EPS
Escola Politècnica Superior
UdG

# Projecte/Treball Fi de Carrera

**Estudi:** Enginyeria Tècn. Ind. Electrònica Ind. Pla 2002

**Títol:** Millora experimental de les lleis de control predictiu, aplicades sobre la plataforma PRIM I, per el seguiment de trajectòries

**Document:** 1. Memòria. Annex A. Codi del programa

**Alumne:** Xavier Serra Serinya

**Director/Tutor:** Lluís Pacheco Valls
**Departament:** Arquitectura i Tecnologia de Computadors
**Àrea:** ATC

**Convocatòria** (mes/any): febrer / 2009 Volum 2/2

## ÍNDEX

## A. CODI DEL PROGRAMA

En aquest apartat s'hi inclourà el codi del programa per tal d'utilitzar el robot. El codi del programa es va obtenir amb l'ajuda de Jordi Ferrer, membre del col·lectiu d'Arquitectura i Tecnologia de Computadors (ATC) de la UDG.

### A.1. Codi MakeFile

Aquesta part de codi és la base per assemblar tots els subprogrames que utilitzem, tant en Linux com en Visual C++.

```
CFLAGS=-Wall -ansi -pedantic -O3
# -m32 -mtune=i386
ROBLIB=./RobLib
ROBINC=./RobLib

OBJ=mpc.o sortida.o simulacio.o temps.o robcom.o
OBJ_ROB=${ROBLIB}/robot.o ${ROBLIB}/lpt.o

mpc: ${OBJ} ${OBJ_ROB}
	cd ${ROBLIB} && make && cd ..
	g++ ${OBJ} ${OBJ_ROB} -o $@

mpc.o : mpc.cpp temps.h sortida.h simulacio.h robcom.h
	g++ -I${ROBINC} ${CFLAGS} -c $< -o $@

sortida.o : sortida.cpp sortida.h simulacio.h
	g++ ${CFLAGS} -c $< -o $@

simulacio.o : simulacio.cpp simulacio.h
	g++ ${CFLAGS} -c $< -o $@

temps.o: temps.cpp temps.h
	g++ ${CFLAGS} -c $< -o $@

robcom.o: robcom.cpp robcom.h ${ROBINC}/constants.h ${ROBINC}/robot.h
	g++ -I${ROBINC} ${CFLAGS} -c $< -o $@

clean:
	rm -f *~ $(OBJ)
	rm -f ${ROBLIB}/*~ ${OBJ_ROB}
```

### A.2. Codi mpc.cpp

Aquest és el programa principal, és la base de l'algorisme de control on estan integrades totes les subrutines i funcions que estan explicades a la memòria. Aquest codi és on podem prendre totes les decisions a l'hora d'utilitzar el robot.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

#include "temps.h"
#include "simulacio.h"
#include "sortida.h"
#include "robcom.h"

/* TODO:
  - Definir el fitxer de text configurió
    HoritzoPrediccio = 1, 2, 3
    PosicioTarget = x, y
    SimulacioCamera = 0

  1.  [X] Afegir 3 models.
  2.  [ ] Connectar al robot a més de simular.
  3.  [ ] Lectura de la posició target.

  4.  [ ] Definició del món i lectura del target a partir d'aquest món.
*/

// 1 o 0 depenent de si es vol fer servir el robot o simular.
const int SimularRobot = 0;

// 1 o 0 depenent de si es vol rotar o girar de manera normal el robot
const int Rotacio = 0;

// 1 si es vol fer servir mètode valor òptim
// 0 si es vol fer servir mètode gradient
const int ErrorMinim = 1;

// Temps (ms) de resposta del robot a les consignes (duració d'una iteració)
const int TempsResposta = 100;

const char *CONSIGNES = "Consignes";
const char *ERRORS = "Errors";
const char *MATLAB = "matlab";

double gTDret[2],gTEsq[2];
double gXPos, gYPos;
double gDemora[2];
long int ticsIniciDre;
long int ticsIniciEsq;

FILE *obrirFitxer ( const char *Nom );

int calculaModel ( double Velocitat );

void InicialitzaTaules ( int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                         double TaulaErrors[INC_CON_MAX][INC_CON_MAX] );

void llegirVariables ( TPasIteracio TaulaIteracions[K_MAX],
                       double TaulaXd[4], double TaulaYd[4],
                       double &xd, double &yd, double &od,
                       double &A, double &B, double &C, int &np );

void novaTrajectoria ( const double x, const double y, const double xd, const
double yd,
                       double &A, double &B, double &C, double &od );

                       double grausARadiants ( double Alpha );

int quadrant ( const double od, const double o );

void initConsignesErrors ( const TPasIteracio TaulaIteracions[K_MAX],
```

```
                          int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                          double TaulaErrors[INC_CON_MAX][INC_CON_MAX], const int
NumIteracio,
                          const int q, const int Correccio );

void minimaDiferencia ( const TPasIteracio TaulaIteracions[K_MAX],
                        int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                        double TaulaErrors[INC_CON_MAX][INC_CON_MAX],
                        int &fm, int &cm, const double xd, const double yd,
                        const double od, const double A, const double B,
                        const double C, const int NumIteracio, int &Simulacions,
                        int puntactual );

double Simulacio ( const TPasIteracio TaulaIteracions[K_MAX],
                   TPasSimulacio   TaulaSimulacio[HOR_PRED+1], const int AUE,
                   const int AUD, const double xd, const double yd,
                   const double od, const int FLAG, const int NumIteracio );

void Calcula_x_y_o ( const TPasIteracio TaulaIteracions[K_MAX],
                     const int NumIteracio, const int AUE, const int AUD,
                     TPasSimulacio TaulaSimulacio[HOR_PRED+1] );

double Calcul_Ard ( double Ard, double UD );

double Calcul_Are ( double Are, double UE );

double Factor_Ard ( double UD );

double Factor_Are ( double UE );

double Calcul_x ( double x, double o, double Are, double Ard );

double Calcul_y ( double y, double o, double Are, double Ard );

double Calcul_o ( double o, double Are, double Ard );

double normalitzaAngle ( double O );

void modelRobot ( TPasIteracio TaulaIteracions[K_MAX], const int AUE,
                  const int AUD, const int NumIteracio );

double sumaDiferencies2 ( double x0, double y0, double x, double y );

double sumaDifAngles ( const double o, const double od );

double DifAngles ( const double x, const double y, const double o, const double xd,
                   const double yd );

double relacioIncrements ( const double xd, const double yd, const double od,
                           const TPasIteracio TaulaIteracions[K_MAX],
                           const TPasSimulacio TaulaSimulacio[HOR_PRED+1],
                           const  int k );

double distanciaTrajectoria ( const double x, const double y, const double a,
                              const double b, const double c );

void filtreKalman ( TPasIteracio TaulaIteracions[K_MAX],
                    TPasKalman TaulaKalman[K_MAX], const int k );

int comprovaError ( const TPasIteracio TaulaIteracions[K_MAX], const int i,
                    const int np, const int k );

unsigned char convertirPWM ( const int Consigna );

void RobotReal( TPasIteracio TaulaIteracions[K_MAX], const int NumIteracio);


/* Cicle principal */
```

```
int main ( void )
{
  FILE *fd, *fd1, *fd2;
  int Final, Correccio, fm, cm, k, Simulacions, np, i, K, q, Orientat;
  double odif, oref, DifTot;
  double A, B, C;
  double xd, yd, od;
  os_TIME TempsI, TempsF;
  int TempsT;

  // Historic
  TPasIteracio TaulaIteracions[K_MAX];
  TPasKalman TaulaKalman[K_MAX];
  /* Taula d'error per cada una de les possibles consignes */
  double TaulaErrors[INC_CON_MAX][INC_CON_MAX];
  /* Taula amb tots les possibles consignes al voltant de la consigna actual */
  int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2];

  double TaulaXd[5];
  double TaulaYd[5];


  //assegurem la conexio amb el robot, que els motors estiguin parats i llegim
encoders
  if ( !SimularRobot )
  {
    if ( Obrir_LPT ( 1 ) != 0 )
    {
      fprintf ( stderr, "No s'ha pogut inicialitzar el robot (falten permisos per
escriure a LPTx)!\n" );
      exit ( -1 );
    }
    EnviarConsignaMotor ( ESQUERRE, 127);
    EnviarConsignaMotor ( DRET, 127);
    ticsIniciDre = Llegeix_Tics_Encoder(DRET);
    ticsIniciEsq = Llegeix_Tics_Encoder(ESQUERRE);
  }

  fd = obrirFitxer ( CONSIGNES );
  fd1 = obrirFitxer ( ERRORS );
  fd2 = obrirFitxer ( MATLAB );

  Correccio = fm = cm = Simulacions = 0;
  k = 0;
  DifTot = 0;

  /* definir variables inicials i finals */
  InicialitzaTaules ( TaulaConsignes, TaulaErrors );
  llegirVariables ( TaulaIteracions, TaulaXd, TaulaYd, xd, yd, od, A, B, C, np );
  escriurePlantejament ( fd, TaulaIteracions, xd, yd, od );

  os_GetTime ( &TempsI );

  escriureTaulaConsignes ( fd, TaulaConsignes, TaulaIteracions, fm, cm, k );
  escriureError ( fd1, TaulaConsignes, TaulaErrors, fm, cm, k );

  TaulaIteracions[k].Dif = sumaDiferencies2 ( TaulaIteracions[k].Pas.x,
                           TaulaIteracions[k].Pas.y, xd, yd );
  fprintf ( stderr, "\t%d\t%f\t%d\t%d\n", k, TaulaIteracions[k].Dif, fm, cm );

