EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

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

Títol: Disseny de controladors òptims per al robot Pioneer

Document: 1. Memòria

Alumne: Gerard Alarcón Rodríguez

Director/Tutor: Bianca Mariela Innocenti Badano
Departament: Enginyeria Elèctrica, Electrònica i Automàtica
Àrea: ESA

Convocatòria (mes/any): gener/2011

Índex

1. Introducció

1.1. Antecedents

El grup de recerca eXit de la Universitat de Girona està desenvolupant una arquitectura de control distribuïda per al robot Pioneer 2DX. L'estudi d'aquest model de robot va començar mitjançant el projecte de Gemma Pou titulat: "Implementació dels comportaments bàsics de navegació pel robot GRILL utilitzant el Sistema Operatiu de Temps Real QNX".

Més tard, Sílvia Garcia, en el seu PFC titulat: "Implementació d'una arquitectura multi-agent reactiva per al control del Grill" va implementar aquesta arquitectura com a sistema multi-agent utilitzant una plataforma multi-agent comercial anomenada Open Agent Architecture (OAA). Ara, aquesta arquitectura de control del robot ha evolucionat de manera que va ser possible solucionar alguns problemes que tenia anteriorment. Un exemple d'això es la implementació del control en un Sistema Operatiu més versàtil, cosa que va permetre poder canviar de QNX a Linux, o eliminar la centralització produïda per un agent especial d'OAA. L'arquitectura actual funciona sota Linux i està programada en C++ .

Posteriorment en el projecte de l'Encarnació González es va aconseguir dissenyar un únic agent utilitzant la lògica difusa, de tal manera que amb aquest únic control es pogués anar des d'un punt d'una habitació a un altre evitant tots els obstacles que es pogués trobar. D'aquesta manera es va aconseguir reduir la interacció entre dos agents pel control de la trajectòria del robot a un únic agent que la controlés.

1.2. Objecte

Amb aquest projecte es pretén dissenyar dos agents "go to" (posicionament) que acompleixin uns requeriments estipulats.El present projecte pretén dissenyar i integrar en l'arquitectura de control actual del robot Pioneer diferents agents "go to"(posicionament) que acompleixin certs requeriments estipulats prèviament.

1.3. Especificacions i abast

Després de definir els objectius d' aquest projecte, es podran aconseguir seguint l'aplicació d'una metodologia que recollirà les següents fases:
Construcció e implementació dels agents de control mitjançant el llenguatge C++ amb el programa KDevelop® del sistema operatiu Linux.
Simulació de la resposta del robot enfront aquests mètodes mitjançant el programa MobileSim®.

Anàlisi dels resultats i comparació amb d'aquests amb els resultats obtinguts amb l'arquitectura original (ús de Matlab).
Comparació dels resultats trobats mitjançant el simulador amb la resposta en el robot real.

1.4. Especificacions

Totes les especificacions de qualsevol apartat d'aquest projecte estan referides i referenciades en els seus annexos corresponents.
Aquest projecte està centrat únicament en una investigació, i qualsevol estudi posterior, necessitarà tenir com a referència tot el procediment que s'esmentarà a continuació i que serà necessari per portar a terme un assaig d'aquestes condicions.

2. El robot Pioneer 2DX

2.1. Introducció a el robot Pioneer 2DX

El MobileRobots Inc Pioner 2DX és un robot de tres rodes mòbils de 44 cm de llarg, 33 cm d'ample i 22 cm d'alçada. El seu pes és de 16,55 kg (sense bateries) i ell seu cos és d'alumini.

A la part posterior disposa de 3 bateries de 12 V que li permeten funcionar contínuament durant 8-10 hores. A la part frontal, compta amb dues rodes motrius amb el mateix eix de rotació que funcionen amb motors de corrent continu.

Una roda lliure es connecta a la plataforma per una estructura rígida i pot girar al voltant del seu eix vertical.

Les velocitats màximes a les que el robot pot arribar a 1.6 m/s lineals i 300 graus/s angular. El robot té un cinturó de vuit sensors d'ultrasò a la part davantera per detectar obstacles, dos encoders per determinar la posició de les rodes, i un sensor de nivell de la bateria per comprovar la càrrega de les bateries. Hi ha dos sensors d'ultrasons a cada costat del robot (a 90 graus) i els sis es col·loquen en intervals de 20 graus, com es mostra a la figura 2. La freqüència d'adquisició del sonar és de 25 Hz i la sensibilitat varia entre 10 cm i 5 m.

