED,&consigna_vel_tilt,ABSOLUTE);
```

```
                        if(consigna_vel_tilt==0) halt(TILT);

                        previous_state_x = pgen->curr_x;
                        previous_state_y = pgen->curr_y;

                        previous_consigna_vel_pan = consigna_vel_pan;
                        previous_consigna_vel_tilt = consigna_vel_tilt;
                       }

                       Ellip.MakeEllipticalOutline(newz,int(1.2*newz));
                       if (newx>15 && newx<I0->nrows-15 && newy>15 && newy<I0-
                       >ncols-15) Ellip.PrintEllipInImame(I0->data,newx,newy,I0->nrows,I0-
                       >ncols);
                       MbufPutColor(M->MilImatge1,M_RGB24+M_PACKED,
                       M_ALL_BAND, I0->data);
                       MimResize(M->MilImatge1, M->MilImatge1_disp, pgen-
                       >ZoomX_Im_to_Disp, pgen->ZoomY_Im_to_Disp, M_DEFAULT);
                }

                if (cont>seq->NumImages){
                    pgen->parar_el_thread_del_tracker=1;
                    pgen->curr_x=pgen->XRedim;
                    pgen->curr_y=pgen->YRedim;
                    pgen->curr_z=12; //Tamaño inicial de la elipse
                }

                t2=clock();
                fps=CLOCKS_PER_SEC/(double(t2-t1));
                sprintf(cad, " x:%ld, y:%ld, vpan:%ld, vtilt:%ld, best_score:%ld,fps:%lf",pgen-
                >curr_x,pgen->curr_y,(long)consigna_vel_pan, (long)consigna_vel_tilt, pgen-
                >best_score, fps);
                button->SetWindowText(cad);

                sprintf(cad, "fps:%lf  best_score:%ld",fps,best_score);
        }

         if (pt->JA_CONNECTAT==1) halt(ALL);
        _endthread();

        delete [] I0;
}

void CTrackingDlg::OnButtonSearch()
{
        CToolBar* pBar = &m_wndToolBar;
        UINT iButtonID, iButtonStyle;
        int  iButtonImage;
        pBar->GetButtonInfo(4, iButtonID, iButtonStyle,iButtonImage);     //Button search
        pBar->SetButtonInfo(4, iButtonID, TBBS_DISABLED ,iButtonImage);
        pBar->GetButtonInfo(6, iButtonID, iButtonStyle,iButtonImage);     //Button stop
        pBar->SetButtonInfo(6, iButtonID, 0 ,iButtonImage);
        pBar->GetButtonInfo(8, iButtonID, iButtonStyle,iButtonImage);     //Button exit
```

```
	pBar->SetButtonInfo(8, iButtonID, TBBS_DISABLED ,iButtonImage);

	tts_salle("Estigues quiet que et buscare","Lenta","");

	pgen->parar_thread_search=0;
	_beginthread(Myfunction_search_head,0,&m_wndToolBar);
}

void Myfunction_search_head(void *resderes)
{
	double fps;
	int x,y,z,score;
	int pos_pan=0, pos_pan_max=900, pos_pan_min=-900, step=300;
	int step_turn=15;
	int borde_img=30;

	std::string str;
	ArTime start;

	clock_t t1,t2;

	//Posición inicial pantilt
	if (pt->JA_CONNECTAT ==1) {
		close_host_port(pt->COMstream);
		free(pt);
		pt=(pantilt*)malloc(sizeof(pantilt));
		pt->JA_CONNECTAT=1;
		pt->status=NO_ERROR;
		pt->status=LoadParam(".\\","param_PTU.txt",pt->param_pt,pt->status);
		strcpy(pt->COMportName,COM_PANTILT); //COM1

		pt->COMstream = open_host_port(pt->COMportName);

		if (pt->COMstream != PORT_NOT_OPENED) {
			pt->status=Settings(pt->param_pt,pt->status);
			pos_pan=-300;
			MoveToPositionPan(pt->COMstream, pt->param_pt, pos_pan, pt-
					>status);
			MoveToPositionTilt(pt->COMstream, pt->param_pt, -400, pt-
					>status);
		}
	}

	// the serial connection (robot)
	ArSerialConnection serConn;
	// robot
	ArRobot robot;

	//Para obtener la posición del robot
	//ArPose pose;

	// mandatory init
```

```
Aria::init();

//modify the next line if you're not using the first serial port
//to talk to your robot
serConn.setPort(COM_ROBOT); //COM2
robot.setDeviceConnection(&serConn);

//try to connect, if we fail exit
if (!robot.blockingConnect()) {
        MessageBox(0,"Could not connect to robot... exiting\n","",0);
        Aria::shutdown();
}

else {
        Tracker.LoadReFromFile("Histogram.bin");

        robot.comInt(ArCommands::ENABLE, 1);
        robot.comInt(ArCommands::SOUNDTOG, 0);

        // run the robot in its own thread, so it gets and processes packets and such
        robot.runAsync(false);


        while (!pgen->parar_thread_search) {
                t1=clock();
                for (int k=0; k<=7; k=k+1) { //Movimiento del pantilt
                        pgen->best_score=0;
                        MoveToPositionPan(pt->COMstream, pt->param_pt,
                                        pos_pan, pt->status);
                        robot.lock();
                        robot.setDeltaHeading(step_turn);
                        robot.unlock();

                        for (int i =borde_img; i<I0->nrows - borde_img; i=i+15) {
                              for (int j =borde_img; j<I0->ncols - borde_img ; j=j+15) {
                                  Tracker.SearchForHead(I0-
                                                  >data,i,j,11,&x,&y,&z,&score);
                                  if  (score >  pgen->best_score) {
                                          pgen->best_score=score;
                                          pgen->curr_x=x; pgen->curr_y=y; pgen-
                                          >curr_z=z;
                                  }
                              }
                        }
                        if (pgen->best_score>THRESHOLD_HISTO) {
                                halt(ALL);
                                set_mode(SPEED_CONTROL_MODE,PTU_
                                PURE_VELOCITY_SPEED_CONTROL_MODE);
                                pgen->parar_thread_search = 1;
                                pgen->track=TRUE;
                                tts_salle("T'he trobat","Lenta","");
                                break;
```

```
                                }
                                if (pgen->parar_thread_search==1) break;
                                if (step>0 && pos_pan==pos_pan_max){ step=-300;
                                      step_turn=-15;}
                                if (step<0 && pos_pan==pos_pan_min){ step=300;
                                      step_turn=+15;}

                                pos_pan=pos_pan+step;
                                t2=clock();
                                fps=CLOCKS_PER_SEC/(double(t2-t1));
                        }
                }
        }
        //obtiene la posición del robot de acuerdo a los encoders
        //thrad=(robot.getTh())*3.1416/180;
        //printf("position: %lf, %lf, %lf\n",(robot.getX())/1000,(robot.getY())/1000,thrad);

        //exit
        ArUtil::sleep(2000);
        Aria::shutdown();

}
```