  TaulaIteracions[k].DistTrj = distanciaTrajectoria ( TaulaIteracions[k].Pas.x,
                               TaulaIteracions[k].Pas.y, A, B, C );

  /*Comprovar fi d'iteracions*/
  Final = comprovaError ( TaulaIteracions, 0, 0, k );

  k++;
```

```
for ( i=0; i < np; i++ )
{
  Orientat = 0;

  q = quadrant ( od, TaulaIteracions[k-1].Pas.o );

  while ( Final == 0 )
  {
    odif = sumaDifAngles ( TaulaIteracions[k-1].Pas.o, od );

    oref = grausARadiants(3);
    oref *= oref;

    if ( odif > oref )
    {
      if (Rotacio && ( Orientat == 0 )) Correccio = 1;
      novaTrajectoria ( TaulaIteracions[k-1].Pas.x, TaulaIteracions[k-1].Pas.y,
                        xd, yd, A, B, C, od );
      printf ( "%fx + %fy + %f = 0\n", A, B, C );
    }
    else
    {
      Correccio = 0;
      Orientat++;
    }

    /* Asignar valors a la taula d'increments i errors */
    initConsignesErrors ( TaulaIteracions, TaulaConsignes, TaulaErrors, k, q,
                          Correccio );

    /* Calculem el valor de UE i UD optim en el que obtenim minim error */
    minimaDiferencia ( TaulaIteracions, TaulaConsignes, TaulaErrors, fm, cm, xd,
                       yd, od, A, B, C, k, Simulacions, i );

    if ( !SimularRobot )
    {
      TaulaIteracions[k].Pas.UE = TaulaIteracions[k-1].Pas.UE +
                                  TaulaConsignes[fm][cm][0];
      TaulaIteracions[k].Pas.UD = TaulaIteracions[k-1].Pas.UD +
                                  TaulaConsignes[fm][cm][1];
      EnviarConsignaMotor ( ESQUERRE, TaulaIteracions[k].Pas.UE + 127);
      EnviarConsignaMotor ( DRET, TaulaIteracions[k].Pas.UD + 127 );

      //deixem transcorre el temps que ens quedi per arrobar als 100ms
      os_GetTime ( &TempsF );
      TempsT = os_TimeDiff ( &TempsF, &TempsI );

      fprintf ( stderr, "Temps transcorregut: %d ms", TempsT );

      if ( TempsT < TempsResposta )
        msleep ( TempsResposta - TempsT );

      os_GetTime ( &TempsI );

      if ( Rotacio && (Orientat == 1)) //posem els motors a zero un cop orientat
      {
        TaulaIteracions[k].Pas.UE = 0;
        TaulaIteracions[k].Pas.UD = 0;
        EnviarConsignaMotor ( ESQUERRE, 127);
        EnviarConsignaMotor ( DRET, 127);
      }
      RobotReal ( TaulaIteracions, k );
    }
    else
    {
      if ( Rotacio && (Orientat == 1)) //posem els motors a zero un cop orientat
      {
        modelRobot ( TaulaIteracions, 0, 0, k );
```

```
          TaulaIteracions[k].Pas.UE = 0;
          TaulaIteracions[k].Pas.UD = 0;
        }
        else
          modelRobot ( TaulaIteracions, TaulaConsignes[fm][cm][0],
TaulaConsignes[fm][cm][1], k );
      }
      filtreKalman ( TaulaIteracions, TaulaKalman, k );

      escriureTaulaConsignes ( fd, TaulaConsignes, TaulaIteracions, fm, cm, k );
      escriureError ( fd1, TaulaConsignes, TaulaErrors, fm, cm, k );

      TaulaIteracions[k].Dif = sumaDiferencies2 ( TaulaIteracions[k].Pas.x,
                               TaulaIteracions[k].Pas.y, xd, yd );
      fprintf ( stderr, "\t%d\t%f\t%d\t%d\n", k, TaulaIteracions[k].Dif, fm, cm );

      TaulaIteracions[k].DistTrj = distanciaTrajectoria ( TaulaIteracions[k].Pas.x,
                                   TaulaIteracions[k].Pas.y, A, B, C );
      DifTot = DifTot + fabs(TaulaIteracions[k].DistTrj);

      /* Comprovar fi d'iteracions */
      Final = comprovaError ( TaulaIteracions, i, np, k );

      k++;
    }

    if ( Final == 1 )
    {
      fprintf ( stderr, "NOU PUNT\n" );

      xd = TaulaXd[i+1];
      yd = TaulaYd[i+1];

      /*calcul de trajectoria amb coordenades aboslutes*/
      novaTrajectoria ( TaulaXd[i], TaulaYd[i], xd, yd, A, B, C, od );
      /*calcul de trajectoria amb coordenades relatives*/
      //novaTrajectoria ( TaulaIteracions[k-1].Pas.x, TaulaIteracions[k-1].Pas.y,
                          xd, yd, A, B, C, od );

      printf ( "%fx+%fy+%f=0\n", A, B, C );
      printf ( "od = %6.3g\n", od );

      if ( !SimularRobot )
      {
        EnviarConsignaMotor ( ESQUERRE, 127);
        EnviarConsignaMotor ( DRET, 127);
        RobotReal ( TaulaIteracions, k );
        TaulaIteracions[k].Pas.UE = 0;
        TaulaIteracions[k].Pas.UD = 0;
      }
      else
      {
        modelRobot ( TaulaIteracions, 0, 0, k );  // primer el model robot
        TaulaIteracions[k].Pas.UE = 0;            // si ho declarem abans no afecta
                                                  // i no keda a 0
        TaulaIteracions[k].Pas.UD = 0;
      }
      filtreKalman ( TaulaIteracions, TaulaKalman, k );

      escriureTaulaConsignes ( fd, TaulaConsignes, TaulaIteracions, fm, cm, k );
      escriureError ( fd1, TaulaConsignes, TaulaErrors, fm, cm, k );

      TaulaIteracions[k].Dif = sumaDiferencies2 ( TaulaIteracions[k-1].Pas.x,
                               TaulaIteracions[k-1].Pas.y, xd, yd );
      fprintf ( stderr, "\t%d\t%f\t%d\t%d\n", k, TaulaIteracions[k].Dif, fm, cm );

      Final = comprovaError ( TaulaIteracions, i, np, k  );
```

```
      k++;
    }
  }
  if ( Final == 2 )
    fprintf ( stderr, "Error! Nombre maxim d'iteracions excedit!\n" );

  fprintf ( stderr, "L'error es de %f\n", TaulaIteracions[k-1].Dif );
  fprintf ( stderr, "L'error Mig es de %f\n", DifTot );

  os_GetTime ( &TempsI );

  for ( K = 0; K < 5; K ++)
  {
    if ( !SimularRobot ) //donem 5 iteracions més per deixar actuar la inèrcia
    {
     TaulaIteracions[k].Pas.UE = 0;
     TaulaIteracions[k].Pas.UD = 0;
     EnviarConsignaMotor ( ESQUERRE, 127);
     EnviarConsignaMotor ( DRET, 127);

     os_GetTime ( &TempsF );
     TempsT = os_TimeDiff ( &TempsF, &TempsI );

     fprintf ( stderr, "Temps transcorregut: %d ms", TempsT );

     if ( TempsT < TempsResposta )
       msleep ( TempsResposta - TempsT );

     os_GetTime ( &TempsI );

     RobotReal ( TaulaIteracions, k );
    }
    else
    {
      modelRobot ( TaulaIteracions, 0, 0, k );
      TaulaIteracions[k].Pas.UE = 0;
      TaulaIteracions[k].Pas.UD = 0;
      filtreKalman ( TaulaIteracions, TaulaKalman, k );
    }
    k++;
  }
  matlab (fd2, k, TaulaIteracions, TaulaKalman );

  // Escriure la trajectòria a la pantalla
  escriureTrajectoria ( stdout, k, TaulaIteracions, TaulaKalman );

  printf ( "El nombre de simulacions es de %d\n", Simulacions );

  fprintf ( stderr, "Fi del programa!\n" );

  fclose ( fd );
  fclose ( fd1 );
  fclose ( fd2 );

  return 0;
}


/* Obrir un fitxer i comprovar que s'hi pugui escriure. */
FILE *obrirFitxer ( const char *Nom )
{
  FILE *fd;

  if ( !( fd = fopen ( Nom, "w" ) ) )
  {
     printf ( "No s'ha pogut obrir el fitxer '%s' per escriure-hi!\n", Nom );
     exit ( -1 );
  }
```

```
  return fd;
}


/* Funcio per posar a 0 la taula consignes i errors, per evitar valors aleatoris */
void InicialitzaTaules ( int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                         double TaulaErrors[INC_CON_MAX][INC_CON_MAX] )
{
  int i,j;

  for ( i = 0; i < INC_CON_MAX; i++)
    for ( j = 0; j < INC_CON_MAX; j++)
    {
      TaulaConsignes[i][j][0] = TaulaConsignes[i][j][1] = 0;
      TaulaErrors[i][j] = 0.0;
    }
}


/* Funcio per inicialitzar les variables */
void llegirVariables ( TPasIteracio TaulaIteracions[K_MAX],
                       double TaulaXd[4], double TaulaYd[4],
                       double &xd, double &yd, double &od,
                       double &A, double &B, double &C, int &np )
{
  double xi, yi, oi;
  double Ax, Ay;
  int s, i;