Figura 1. Robot PIoneer 2DX.

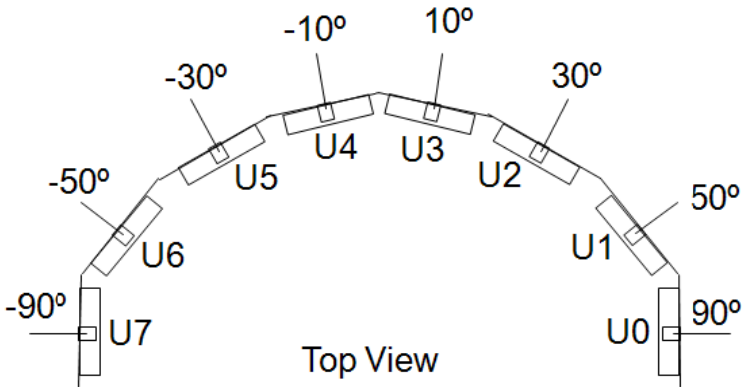

Figura 2. Abast dels sensors sonar

2.2. Paràmetres tècnics del robot

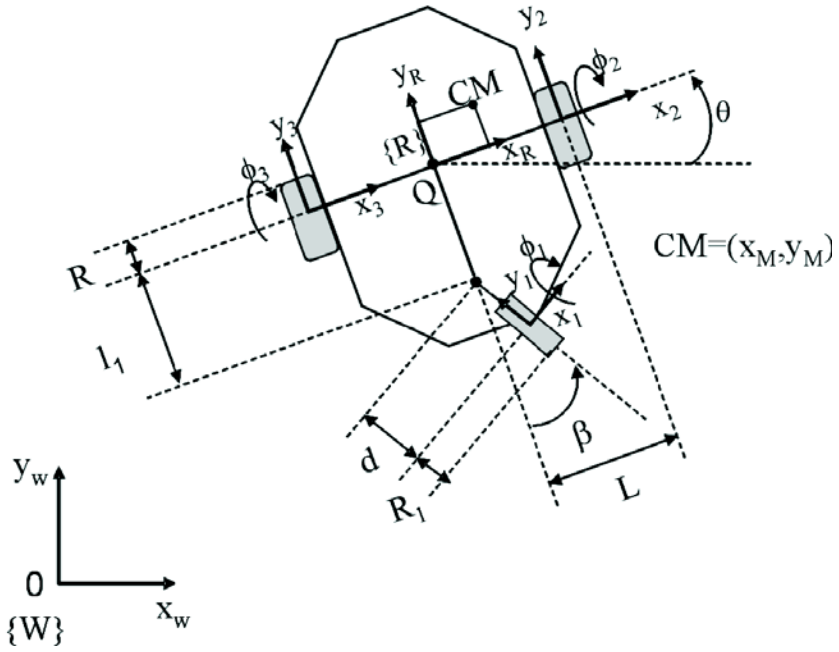

Figura 3. Robot

La figura 3 mostra tots els paràmetres del model de robot mentre que a la taula 1 es mostra la seva descripció (què representen) i les seves unitats.

Tant la figura 3 com la taula 1 han estat exretes de la tesi doctoral citada en la bibliografia d'aquest mateix document.

Symbol	Description	Units
x, y	origin of robot fixed frame	[m]
θ	robot heading	[rad]
β	orientation of the free wheel	[rad]
ϕ_1	angular position wheel 1	[rad]
ϕ_2	angular position wheel 2	[rad]
ϕ_3	angular position wheel 3	[rad]
η_1	linear velocity of robot	[m/s]
η_2	angular velocity of robot	[rad/s]
M	mass of the robot	[kg]
m_2	mass of driving wheel 2	[kg]
m_3	mass of driving wheel 3	[kg]
m_1	mass of free wheel	[kg]
R	radius of wheels 2 and 3	[m]
R_1	radius of wheel 1	[m]
L	distance from robot frame to driving wheels	[m]
l_1	distance from robot frame to the moving bar of the free wheel	[m]
d	distance of the moving bar of the free wheel	[m]
V_a	armature voltage	[V]
i_a	armature current	[A]
x_m, y_m	coordinates of the center of mass (CM)	[m]
I_0	inertia robot	$[kg.m^2]$
I_{r1}	inertia in relation to the rotational axis of the free wheel	$[kg.m^2]$
I_{r2}	inertia in relation to the rotational axis of the driving wheel 2	$[kg.m^2]$
I_{r3}	inertia in relation to the rotational axis of the driving wheel 3	$[kg.m^2]$
I_{p1}	inertia in relation to the vertical axis of the free wheel	$[kg.m^2]$
I_p	inertia in relation to the vertical axis of the wheels 2 and 3	$[kg.m^2]$
C_{s1}	coulomb friction torque free wheel	$[N.m]$
C_{v1}	viscous friction coefficient free wheel	$[kg.m^2/s]$
C_s	coulomb friction torque wheels 2 and 3	$[N.m]$
C_v	viscous friction coefficient wheels 2 and 3	$[kg.m^2/s]$

Taula 1. Paràmetres del model

2.3. Simulador

El programari que utilitzarem per fer la simulació de la trajectòria del robot serà el MobileSim. Aquest programa consta d'una pantalla senzilla que pot funcionar tant en Linux com en Windows®

El programa es una eina senzilla de fer servir i útil per la tasca que ens porta. En una primera pantalla ens demana que entrem el mapa de l'entorn per on volem que es mogui el nostre robot quedant una pantalla com la Figura 4.

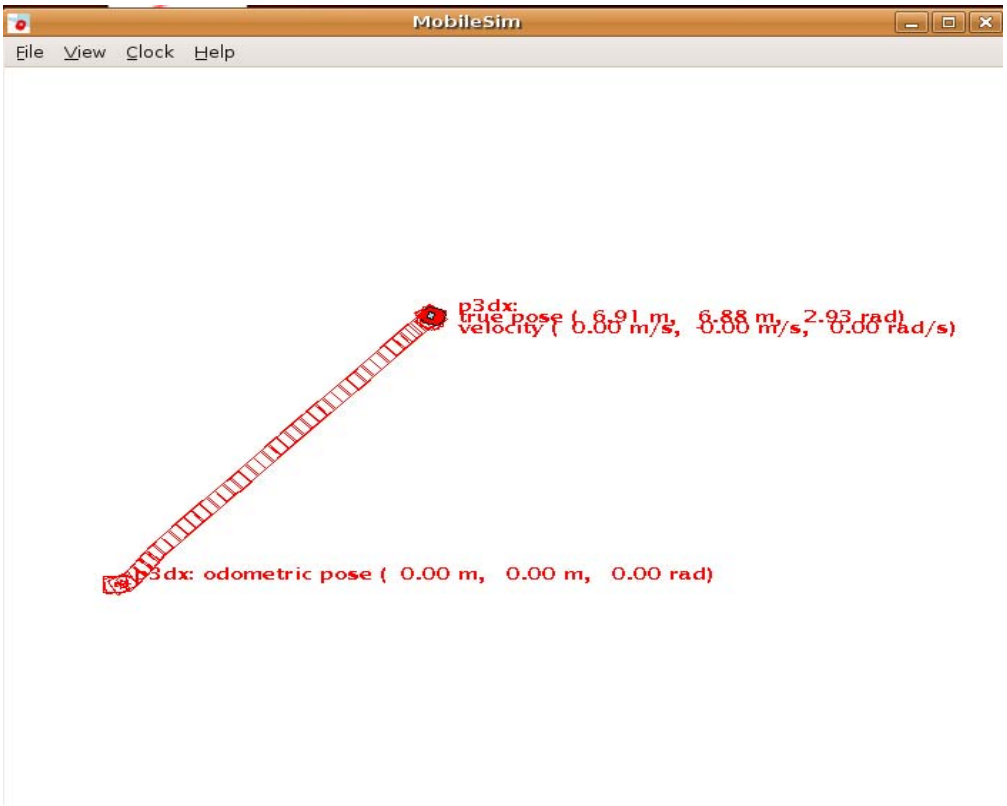

Figura 4. Pantalla del MobileSim

3. Arquitectura multiagent de control del robot

3.1. Introducció als agents que formen part de l'arquitectura

En aquests moments l'arquitectura multiagent és una estructura híbrida, és a dir que una part respon als canvis de l'entorn, i una altra que dissenya els moviments que farà el robot en funció de la resposta obtinguda. Aquesta arquitectura híbrida s'anomena ARMADiCo (Autonomous Robot Mutli-Agent Architecture whit Distribiuted Coordination) i la formen cinc grups d'agents que controlen el robot, com es mostra a la Figura 5.