  /* definir xd, yd i od amb un scanf, de moment les considerem 3,3,0.7 */
  printf ( " Prem 1-4 per escollir els punts de trajectoria\n" );
  scanf ( "%d", &np);

  for ( i = 0; i < np; i++)
  {
    printf ( " xd=" ); scanf ( "%lf", &xd );
    printf ( " yd=" ); scanf ( "%lf", &yd );

    TaulaXd[i] = xd;
    TaulaYd[i] = yd;
  }

  /* definir xi, yi i oi amb un scanf, de moment les considerem 0 */
  /* s=0; */
  printf ( " Prem 1 per definir els valors inicials\n" );
  printf ( " Prem 0 per pendre els valors per defecte 0,0,90º\n ");
  scanf ( "%d", &s );

  if ( s == 0 )
  {
    xi = 0; yi = 0; oi = grausARadiants ( 90 );
    printf(" xi=%6.3g yi=%6.3g oi=%6.3g\n", xi, yi, oi );
  }
  else
  {
    printf ( " xi=" ); scanf ( "%lf", &xi );
    printf ( " yi=" ); scanf ( "%lf", &yi );
    printf ( " oi=" ); scanf ( "%lf", &oi );
    oi = grausARadiants ( oi );
  }

  /* passem els metres a u.a.l. i els graus a radiants */
  xd = TaulaXd[0];
  yd = TaulaYd[0];

  Ay = yd - yi;
  Ax = xd - xi;
  od = atan2 ( Ay, Ax );
```

```
  if ( od < 0 ) od = (PI-fabs(od))+PI;

  printf ( " od=%6.3g\n", od );

  TaulaIteracions[0].Pas.x = xi;
  TaulaIteracions[0].Pas.y = yi;
  TaulaIteracions[0].Pas.o = oi;
  TaulaIteracions[0].Pas.UE = 0;
  TaulaIteracions[0].Pas.UD = 0;
  TaulaIteracions[0].Pas.Are = 0;
  TaulaIteracions[0].Pas.Ard = 0;

  /* definir coeficients recta trajectòria rectilínia (A,B,C) */
  A = yd-yi;
  B = xi-xd;
  C = (xd*yi) - (xi*yd);

  printf ( "%fx + %fy + %f = 0\n", A, B, C );
}


void novaTrajectoria ( const double x, const double y, const double xd,
                       const double yd, double &A, double &B, double &C,
                       double &od )
{
 double Ax, Ay;

 Ax = xd - x;
 Ay = yd - y;
 od = atan2 (Ay,Ax);
 if(od<0) od = (PI-fabs(od))+PI;

 A = yd - y;
 B = x - xd;
 C = (xd * y) - (x * yd);
}

/* Funció per passar de graus a radiants */
double grausARadiants ( double Alpha )
{
  return Alpha * PI / 180.0;
}

/* funció per determinar el quadrant de direcció quan fem rotació */
int quadrant ( const double od, const double o )
{
  int q;
  double odif;

  odif = od - o;

  if ( od > o )
  {
    if ( odif > PI ) q = 1;
    else q = 2;
  }
  else
  {
    if ( fabs(odif) > PI ) q = 2;
    else q = 1;
  }

  return q;
}


/* Funció per definir taula_valors */
void initConsignesErrors ( const TPasIteracio TaulaIteracions[K_MAX],
```

```
                          int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                          double TaulaErrors[INC_CON_MAX][INC_CON_MAX],
                          const int NumIteracio, const int q,
                          const int Correccio )
{
  int AE, AD;
  int AUEmin, AUDmin;
  int i, j;

  if ( Correccio == 1 )
  {
    if (q == 1)
    {
      AUEmin = -TaulaIteracions[NumIteracio-1].Pas.UE / INC_MAX;
      AUDmin = TaulaIteracions[NumIteracio-1].Pas.UD / INC_MAX;

      /* Ull, si AUEmin o AUDmin són 0, pot ser que no ens deixi mai parar
         motors */
      if ( AUEmin == 0 && TaulaIteracions[NumIteracio-1].Pas.UE > 0 )
        AUEmin = -1;

      if ( AUDmin == 0 && TaulaIteracions[NumIteracio-1].Pas.UD > 0 )
        AUDmin = 1;

      /* Caldrà ara limitar les consignes a 0 en simulació */
      for ( i = 0; i < INC_CON_MAX; i++ )
      {
        AE = AUEmin + i;

        for ( j = 0; j < INC_CON_MAX; j++ )
        {
          AD = AUDmin + j;
          TaulaConsignes[i][j][0] = AE;
          TaulaConsignes[i][j][1] = -AD;
          /* Apliquem Zona morta per consignes inferiors a 15 */
          if ( ( (TaulaIteracions[NumIteracio-1].Pas.UE + AE) <=15 ) &&
             ( fabs(TaulaIteracions[NumIteracio-1].Pas.UD + fabs (AD)) <= 15))
             {
             TaulaConsignes[i][j][0] = 0;
             TaulaConsignes[i][j][1] = 0;
             }

             /* Apliquem Zona morta per consignes esq. inferiors a 20 i consignes
dreta inferiors a 5 */
          if ( ( (TaulaIteracions[NumIteracio-1].Pas.UE + AE) <=20 ) &&
             ( fabs(TaulaIteracions[NumIteracio-1].Pas.UD + fabs (AD)) <= 5))
             {
             TaulaConsignes[i][j][0] = 0;
             TaulaConsignes[i][j][1] = 0;
             }

          /* Apliquem Zona morta per consignes esq. inferiors a 5 i consignes dreta
inferiors a 20 */
          if ( ( (TaulaIteracions[NumIteracio-1].Pas.UE + AE) <=5 ) &&
             ( fabs(TaulaIteracions[NumIteracio-1].Pas.UD + fabs (AD)) <= 20))
             {
             TaulaConsignes[i][j][0] = 0;
             TaulaConsignes[i][j][1] = 0;
             }

          TaulaErrors[i][j] = 0.0;
        }
      }
    }
    if (q == 2)
    {
      AUEmin = TaulaIteracions[NumIteracio-1].Pas.UE / INC_MAX;
      AUDmin = -TaulaIteracions[NumIteracio-1].Pas.UD / INC_MAX;
```

```
      /* Ull, si AUEmin o AUDmin són 0, pot ser que no ens deixi mai parar motors
*/
      if ( AUEmin == 0 && TaulaIteracions[NumIteracio-1].Pas.UE > 0 )
        AUEmin = 1;

      if ( AUDmin == 0 && TaulaIteracions[NumIteracio-1].Pas.UD > 0 )
       AUDmin = -1;

      /* Caldrà ara limitar les consignes a 0 en simulació */
      for ( i = 0; i < INC_CON_MAX; i++ )
      {
        AE = AUEmin + i;

        for ( j = 0; j < INC_CON_MAX; j++ )
        {
          AD = AUDmin + j;
          TaulaConsignes[i][j][0] = -AE;
          TaulaConsignes[i][j][1] = AD;
          /* Apliquem Zona morta per consignes inferiors a 15 */
          if ( ( fabs(TaulaIteracions[NumIteracio-1].Pas.UE + fabs(AE)) <= 15 ) &&
             ((TaulaIteracions[NumIteracio-1].Pas.UD + AD) <= 15))
          {
          TaulaConsignes[i][j][0] = 0;
          TaulaConsignes[i][j][1] = 0;
          }
          /* Apliquem Zona morta per consignes esq. inferiors a 20 i consignes
          dreta inferiors a 5 */

          if ( ( fabs(TaulaIteracions[NumIteracio-1].Pas.UE + fabs(AE)) <= 20 ) &&
             ((TaulaIteracions[NumIteracio-1].Pas.UD + AD) <= 5))
          {
          TaulaConsignes[i][j][0] = 0;
          TaulaConsignes[i][j][1] = 0;
          }

          /* Apliquem Zona morta per consignes esq. inferiors a 5 i consignes dreta
             inferiors a 20 */

          if ( ( fabs(TaulaIteracions[NumIteracio-1].Pas.UE + fabs(AE)) <= 5 ) &&
             ((TaulaIteracions[NumIteracio-1].Pas.UD + AD) <= 20))
          {
          TaulaConsignes[i][j][0] = 0;
          TaulaConsignes[i][j][1] = 0;
          }
          TaulaErrors[i][j] = 0.0;
        }
      }
    }
  }
  else
  {
    AUEmin = -TaulaIteracions[NumIteracio-1].Pas.UE / INC_MAX;
    AUDmin = -TaulaIteracions[NumIteracio-1].Pas.UD / INC_MAX;

    /* Ull, si AUEmin o AUDmin són 0, pot ser que no ens deixi mai parar motors */
    if ( AUEmin == 0 && TaulaIteracions[NumIteracio-1].Pas.UE > 0 )
      AUEmin = -1;

    if ( AUDmin == 0 && TaulaIteracions[NumIteracio-1].Pas.UD > 0 )
     AUDmin = -1;

    /* Caldrà ara limitar les consignes a 0 en simulació */
    for ( i = 0; i < INC_CON_MAX; i++ )
    {
      AE = AUEmin + i;

      for ( j = 0; j < INC_CON_MAX; j++ )
      {
```

```
        AD = AUDmin + j;
        TaulaConsignes[i][j][0] = AE;
        TaulaConsignes[i][j][1] = AD;
        /* Apliquem Zona morta per consignes inferiors a 15 */
        if ( ( (TaulaIteracions[NumIteracio-1].Pas.UE + AE) <= 15 ) &&
           ( ( TaulaIteracions[NumIteracio-1].Pas.UD + AD) <= 15 ))
        {
        TaulaConsignes[i][j][0] = 0;
        TaulaConsignes[i][j][1] = 0;
        }
        /* Apliquem Zona morta per consignes esq. inferiors a 20 i consignes dreta
inferiors a 5 */

          if ( ( (TaulaIteracions[NumIteracio-1].Pas.UE + AE) <= 20 ) &&
             ( (TaulaIteracions[NumIteracio-1].Pas.UD + AD) <= 5))
          {
          TaulaConsignes[i][j][0] = 0;
          TaulaConsignes[i][j][1] = 0;
          }

          /* Apliquem Zona morta per consignes esq. inferiors a 5 i consignes dreta
inferiors a 20 */

          if ( ( (TaulaIteracions[NumIteracio-1].Pas.UE + AE) <= 5 ) &&
             ( (TaulaIteracions[NumIteracio-1].Pas.UD + AD) <= 20))
          {
          TaulaConsignes[i][j][0] = 0;
          TaulaConsignes[i][j][1] = 0;
          }


        TaulaErrors[i][j] = 0.0;
      }
    }
  }
}


/* Cercar la consigna òptima */
void minimaDiferencia ( const TPasIteracio TaulaIteracions[K_MAX],
                        int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                        double TaulaErrors[INC_CON_MAX][INC_CON_MAX],
                        int &fm, int &cm, const double xd, const double yd,
                        const double od, const double A, const double B,
                        const double C, const int NumIteracio, int &Simulacions,
                        int puntactual )
{

  // Historic
  TPasSimulacio TaulaSimulacio[HOR_PRED+1];

  double MinDif, odif, m1, m2, q1, q12, q2, q23, q3, q34, q4, q41, condicio,
         errorAccOrientacio, errorAccDistTraj;
  int f, c, k, distancia[5], bucle, f1, c1, q, i;
  int b, Fi, Ff, Ci, Cf;

  MinDif = 10e23;
  fm = cm = -1;
  k = NumIteracio;

  if ( ErrorMinim )
  {
    b = 1;
    Fi = Ci = 0;
    Ff = Cf = INC_CON_MAX -1;
  }
  else
  {
```

```
  distancia[0]=(int)INC_CON_MAX/4;
  distancia[1]=(int)INC_CON_MAX/8;
  distancia[2]=(int)INC_CON_MAX/20;
  distancia[3]=(int)INC_CON_MAX/20;
  distancia[4]=(int)INC_CON_MAX/INC_CON_MAX;

  f1 = c1 =(int)INC_CON_MAX/2;

  b = 5;
  Fi = f1 - 1;
  Ci = c1 - 1;
  Ff = f1 + 1;
  Cf = c1 + 1;
}

odif = sumaDifAngles ( TaulaIteracions[k-1].Pas.o, od );

condicio = grausARadiants (3);
condicio *= condicio;

for ( bucle = 0; bucle < b; bucle++ )
{
  for ( f = Fi; f <= Ff; f++  )
  {
    for ( c = Ci; c <= Cf; c++ )
    {
      Calcula_x_y_o ( TaulaIteracions, NumIteracio, TaulaConsignes[f][c][0],
       TaulaConsignes[f][c][1], TaulaSimulacio );
      //entra en funcionament la funció de costos error acumulat de la diferència
      //d'orientació
       errorAccOrientacio = 0;

          for (i = 1; i < HOR_PRED+1; i++)
          {
              errorAccOrientacio = errorAccOrientacio +
              DifAngles (TaulaSimulacio[i].x, TaulaSimulacio[i].y,
              TaulaSimulacio[i].o, xd, yd);
          }

          errorAccOrientacio = errorAccOrientacio / HOR_PRED;

      if ((fabs(xd-TaulaIteracions[k].Pas.x))/(fabs(yd-TaulaIteracions[k].Pas.y))
           <= 0.1)
      {

       TaulaErrors[f][c] = ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
        TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
        /* ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
           distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, A, B, C ))+ */
       ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*( errorAccOrientacio/0.025)) ;
      }

       if ((0.1 < fabs(xd-TaulaIteracions[k].Pas.x)/fabs(yd-
          TaulaIteracions[k].Pas.y) <= 0.48) && (xd <= TaulaIteracions[k].Pas.x))
            {

       TaulaErrors[f][c] = ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
       TaulaSimulacio[HOR_PRED].y, xd, yd )/4)+
       /* ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
            distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, A, B, C ))+*/
       ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
        TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*( errorAccOrientacio /0.025)) ;
      }
```

```
         if ((0.48 < fabs(xd-TaulaIteracions[k].Pas.x)/fabs(yd-
         TaulaIteracions[k].Pas.y)) && (xd <= TaulaIteracions[k].Pas.x))
        {

         TaulaErrors[f][c] =
        ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, xd, yd )/4)+
         /* ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
         distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, A, B, C ))+*/
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
                ( errorAccOrientacio /0.025)) ;
        }

         if ((0.1 < fabs(xd-TaulaIteracions[k].Pas.x)/(fabs(yd-
             TaulaIteracions[k].Pas.y)) <= 0.48) &&
             (xd >  TaulaIteracions[k].Pas.x))
        {

         TaulaErrors[f][c] =( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
         /* ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
            distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, A, B, C ))+*/
            ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*( errorAccOrientacio /0.025)) ;
        }

         if ((0.48 < fabs(xd-TaulaIteracions[k].Pas.x)/(fabs(yd-
         TaulaIteracions[k].Pas.y))) && (xd > TaulaIteracions[k].Pas.x))
        {

         TaulaErrors[f][c] =
        ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, xd, yd )/4)+
        /* ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
                distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, A, B, C ))+*/
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/4)*
                ( errorAccOrientacio /0.025)) ;
        }

        else
         {

         TaulaErrors[f][c] =( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
        /* ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
           distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, A, B, C ))+*/
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*( errorAccOrientacio /0.025)) ;
        }
// entra en funcionament funció de costos error acumulat de la distància a la
//trajectòria
         /* if ( odif > condicio )
          {
            TaulaErrors[f][c] =
              sumaDifAngles ( TaulaSimulacio[HOR_PRED].o, od );
          }
          else
          {
```

```
            errorAccDistTraj = 0;

            for (i = 1; i < HOR_PRED+1; i++)
            {
                errorAccDistTraj = errorAccDistTraj +
                distanciaTrajectoria (TaulaSimulacio[i].x, TaulaSimulacio[i].y, A,
                                      B, C );
            }

            errorAccDistTraj = errorAccDistTraj / HOR_PRED;



            TaulaErrors[f][c] =
        ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
               errorAccDistTraj )/*+( (sumaDiferencies2 (
TaulaSimulacio[HOR_PRED].x,
                TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
               ( DifAngles (TaulaSimulacio[HOR_PRED].x, TaulaSimulacio[HOR_PRED].y,
                 TaulaSimulacio[HOR_PRED].o, xd, yd )/0.025))*/ ;


          if ((fabs(xd-TaulaIteracions[k].Pas.x))/(fabs(yd-
              TaulaIteracions[k].Pas.y)) <= 0.1)
        {
         TaulaErrors[f][c] = ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*errorAccDistTraj )/*+
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
                ( DifAngles (TaulaSimulacio[HOR_PRED].x,
                TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
                yd  )/0.025))*/ ;

        }

         if ((0.1 < fabs(xd-TaulaIteracions[k].Pas.x)/fabs(yd-
             TaulaIteracions[k].Pas.y) <= 0.48) &&
            (xd <= TaulaIteracions[k].Pas.x))
        {
         TaulaErrors[f][c] =
        ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, xd, yd )/4)+
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/4)*
               errorAccDistTraj )/*+( (sumaDiferencies2 (
TaulaSimulacio[HOR_PRED].x,
               TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
               ( DifAngles (TaulaSimulacio[HOR_PRED].x, TaulaSimulacio[HOR_PRED].y,
                TaulaSimulacio[HOR_PRED].o, xd, yd )/0.025))*/ ;

        }

         if ((0.48 < fabs(xd-TaulaIteracions[k].Pas.x)/fabs(yd-
             TaulaIteracions[k].Pas.y)) && (xd <= TaulaIteracions[k].Pas.x))
        {
         TaulaErrors[f][c] =
        ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
            TaulaSimulacio[HOR_PRED].y, xd, yd )/4)*
               errorAccDistTraj )/*+
         ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
```

```
    TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
        ( DifAngles (TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
          yd )/0.025)) */;

}

 if ((0.1 < fabs(xd-TaulaIteracions[k].Pas.x)/(fabs(yd-
     TaulaIteracions[k].Pas.y)) <= 0.48) &&
     (xd > TaulaIteracions[k].Pas.x))
{
 TaulaErrors[f][c] =
( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
  TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
 ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
     TaulaSimulacio[HOR_PRED].y, xd, yd )/4)*
       errorAccDistTraj )/*+
 ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
     TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
        ( DifAngles (TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
          yd )/0.025))*/ ;

}

 if ((0.48 < fabs(xd-TaulaIteracions[k].Pas.x)/(fabs(yd-
     TaulaIteracions[k].Pas.y))) && (xd > TaulaIteracions[k].Pas.x))
{
 TaulaErrors[f][c] =
( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
  TaulaSimulacio[HOR_PRED].y, xd, yd )/4)+
 ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
    TaulaSimulacio[HOR_PRED].y, xd, yd )/4)*
       errorAccDistTraj )/*+
 ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
    TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
        ( DifAngles (TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
          yd )/0.025))*/ ;

}

else
 {
 TaulaErrors[f][c] =
( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
  TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
 ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
   TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
       errorAccDistTraj )/*+
 ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
     TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
        ( DifAngles (TaulaSimulacio[HOR_PRED].x,
         TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
         yd )/0.025))*/ ;