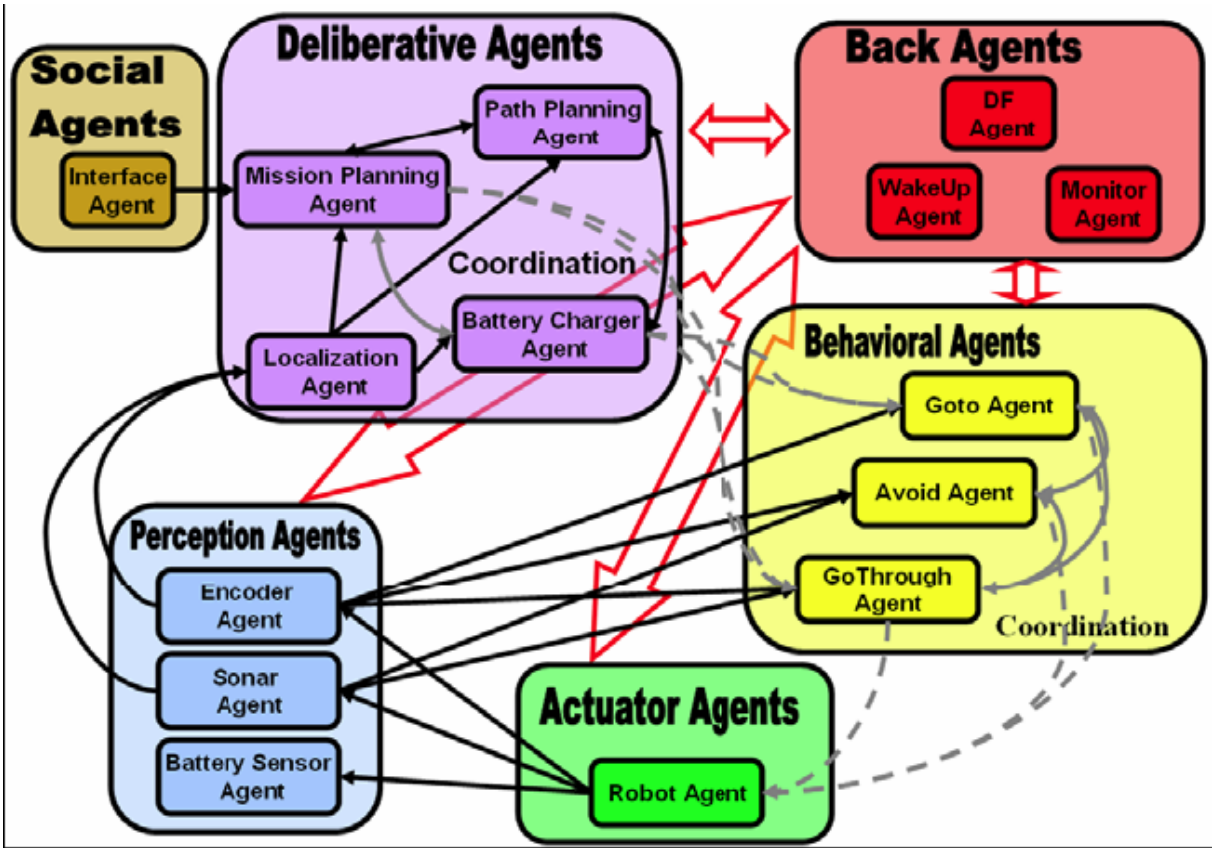

Figura 5. Agents que formen ARMADiCo.

Aquest cinc gran grups són tal i com es pot observar a la Figura 5 són: els agents socials, els agents deliberatius, els agents reactius, els agents de percepció i els agents actuadors.

Primerament tenim els Agents Socials, que són els que coordinen les comunicacions entre els agents de l'arquitectura de control i els agents d'altres plataformes i/o humans. Dins dels agents socials s'ha implementat l'agent Interface que permet la comunicació amb l'usuari i a través d'aquest es poden modificar els paràmetres com per exemple l'adreça IP, el port de l'ordinador per on enviar dades, o les coordenades finals i inicials del recorregut que

desitgem. No només serveix per modificar les dades sinó que també es l'encarregat de ensenyar-nos missatges i valors que proporcionen els agents de l'arquitectura de control del robot.