}
 // Habilitant i deshabilitant els factors corresponents podem utilitzar
 // funció de costos distància a la trajectòria o
 // funció de costos diferència d'orientació
 // tot hi que no n'hem de fer cas per que aquest tros de codi el vam
 // utilitzar per fer proves

   if (puntactual == 1)
   {

    TaulaErrors[f][c] =
( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
```

```
         TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
        ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
           distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, A, B, C ))+
           ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
             TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
               ( DifAngles (TaulaSimulacio[HOR_PRED].x,
              TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
              yd )/0.025)) ;
          }

          if (puntactual == 2)
          {

           TaulaErrors[f][c] =
      ( sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
        TaulaSimulacio[HOR_PRED].y, xd, yd )/1)+
       ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
          TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
              distanciaTrajectoria ( TaulaSimulacio[HOR_PRED].x,
              TaulaSimulacio[HOR_PRED].y, A, B, C ))+
       ( (sumaDiferencies2 ( TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, xd, yd )/1)*
              ( DifAngles (TaulaSimulacio[HOR_PRED].x,
           TaulaSimulacio[HOR_PRED].y, TaulaSimulacio[HOR_PRED].o, xd,
           yd )/0.025)) ;
         }
      }*/

      if ( ErrorMinim ) /* Eleccio Error, metode Error Minin */
      {
        if ( TaulaErrors[f][c] < MinDif )
        {
          MinDif = TaulaErrors[f][c];
          fm = f;
          cm = c;
        }
      }

      Simulacions ++;
    }
  }
  if ( !ErrorMinim ) /* Eleccio Error, metode gradient */
  {
    if ( bucle == 4 )
    {
      q1  = TaulaErrors[f1-1][c1+1];
      q12 = TaulaErrors[f1-1][c1];
      q2  = TaulaErrors[f1-1][c1-1];
      q23 = TaulaErrors[f1][c1-1];
      q3  = TaulaErrors[f1+1][c1-1];
      q34 = TaulaErrors[f1+1][c1];
      q4  = TaulaErrors[f1+1][c1+1];
      q41 = TaulaErrors[f1][c1+1];

      m1 = q1; q = 1;
      m2 = m1 - q12; if ( m2 > 0 ) { m1 = q12; q = 12;}
      m2 = m1 - q2;  if ( m2 > 0 ) { m1 = q2;  q = 2;}
      m2 = m1 - q23; if ( m2 > 0 ) { m1 = q23; q = 23;}
      m2 = m1 - q3;  if ( m2 > 0 ) { m1 = q3;  q = 3;}
      m2 = m1 - q34; if ( m2 > 0 ) { m1 = q34; q = 34;}
      m2 = m1 - q4;  if ( m2 > 0 ) { m1 = q4;  q = 4;}
      m2 = m1 - q41; if ( m2 > 0 ) { m1 = q41; q = 41;}
    }
    else
    {
      q1=  TaulaErrors[f1-1][c1] + TaulaErrors[f1-1][c1+1] +
```

```
            TaulaErrors[f1][c1+1];
        q2=  TaulaErrors[f1-1][c1] + TaulaErrors[f1-1][c1-1] +
            TaulaErrors[f1][c1-1];
        q3=  TaulaErrors[f1+1][c1] + TaulaErrors[f1+1][c1-1] +
            TaulaErrors[f1][c1-1];
        q4=  TaulaErrors[f1+1][c1] + TaulaErrors[f1+1][c1+1] +
            TaulaErrors[f1][c1+1];

        m1 = q1; q = 1;
        m2 = m1 - q2; if ( m2 > 0 ) { m1 = q2; q = 2;}
        m2 = m1 - q3; if ( m2 > 0 ) { m1 = q3; q = 3;}
        m2 = m1 - q4; if ( m2 > 0 ) { m1 = q4; q = 4;}
      }

      switch ( q )
      {
        case 1:
          fm = f1 = f1 - distancia[bucle]; cm = c1 = c1 + distancia[bucle];
          break;

        case 2:
          fm = f1 = f1 - distancia[bucle]; cm = c1 = c1 - distancia[bucle];
          break;

        case 3:
          fm = f1 = f1 + distancia[bucle]; cm = c1 = c1 - distancia[bucle];
          break;

        case 4:
          fm = f1 = f1 + distancia[bucle]; cm = c1 = c1 + distancia[bucle];
          break;

        case 12:
          fm = f1 - distancia[bucle]; cm = c1;
          break;

        case 23:
          fm = f1; cm = c1 - distancia[bucle];
          break;

        case 34:
          fm = f1 + distancia[bucle]; cm = c1;
          break;

        case 41:
          fm = f1; cm = c1 + distancia[bucle];
          break;
      }
      Fi = f1 - 1;
      Ci = c1 - 1;
      Ff = f1 + 1;
      Cf = c1 + 1;
    }
  }
}

/* Funció per calcular els valors de x,y,teta(o) per cada simulació de k */
void Calcula_x_y_o ( const TPasIteracio TaulaIteracions[K_MAX], const int
NumIteracio,
                     const int AUE, const int AUD, TPasSimulacio
TaulaSimulacio[HOR_PRED+1] )
{
  int i;

  /* El valor anterior i 2 valors anteriors */
  TaulaSimulacio[0].x = TaulaIteracions[NumIteracio-1].Pas.x;
  TaulaSimulacio[0].y = TaulaIteracions[NumIteracio-1].Pas.y;
  TaulaSimulacio[0].o = TaulaIteracions[NumIteracio-1].Pas.o;
```

```
  TaulaSimulacio[0].Are = TaulaIteracions[NumIteracio-1].Pas.Are;
  TaulaSimulacio[0].Ard = TaulaIteracions[NumIteracio-1].Pas.Ard;

  /* UE i UD 3 valors d'increment i dos constants */
  /* Cal notar que el model dependra de U(k-1), per tant, la consigna anirà
retrassada un període */

  for ( i = 0; i < HOR_PRED; i++ )
  {
    TaulaSimulacio[i].UE = TaulaIteracions[NumIteracio-1].Pas.UE +
     HoritzoControl[i] * AUE;
    TaulaSimulacio[i].UD = TaulaIteracions[NumIteracio-1].Pas.UD +
     HoritzoControl[i] * AUD;
  }

  for ( i = 1; i < HOR_PRED+1; i++ )
  {
    TaulaSimulacio[i].Ard = Calcul_Ard ( TaulaSimulacio[i-1].Ard,
    TaulaSimulacio[i-1].UD );
    TaulaSimulacio[i].Are = Calcul_Are ( TaulaSimulacio[i-1].Are,
    TaulaSimulacio[i-1].UE );

    /* Càlcul dels valors per la variable x */
    TaulaSimulacio[i].x =
             Calcul_x ( TaulaSimulacio[i-1].x, TaulaSimulacio[i-1].o,
                        TaulaSimulacio[i].Ard, TaulaSimulacio[i].Are);

    /* Càlcul dels valors per la variable y */
    TaulaSimulacio[i].y =
             Calcul_y ( TaulaSimulacio[i-1].y, TaulaSimulacio[i-1].o,
                        TaulaSimulacio[i].Ard, TaulaSimulacio[i].Are);

    /* Càlcul dels valors per la variable teta (o) */
    TaulaSimulacio[i].o =
             normalitzaAngle(Calcul_o ( TaulaSimulacio[i-1].o,
                             TaulaSimulacio[i].Ard, TaulaSimulacio[i].Are ));
  }
}

/* funció per escollir el model de velocitat en funció de la consigna*/
int calculaModel ( double Velocitat )
{
   int Res;

   if ( Velocitat < (RANG_VEL/3))
     Res = VEL_LENT;
   else
     if ( Velocitat < (RANG_VEL/2))
        Res = VEL_MIG;
     else
       Res = VEL_RAPID;

   return Res;
}

/* Calcul de l'increment roda dreta en funció del model del robot */
double Calcul_Ard ( double Ard, double UD )
{
  double Res;

  switch ( calculaModel ( UD ) )
  {
    case VEL_LENT:
      Res = 0.8007 * Ard + 0.0164 * UD;
      break;

      case VEL_MIG:
      Res = 0.7812 * Ard + 0.0202 * UD;
```

```
      break;

      case VEL_RAPID:
      Res = 0.7894 * Ard + 0.02 * UD;
      break;

    default:
      Res = 0;  // No pot pasar mai
      break;
  }
  return Res;
}

/* Calcul de l'increment roda esquerra en funció del model del robot */
double Calcul_Are ( double Are, double UE )
{
  double Res;

  switch ( calculaModel ( UE ) )
  {
    case VEL_LENT:
      Res = 0.8007 * Are + 0.0174 * UE;
      break;

    case VEL_MIG:
      Res = 0.7812 * Are + 0.0201 * UE;
      break;

    case VEL_RAPID:
      Res = 0.7867 * Are + 0.0194 * UE;
      break;

    default:
      Res = 0;  // No pot passar mai
      break;
  }
  return Res;
}

double Factor_Ard ( double UD )
{
  double Res;

  switch ( calculaModel ( UD ) )
  {
    case VEL_LENT:
      Res = 0.8007 * 0.8007;
      break;

      case VEL_MIG:
      Res = 0.7812 * 0.7812;
      break;

      case VEL_RAPID:
      Res = 0.7894 * 7894;
      break;

    default:
      Res = 0;  // No pot pasar mai
      break;
  }
  return Res;
}

double Factor_Are ( double UE )
{
  double Res;
```

```
  switch ( calculaModel ( UE ) )
  {
    case VEL_LENT:
      Res = 0.8007 * 0.8007;
      break;

    case VEL_MIG:
      Res = 0.7812 * 0.7812;
      break;

    case VEL_RAPID:
      Res = 0.7867 * 0.7867;
      break;

    default:
      Res = 0;  // No pot passar mai
      break;
  }
  return Res;
}

/* Calcul de la variable x */
double Calcul_x ( double x, double o, double Ard, double Are )
{
  return x + ((( Ard + Are) / 2) * cos (o) );
}

/* Calcul de la variable y */
double Calcul_y ( double y, double o, double Ard, double Are )
{
  return y + ((( Ard + Are) / 2) * sin (o) );
}

/* Calcul de la variable o */
double Calcul_o ( double o, double Ard, double Are )
{
  return o + (( Ard - Are) / L );
}

/* funció del filtre de Kalman */
void filtreKalman ( TPasIteracio TaulaIteracions[K_MAX], TPasKalman
TaulaKalman[K_MAX], const int k )
{
  /* Inicialitza Parametres */
  TaulaKalman[0].Are=0;
  TaulaKalman[0].ARE=0;
  TaulaKalman[0].Ard=0;
  TaulaKalman[0].ARD=0;
  TaulaKalman[0].KAre=0;
  TaulaKalman[0].KArd=0;
  TaulaKalman[0].pAre=0;
  TaulaKalman[0].pArd=0;
  TaulaKalman[0].PAre=0;
  TaulaKalman[0].PArd=0;

  /* Aplicacio filtre */
  TaulaKalman[k].Are = Calcul_Are ( TaulaKalman[k-1].ARE,
                       TaulaIteracions[k].Pas.UE );
  TaulaKalman[k].Ard = Calcul_Ard ( TaulaKalman[k-1].ARD,
                       TaulaIteracions[k].Pas.UD );

  TaulaKalman[k].pAre = (Factor_Are(TaulaIteracions[k].Pas.UE)*
                         TaulaKalman[k-1].PAre) + 0.01;
  TaulaKalman[k].pArd = (Factor_Ard(TaulaIteracions[k].Pas.UD)*
                         TaulaKalman[k-1].PArd) + 0.01;
  //El segon parametre 1 ha de ser el quadrat del primer parametre 1
  TaulaKalman[k].KAre = TaulaKalman[k].pAre * 1 / (0.001 +
                         1 * TaulaKalman[k].pAre);
```

```
  TaulaKalman[k].KArd = TaulaKalman[k].pArd * 1 / (0.01 +
                         1 * TaulaKalman[k].pArd);

  TaulaKalman[k].PAre = TaulaKalman[k].pAre - (TaulaKalman[k].KAre *1*
                        TaulaKalman[k].pAre);
  TaulaKalman[k].PArd = TaulaKalman[k].pArd - (TaulaKalman[k].KArd *1*
                        TaulaKalman[k].pArd);

  TaulaKalman[k].ARE = TaulaKalman[k].Are + TaulaKalman[k].KAre *
                       (TaulaIteracions[k].Pas.Are - 1 * TaulaKalman[k].Are);
  TaulaKalman[k].ARD = TaulaKalman[k].Ard + TaulaKalman[k].KArd *
                       (TaulaIteracions[k].Pas.Ard - 1 * TaulaKalman[k].Ard);

  TaulaIteracions[k].Pas.Are = TaulaKalman[k].ARE;
  TaulaIteracions[k].Pas.Ard = TaulaKalman[k].ARD;
}

/* Normalitzar un angle entre -PI i PI */
double normalitzaAngle ( double O )
{
  if ( O > 2*PI ) O = O - 2*PI;
  if ( O < 0 ) O = O + 2*PI;

  return O;
}

/* funció que simula el robot */
void modelRobot ( TPasIteracio TaulaIteracions[K_MAX], const int AUE,
                  const int AUD, const int NumIteracio )
{
  int k;

  k = NumIteracio;

  /* Càlcul del valor dels increments */
  TaulaIteracions[k].Pas.Ard = Calcul_Ard ( TaulaIteracions[k-1].Pas.Ard,
                               TaulaIteracions[k-1].Pas.UD );
  TaulaIteracions[k].Pas.Are = Calcul_Are ( TaulaIteracions[k-1].Pas.Are,
                               TaulaIteracions[k-1].Pas.UE );

  TaulaIteracions[k].Pas.x = Calcul_x ( TaulaIteracions[k-1].Pas.x,
                          TaulaIteracions[k-1].Pas.o,
                          TaulaIteracions[k].Pas.Ard, TaulaIteracions[k].Pas.Are );

  TaulaIteracions[k].Pas.y = Calcul_y ( TaulaIteracions[k-1].Pas.y,
                             TaulaIteracions[k-1].Pas.o,TaulaIteracions[k].Pas.Ard,
                             TaulaIteracions[k].Pas.Are );

  TaulaIteracions[k].Pas.o = normalitzaAngle ( Calcul_o (
                             TaulaIteracions[k-1].Pas.o,TaulaIteracions[k].Pas.Ard,
                             TaulaIteracions[k].Pas.Are ));

  TaulaIteracions[k].Pas.UE = TaulaIteracions[k-1].Pas.UE + AUE;
  TaulaIteracions[k].Pas.UD = TaulaIteracions[k-1].Pas.UD + AUD;
}


/* Calcula la suma de diferències al quadrat */
double sumaDiferencies2 ( double x0, double y0, double x, double y )
{
  double xd, yd;

  xd = fabs(x - x0); xd *= xd;
  yd = fabs(y - y0); yd *= yd;

  return xd + yd;
}
```

```
double sumaDifAngles ( const double o, const double od )
{
  double odif;

  odif = od - o;

  if (( o < PI/2 ) && ( odif > PI )) odif = - ((2*PI - od ) + o );
  if (( o > 3*PI/2 ) && ( fabs(odif) > PI )) odif = (2*PI - o ) + od;

  return odif*odif;
}

double DifAngles (const double x, const double y, const double o, const double xd,
                  const double yd )
{
  double od2, odif2;

  od2 = atan2(yd - y, xd - x);

  if (od2 < 0 ) od2 = (PI-fabs(od2))+PI;

  odif2= od2 - o;

  if (( o < PI/2 ) && ( odif2 > PI )) odif2 = - ((2*PI - od2 ) + o );
  if (( o > 3*PI/2 ) && ( fabs(odif2) > PI )) odif2 = (2*PI - o ) + od2;

  return odif2*odif2;

}

/* Calcular distància del punt (x, y) a la recta ax + by + c = 0 */
double distanciaTrajectoria ( const double x, const double y,
                              const double a, const double b, const double c )
{
  return ((a*x+b*y+c)*(a*x+b*y+c)) / (a*a+b*b);
}


/* Comprovar error o limit d'iteracions per saber si cal plegar d'iterar */
int comprovaError ( const TPasIteracio TaulaIteracions[K_MAX],
                    const int i, const int np, const int k )
{
  int Final;

  Final = 0;
  if (( TaulaIteracions[k].Dif <= ERROR_MAXIM ) || ( k > (i+1) * 200))
 /* S'arriba a l'error màxim permès */
// if ((fabs( TaulaIteracions[k].Dif) <fabs( TaulaIteracions[k-1].Dif))||fabs((od
//-TaulaIteracions.Pas.o)) <fabs(od - o)) funció de prova
   Final = 1;
  else
   if ( k >= K_MAX)    /* S'assoleixen el número màxim d'iteracions */
     Final = 2;

  //if (( k > 1) && ( Final == 0 ) && ( TaulaIteracions[k].Dif >
                     TaulaIteracions[k-1].Dif)) Final = 2;
  if (( Final == 1 ) && ( i == (np - 1))) Final = 2;

  return Final;
}

/* funció que convertiex les consignes simulades a consignes del robot */
unsigned char convertirPWM ( const int Consigna )
{
  return Consigna + 128;
}

/* funció aplicació del robot real */
```

```
void RobotReal ( TPasIteracio TaulaIteracions[K_MAX], const int NumIteracio )
{
  long int inc_tdret,inc_tesq;
  double inc;
  if(NumIteracio == 1)
  {
    gXPos=0;
    gYPos = 0;
    gDemora[0]=90;
    gDemora[1]=grausARadiants(90);
    gTDret[0]=0;
    gTDret[1]=0;
    gTEsq[0]=0;
    gTEsq[1]=0;
  }
  gTDret[1]=Llegeix_Tics_Encoder(DRET)-ticsIniciDre;
  gTEsq[1]=Llegeix_Tics_Encoder(ESQUERRE)-ticsIniciEsq; //s'ha de retocar.;

  inc_tdret=(long int)(gTDret[0]-gTDret[1]);
  if (inc_tdret<-30000) inc_tdret=65535+inc_tdret;       // Overflow dels tics cap
                                                         // endavant
  if (inc_tdret>30000)  inc_tdret=inc_tdret-65535;       // Overflow dels tics cap
                                                         // enrera

  inc_tesq=(long int)(gTEsq[0]-gTEsq[1]);
  if (inc_tesq<-30000) inc_tesq=65535+inc_tesq;          // Overflow dels tics cap
                                                         // endavant
  if (inc_tesq>30000)  inc_tesq=inc_tesq-65535;          // Overflow dels tics cap
                                                         // enrera

  gDemora[1]=gDemora[1] + atan2(((inc_tesq-
inc_tdret)*DIST_1_TIC),DIST_ENTRE_RODES);
  gDemora[0]=(gDemora[1]*180)/M_PI;

  inc=((inc_tdret+inc_tesq)/2)*DIST_1_TIC;
  gYPos=gYPos + inc * cos(gDemora[1]-M_PI_2);
  gXPos=gXPos + inc * sin(gDemora[1]-M_PI_2)*(-1);

  gTEsq[0]=gTEsq[1]; gTDret[0]=gTDret[1];