Els agents Deliberatius, tenen la missió d'aconseguir que el robot pugui realitzar tasques complexes com poden ser; descomposar una tasca complexa en tasques senzilles (Mission Planning Agent), planificar una trajectòria per aconseguir el recorregut òptim (Path Planning Agent), determinar en quina part del mapa global es troba el robot (Localization Agent) i també s'encarrega de decidir quan el robot ha d'anar a la plataforma de recàrrega perquè les seves bateries es troben en un punt crític de càrrega (Battery Charger Agent).

Els agents Reactius, són els encarregats de reaccionar als esdeveniments que es produeixen a l'entorn del robot. Aquests són els agents que hem tractat en el desenvolupament del projecte. L'Avoid, té la funció d'esquivar un obstacle captat pels sensors. El GoTo és el que s'encarrega d'anar d'un punt a un altre del mapa i el GoThrough gestiona el moviment del robot quan ha de passar per llocs estrets com passadissos i portes.

Els agents de Percepció , s´'encarreguen de obtenir les lectures dels sensors tant interns (relacionats amb el propi robot), com externs (percepció de l'entorn).. Aquests agents són l'agent Encoder, que té la funció de calcular la distància que ja ha recorregut el robot i l'angle que ha girat. L'agent Sonar, que té la missió de crear un mapa local amb els punts d'obstacle que detecta en el seu entorn. Finalment l'agent Battery sensor, que és l'encarregat de llegir el sensor de bateria i donar una alarma quan el nivell està per sota del nivell crític.

Els agents Actuadors, que són els que desenvolupen la funció de fer que el robot es mogui mecànicament. En aquest grup trobem el Robot Agent, que per al Pioneer, és realment una interfície entre el robot i la arquitectura de control multi-agent (el robot no permet més d'una connexió amb el micro-processador a la vegada i per tant per obtenir totes les dades necessaries necessitem d'aquesta interfície).

Finalment, existeix el grup d'els Back Agents que s'encarreguen del correcte funcionament de la plataforma multi-agent i ajuden a l'arquitectura de control a mantenir tot els agents necessaris per desenvolupar una missió funcionant i communicant-se entre si. Fonamentalment en aquest grup cal destacar el DF Agent (Directory Facilitater Agent) que s'encarrega de mantenir una agenda de tots els agents involucrats en el control per una

determinada missió, dels serveis que ofereix cada agent i dels recursos que necessiten. Per exemple, l'encoder agent necessita del robot agent (recurs) per obtenir les lectures de l'encoder i proporcionen les coordenades globals de posicionament del robot (servei).

Els agents que es faran servir per la realització d'aquest projecte són: DF Agent, Avoid Agent, GoThrough Agent, Robot Agent, Encoder Agent, Sonar Agent, Goto Agent i Interface Agent.

3.2. Agent goto

Com que l'objectiu d'aquest projecte és dissenyar altres agents GoTo, primerament farem una descripció de l'agent que actualment es troba a la plataforma per fer el control de posició del robot.

Aquest és l'agent responsable de dirigir el robot a la posició meta. Tenint en compte la posició desitjada (x,y) i una orientació θ, i en base a la posició actual i el destí, aquest agent calcula la velocitat lineal i angular per impulsar el robot a la posició objectiu (meta).
És també qui rep les coordenades globals del robot de l'agent codificador i els punts desitjats de la trajectòria de l'agent de planificació de la missió o de l'agent carregador de bateries.
El GoTo agent envia les velocitats lineals i angulars que es desitgen pel Robot Agent.

L'implementació actual de l'agent GoTo és un controlador col·laboratiu que fa servir dos PIDs, un precís és a dir més lent, i l'altre més ràpid, per arribar als punts destí de la manera més precisa i ràpida possible, combinant els senyals de control dels dos PIDs a través de la lògica difusa (Per a més informació [TesisBianca]).

3.3. Funcionalitat dels nous agents GoTo

En aquest projecte es buscarà, la implementació del mètode de la persecució pura (pure pursuit) a la plataforma com a alternativa a agent GoTo actual i també es dissenyarà un controlador PID adaptat per treballar a l'exterior (tot el que s'ha fet fins ara és per treballar a l'interior dels edificis) és a dir, que permiti al robot moure's per superfícies rugoses (amb més fricció que les rajoles de l'interior).