  /* Càlcul del valor dels increments */
  TaulaIteracions[NumIteracio].Pas.Are =
        inc_tdret *  DIST_1_TIC;//LlegirVelocitatMotor ( DRET )*4.5/255;
  if (TaulaIteracions[NumIteracio].Pas.Are > 5 )
 TaulaIteracions[NumIteracio].Pas.Are =
                                TaulaIteracions[NumIteracio-1].Pas.Are;
  TaulaIteracions[NumIteracio].Pas.Ard =
              inc_tesq * DIST_1_TIC;//LlegirVelocitatMotor ( ESQUERRE)*4.5/255;
  if (TaulaIteracions[NumIteracio].Pas.Ard > 5 )
TaulaIteracions[NumIteracio].Pas.Ard = TaulaIteracions[NumIteracio-1].Pas.Ard;

  TaulaIteracions[NumIteracio].Pas.x =  gXPos;

  TaulaIteracions[NumIteracio].Pas.y =  gYPos;

  TaulaIteracions[NumIteracio].Pas.o =  gDemora[1];
  printf("\nposicio actual:    x: %f      y: %f\n", gXPos, gYPos);

}
```

Tot seguit comentarem breument les instruccions bàsiques del programa.

Inicialment ens trobem les variables SimularRobot, Rotacio i ErrorMinim. La variable SimularRobot ens permet escollir entre utilitzar el robot (0) o només fer simulacions (1). La

variable rotació permetrà que a l'hora de girar, el robot faci una rotació sobre el seu propi eix z (1) o que efectuï el gir de manera normal (0). I la variable ErrorMinim permetrà que busquem l'error mínim aplicant el mètode valor òptim (1) o el mètode gradient (0). Escollirem una opció o una altra en funció de posar un 1 o un 0.

Si volem realitzar trajectòries variables de més d'un punt, tal com està configurat el programa només ens permetrà fer trajectòries de 4 punts com a màxim. Si en volguéssim fer més, hauríem de canviar a l'apartat d'Historics del programa principal el valor de la TaulaXd i TaulaYd, i escollir el valor que volem.

Un altra factor a escollir és la zona morta, com hem comentat a la memòria descriptiva, quan parlem de la zona morta ens referim a les consignes on el robot no respon. Per evitar que la zona morta aparegui, hem posat unes condicions perque quan es doni el cas, aquestes consignes no actuïn, substituint-les per consignes zero. Aquestes condicions les podem trobar a la funció initConsignesErrors, i les hem prefixat per; consignes esquerres i dretes inferiors o iguals a 15, per consignes esquerres inferiors o iguals a 20 i consignes dretes inferiors o iguals a 5, i per consignes esquerres inferiors o iguals a 5 i consignes dretes inferiors o iguals a 20. Canviant aquests valors podrem variar la zona morta en funció de la consigna que desitgem.

Per trobar les consignes òptimes que donin l'error mínim en cada període, hem d'anar a la funció minimaDiferencia. En aquesta funció podrem escollir qualsevol de les funcions de costos que hem comentat a la memòria descriptiva.

### A.3. Codi simulacio

Aquesta part del codi conté les variables constants que utilitzem en el programa principal mpc.cpp. Aquest codi està inclòs en dos fitxers: simulacio.h que conté definides totes les variables i simulacio.cpp que conté les variables respecte els horitzons de predicció.

#### A.3.1. simulacio.h

```
#ifndef __SIMULACIO_H__
#define __SIMULACIO_H__

/* Constants */
#define PI            3.1415926535
/* Rang de velocitats */
#define RANG_VEL      123
```

```
/* Amplada del robot */
#define L             50.265
/* Número màxim d'iteracions */
#define K_MAX         800
/* Error màxim permès */
#define ERROR_MAXIM   20
/* Horitz de prediccio */
#define HOR_PRED      5

#define VEL_LENT      0
#define VEL_MIG       1
#define VEL_RAPID     2

//extern const int HoritzoControl[HOR_PRED];
extern const int HoritzoControl[HOR_PRED];

/* Número màxim d'increments */
#define INC_MAX       3
/* Increment de consignes màxim */
#define INC_CON_MAX   ((int)(RANG_VEL / INC_MAX))

/* Tipus pas d'una simulació */
typedef struct SPasSimulacio
{
   double x;       // Posició
   double y;
   double o;
   int UE;         // Consigna
   int UD;
   double Are;     // Increments segons el model del robot
   double Ard;
} TPasSimulacio;

/* Tipus pas d'una iteració */
typedef struct SPasIteracio
{
   TPasSimulacio Pas;
   double Dif;
   double DistTrj;     // Error respecte la trajectòria que es vol
} TPasIteracio;


typedef struct SPasKalman
{
  double Are;
  double ARE;
  double Ard;
  double ARD;
  double KAre;
  double KArd;
  double pAre;
  double pArd;
  double PAre;
  double PArd;
}TPasKalman;

#endif
```

En aquest codi podrem modificar les variables per poder escollir el tipus d'horitzó de predicció. Ara mateix, està configurat per que actuï com a horitzó de predicció curt. Si volguéssim variar l'horitzó de predicció hauríem de variar les següents variables que ens mostra la Taula 1.

| Variable | Horitzó curt | Horitzó mig | Horitzó llarg |
|---|---|---|---|
| RANG_VEL | 123 | 125 | 125 |
| HOR_PRED | 5 | 8 | 10 |
| INC_MAX | 3 | 4 | 5 |

Taula 1. Variables en funció de l'horitzó de predicció

### A.3.2. simulacio.cpp

En funció de l'horitzó escollit habilitarem una de les tres opcions. Aquesta part de codi afecta directament a la funció "Calcula_x_y_o" del programa principal mpc.cpp, ja que en aquesta funció s'apliquen els increments de consigna per tal de captar els valors simulats que pot obtenir el robot al final del període de l'horitzó de predicció.

```
#include "simulacio.h"

/* Curt */
const int HoritzoControl[HOR_PRED] =
                    { 0, 1, 2, INC_MAX, INC_MAX };

/* Mig */
//const int HoritzoControl[HOR_PRED] =
                    { 0, 1, 2, 3, INC_MAX, INC_MAX, INC_MAX, INC_MAX };

/* Llarg */
//const int HoritzoControl[HOR_PRED] =
                    { 0, 1, 2, 3, 4, INC_MAX, INC_MAX, INC_MAX, INC_MAX, INC_MAX };
```

Tal com es mostra, ara mateix està configurat per que el robot actui amb l'horitzó de predicció curt.

## A.4. Codi temps

El codi temps està inclòs en dos tipus de fitxers. Un és temps.h, que declara les funcions referents al temps. I l'altre, temps.cpp, que té la funció de comptar el temps en ms per tal de poder-les aplicar al mpc.cpp, per poder fer el càlcul de temps (100ms) de cada període.

### A.4.1. temps.h

```
#ifndef __TEMPS_H__
#define __TEMPS_H__

#ifdef WIN32
  #include <windows.h>
#else
  #include <sys/types.h>
  #include <sys/time.h>
```

```
#endif

void msleep ( unsigned int msec );

#ifdef WIN32
  typedef DWORD os_TIME;
#else
  typedef struct timeval os_TIME;
#endif

// Obtenir una marca de temps a la variable entrada
void os_GetTime ( os_TIME *time );
// Calcular la diferència entre dos temps en ms
int os_TimeDiff ( os_TIME *time1, os_TIME *time2 );

#endif
```

### A.4.2. temps.cpp

```
#include "temps.h"

#ifdef WIN32
  #include <windows.h>
#else
  #include <unistd.h>
#endif

void msleep ( unsigned int msec )
{
#ifdef WIN32
  Sleep ( msec );
#else
  usleep ( msec * 1000 );
#endif
}

void os_GetTime ( os_TIME *time )
{
#ifdef WIN32
  *time = GetTickCount();
#else
  struct timezone tz;
  gettimeofday(time, &tz);
#endif
}

int os_TimeDiff ( os_TIME *time1, os_TIME *time2 )
{
#ifdef WIN32
  return *time1 - *time2;
#else
  return (int)((double)time1->tv_sec*1000 + ((double)time1->tv_usec)*1e-3 -
(double)time2->tv_sec*1000 - ((double)time2->tv_usec)*1e-3);
#endif
}
```

## A.5. Codi sortida

En aquesta part de codi ens trobarem totes les funcions considerades de sortides, les referents a les sortides de dades obtingudes del programa mpc.cpp, per exemple: escriure valors en un fitxer o representar-los per pantalla.

### A.5.1. sortida.h

Aquesta part de codi conté les declaracions de les funcions per realitzar les tasques comentades a l'apartat anterior.