El disseny del primer agent GoTo segueix el mètode de la persecució pura i mitjançant paràmetres concrets i condicions basades en aquest mètode, el robot tindrà la capacitat de seguir la trajectòria originada després de conèixer el punt inicial i el punt a el que volem que arribi aquest.

El disseny del segon controlador és bàsicament empíric (ja que el simulador no permet canviar paràmetres com la fricció) el qual, mitjançant una sèrie de probes ens permetrà ajustar amb major precisió una sèrie de variables per tal d'aconseguir sempre les condicions de funcionament més adequades.

4. Mètode de la persecució pura (Pure pursuit)

4.1. Introducció al mètode de seguiment de camins

El mètode de la persecució pura és un mètode que està basat en el mètode del seguiment de camins. A la Figura 6. es mostra un robot i un camí que es pretén que el robot segueixi de forma independent.

En molts casos es suposa que la velocitat es manté constant. En aquest cas convé dir que la velocitat és un paràmetre que intervé en el model del bucle de control de la direcció. Per tant, si s'utilitza una estratègia de control basada en el model, quan es modifica la velocitat també canvia el model i per tant s'ha de modificar de manera apropiada la llei de control del robot.

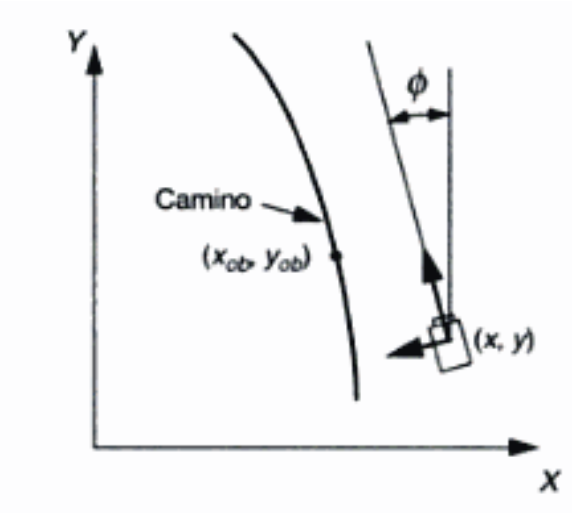

Figura 6. Seguiment de camins explícits

El camí que pretenem que el robot segueixi es pot especificar de diferents maneres entre les que es poden destacar:

Especificació prèvia en coordenades absolutes a mitjançant plànols o mapes, o bé amb el propi robot dotat d'un sistema GPS diferencial recarreguen un camí que quedaria gravat en memòria per a la seva reproducció a posteriori.

Especificació interactiva des d'un terminal utilitzant tècniques de telerobòtica o instruccions específicques.

En una arquitectura de control intel·ligent utilitzant mètodes de planificació de camins. Aquesta és la tècnica que s'utilitza en nombrosos sistemes de navegació autònoma.

Mitjançant el sistema de percepció, tal i com succeeix en el seguiment de línies obtingudes mitjançant un sistema de visió sobre el vehicle, o bé en el seguiment de carreteres utilitzant també el sistema de visió.

En qualsevol cas, es suposarà que la trajectòria desitjada ve donada per una funció explícita en coordenades globals, o bé mitjançant una seqüència de posicions (x,y,θ,φ) essent (x,y) les coordenades, θ l'orientació i φ la curvatura.

Les especificacions per al disseny d'aquest llaç de control són normalment les següents:
Precisió en el seguiment:error de seguiment el menor possible, comportament dinàmic: estabilitat relativa i rapidesa dels transitoris i menor esforç de control: minimització d'actuacions.

Per a controlar el vehicle sobre la trajectòria es necessita aproximar la posició. Tal i com sabem, el mètode més senzill és mitjançant odometria utilitzant les mesures dels codificadors òptics (encoders) a l'eix de les rodes i un determinat model. Així mateix, és possible utilitzar sensors de navegació tals com els giroscopis, per a aproximar els angles del vehicle, i les bruixoles, per a aproximar la direcció del vehicle respecte al nord.