```
#ifndef __SORTIDA_H__
#define __SORTIDA_H__

#include "simulacio.h"

/* Guardar valors xi, yi, oi, xd, yd, od al fitxer. */
void escriurePlantejament ( FILE *fd, const TPasIteracio TaulaIteracions[K_MAX],
                            const double xd, const double yd, const double od );

void escriureTaulaConsignes ( FILE *fd, int
TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                              const TPasIteracio TaulaIteracions[K_MAX],
                              const int fm, const int cm, int NumIteracio );

void escriureError ( FILE *fd,  int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                     const double TaulaErrors[INC_CON_MAX][INC_CON_MAX],
                     const int fm, const int cm, const int NumIteracio );

/* Escriure la trajectoria en un fitxer */
void escriureTrajectoria ( FILE *fd, const int NumIteracions,
                           const TPasIteracio TaulaIteracions[K_MAX],
                           const TPasKalman TaulaKalma[K_MAX] );

void matlab ( FILE *fd, const int NumIteracions,
              const TPasIteracio TaulaIteracions[K_MAX],
              const TPasKalman TaulaKalman[K_MAX] );

#endif
```

### A.5.2. sortida.cpp

En aquesta part de codi tenim el codi de les funcions del programa sortida.h, en aquest programa veurem el que queda escrit en els fitxers de cada funció.

```
#include <stdio.h>
#include "simulacio.h"
#include "sortida.h"

/* Guardar valors xi, yi, oi, xd, yd, od al fitxer. */
void escriurePlantejament ( FILE *fd, const TPasIteracio TaulaIteracions[K_MAX],
                            const double xd, const double yd, const double od )
```

```
{
  fprintf ( fd, "Plantejament del problema\n\n\n");

  fprintf ( fd, "Posicio inicial\n");
  fprintf ( fd, "xi=%6.3g\n", TaulaIteracions[0].Pas.x);// REL_UAL_CM );
  fprintf ( fd, "yi=%6.3g\n", TaulaIteracions[0].Pas.y);// REL_UAL_CM );
  fprintf ( fd, "oi=%6.3g\n\n\n", TaulaIteracions[0].Pas.o );

  fprintf ( fd, "Posicio desitjada\n" );
  fprintf ( fd, "xd=%6.3g\n", xd);// / REL_UAL_CM );
  fprintf ( fd, "yd=%6.3g\n", yd);// / REL_UAL_CM );
  fprintf ( fd, "od=%6.3g\n\n", od );
}

void escriureTaulaConsignes ( FILE *fd,
                              int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                              const TPasIteracio TaulaIteracions[K_MAX],
                              const int fm, const int cm, int NumIteracio )
{
  int i, j;

  /* Prova, impresio arxiu */
  fprintf ( fd,     "Pas número: %d\n", NumIteracio );

  fprintf ( fd, "Taula de consignes AUE:\n");
  for ( i = 0; i < INC_CON_MAX; i++ )
  {
    for ( j = 0; j < INC_CON_MAX; j++ )
      fprintf ( fd, "%4d\t", TaulaConsignes[i][j][0] );
    fprintf ( fd, "\n" );
  }

  fprintf ( fd, "\nTaula de consignes AUD:\n");
  for ( i = 0; i < INC_CON_MAX; i++ )
  {
    for ( j = 0; j < INC_CON_MAX; j++ )
      fprintf ( fd, "%4d\t", TaulaConsignes[i][j][1] );
    fprintf ( fd, "\n" );
  }
  fprintf ( fd, "\n" );
  fprintf ( fd,     "Pas número: %d\n", NumIteracio );
  fprintf ( fd, "Valor de les variables:\n");
  fprintf ( fd, "UEi=%4d\tUDi=%4d\n", TaulaIteracions[NumIteracio-1].Pas.UE,
            TaulaIteracions[NumIteracio-1].Pas.UD );
  fprintf ( fd, "AUE=%4d\tAUD=%4d\n", TaulaConsignes[fm][cm][0],
            TaulaConsignes[fm][cm][1] );
  fprintf ( fd, "UE =%4d\tUD =%4d\n\n", TaulaIteracions[NumIteracio].Pas.UE,
            TaulaIteracions[NumIteracio].Pas.UD );

  fprintf ( fd, "x=%3.5f\n", TaulaIteracions[NumIteracio].Pas.x);
  fprintf ( fd, "y=%3.5f\n", TaulaIteracions[NumIteracio].Pas.y);
  fprintf ( fd, "o=%3.3f\n\n", TaulaIteracions[NumIteracio].Pas.o );
}

void escriureError ( FILE *fd,  int TaulaConsignes[INC_CON_MAX][INC_CON_MAX][2],
                     const double TaulaErrors[INC_CON_MAX][INC_CON_MAX],
                     const int fm, const int cm, const int NumIteracio )
{
  int f, c;

  fprintf ( fd, "Pas número: %d\n\n", NumIteracio );

  fprintf ( fd, "            \t" );
  for ( c = 0; c < INC_CON_MAX; c++ )
    fprintf ( fd, "%12d\t", TaulaConsignes[0][c][1] );

  fprintf ( fd, "\n" );
```

```
  for ( f = 0; f < INC_CON_MAX; f++ )
  {
    fprintf ( fd, "%12d\t", TaulaConsignes[f][0][0] );

    for ( c = 0; c < INC_CON_MAX; c++ )
      fprintf ( fd, "%12.6g\t", TaulaErrors[f][c] );
    fprintf ( fd, "\n" );
  }

  fprintf ( fd, "Index    (Fila, Columna)=(%d,%d)\n", fm, cm );
  fprintf ( fd, "Consigna (M1, M2)       =(%d,%d)\n", TaulaConsignes[fm][cm][0],
                                          TaulaConsignes[fm][cm][1] );

  fprintf ( fd, "\n" );
}

/* Escriure la trajectoria en un fitxer */
void escriureTrajectoria ( FILE *fd, const int NumIteracions,
                           const TPasIteracio TaulaIteracions[K_MAX],
                           const TPasKalman TaulaKalman[K_MAX] )
{
  int i;

  for ( i = 0; i < NumIteracions; i++ )
    fprintf ( fd,
              "x=%3.5f\ty=%3.5f\to=%3.3f\tUE=%4d\tUD=%4d\tAre=%2.2f\tArd=%2.2f\n
               \t\t\t\t\t\tARE =%3.5f\tARD=%3.5f\n",
                 TaulaIteracions[i].Pas.x, TaulaIteracions[i].Pas.y,
                 TaulaIteracions[i].Pas.o,
                 TaulaIteracions[i].Pas.UE, TaulaIteracions[i].Pas.UD,
                 TaulaIteracions[i].Pas.Are, TaulaIteracions[i].Pas.Ard,
                 TaulaKalman[i].ARE, TaulaKalman[i].ARD );
}


void matlab ( FILE *fd, const int NumIteracions,
              const TPasIteracio TaulaIteracions[K_MAX],
              const TPasKalman TaulaKalman[K_MAX] )
{
  int i;

  for ( i = 0; i < NumIteracions; i++ )
   fprintf ( fd,
             "%3.5f\t%3.5f\t%3.3f\t%4d\t%4d\t%2.2f\t%2.2f\t%3.5f\t%3.5f\t%12.6g\n",
             TaulaIteracions[i].Pas.x, TaulaIteracions[i].Pas.y,
             TaulaIteracions[i].Pas.o, TaulaIteracions[i].Pas.UE,
             TaulaIteracions[i].Pas.UD, TaulaIteracions[i].Pas.Are,
             TaulaIteracions[i].Pas.Ard, TaulaKalman[i].ARE, TaulaKalman[i].ARD,
             TaulaIteracions[i].DistTrj);
}
```

### A.6. Codi ltp.h

Aquest codi permetrà que ens puguem comunicar amb el robot a través del port paral·lel.

```
#ifndef __LPT_H__
#define __LPT_H__

#include <asm/io.h>

/* Adreces base dels ports */
#define LPT1 0x378
#define LPT2 0x278
```

```
/* Adreces relatives als registres */
#define REG_DADES   0
#define REG_ESTAT   1
#define REG_CONTROL 2

/* FUNCIONS I ADRECES */

void Obrir_LPT(int fd);
void Tancar_LPT(int fd);
int Llegir_LPTEstat(int fd);
int Llegir_LPTControl(int fd);
   /* fd = 1 -> obra LPT1
      fd = 2 -> obra LPT2 */

void Escriure_LPTDades(int fd, int valor);
   /* Escriu el valor pel registre de dades del port paral·lel */

void Escriure_LPTControl(int fd, int valor);
   /* Escriu el valor pel registre de control del port paral·lel */

#endif
```

## A.7. Codi robcom

Per últim, mostrem el tall de codi utilitzat per enviar les consignes al robot. Aquest codi estarà dividit en dos arxius. Un s'anomena robcom.h, on tenim declarada la funció, i l'altre robcom.cpp on es mostra en el codi la seva implementació.

### A.7.1. robcom.h

```
ifndef __ROBCOM_H__
#define __ROBCOM_H__

// ---- MOTORS ----------------------
#define DRET            1
#define ESQUERRE        0
#define PARAT         127
#define RETARD       4000

void EnviaConsignaMotor ( int Motor, int Vel );

#endif
```

### A.7.2. robcom.cpp

```
#include "robcom.h"

#if defined (WINDOWS) || defined (WIN32)
  #include <stdio.h>
#else
  #include <robot.h>
#endif

void EnviarConsignaMotor ( int CodiMotor, int Vel )
{
```

```
#if defined (WINDOWS) || defined (WIN32)
  fprintf ( stderr, "Enviada consigna %d al motor %d\n", Vel, CodiMotor );
#else
  Mou_Motor ( CodiMotor, Vel );
#endif
}

int LlegirVelocitatMotor ( int CodiMotor )
{
  int Vel;

#if defined (WINDOWS) || defined (WIN32)
  Vel = 10;
  fprintf ( stderr, "Llegida velocitat %d del motor %d\n", Vel, CodiMotor );
#else
  Vel = Llegeix_Velocitat_Motor ( CodiMotor );
#endif

  return 255-Vel;
}
```