S'han utilitzat altres sensors també com acceleròmetres que permeten obtenir la posició del vehicle mitjançant la doble integració de l'acceleració mesurada. No obstant això, és necessari posar de manifest que,a la major part dels robots mòbils, la relació senyal/soroll sol ser petita (acceleracions baixes) el que fa que l'error d'aproximació sigui gran.

Els sistemes de navegació inercial integren diverses mesures d'angles i apliquen tècniques bàsiques de fusió sensorial per a millorar les estimacions, afegint també altres sensors com per exemple acceleròmetres. No obstant, el seu cost sol ser elevat.

Per últim en robòtica d'exteriors, és necessari anomenar també els sistemes de posicionament global per satèl·lit (GPS). A l'actualitat existeixen sistemes diferencials de posicionament global per satèl·lit (DGPS) que permeten arribar a aproximar la posició amb errors de pocs centímetres.

En general és possible millorar l'estimació de la posició del vehicle mitjançant la integració de diferents sensors, per a tal cosa s'utilitzen mètodes com per exemple el filtre de Kalman.

4.2. Seguiment de camins mitjançant la persecució pura (Pure pursuit)

Considerem un sistema de referència local associat a el moviment del vehicle, tal i com podem veure a la Figura 7. Suposem també que, en el interval de control, la curvatura és constant, descrivint així el vehicle un arc de circumferència.

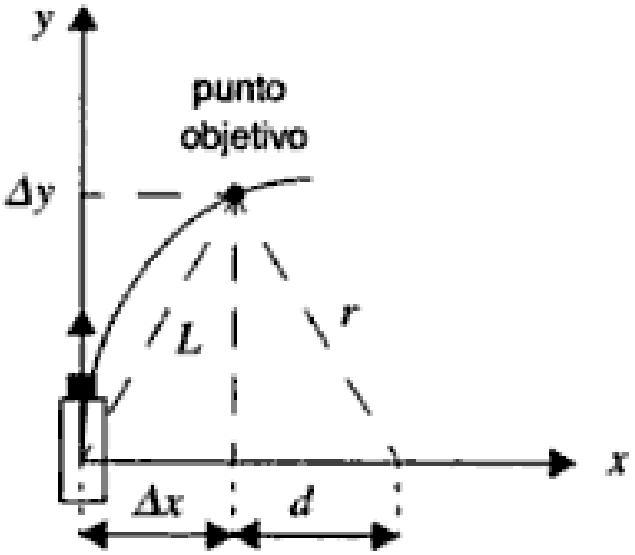

Figura 7. Seguiment de camins mitjançant la persecució pura

De l'anàlisi de la figura es dedueixen les relacions:

$$r = \Delta x + d \qquad \text{(Eq.1)}$$

$$d^2 + (\Delta y)^2 = r^2 \qquad \text{(Eq.2)}$$

Essent r el radi de curvatura del vehicle.

Aïllant la d a la primera (Eq.1) i substituint a la segona (Eq.2) s'obté:

$$(r - \Delta x)^2 + (\Delta y)^2 = r^2 \qquad \text{(Eq.3)}$$

On el radi de curvatura necessari per a que el vehicle es desplaci $\Delta x, \Delta y$ és:

$$r = \frac{(\Delta x)^2 + (\Delta y)^2}{2\Delta x} \qquad \text{(Eq.4)}$$

Per tant la curvatura que és necessari subministrar-li al vehicle és:

$$\gamma_r = \frac{1}{r} = -\frac{2(\Delta x)}{L^2} \qquad \text{(Eq.5)}$$

On el signe ve donat pel sentit de gir necessari per a trobar el punt objectiu a la Figura 7. La L és la distància en la que es troba el punt objectiu i Δx és el desplaçament lateral.

L'expressió anterior constitueix la llei de control de la persecució pura (pure pursuit). S'ha de veure com una llei de control proporcional a l'error lateral (Δx) respecte el punt objectiu. La constant de proporcionalitat (guany) varia amb la inversa del quadrat de L.

A la figura es pot veure també que la curvatura de la persecució pura és l'inversa del radi d'una de les circumferències que passa per la posició actual del vehicle i pel punt objectiu.

El programa per a l'aplicació d'aquesta llei de control és molt senzill. Tant sols és necessari determinar el punt del camí que es troba a una distància prèviament definida L, i calcular l'error lateral (Δx) respecte a la posició actual del centre de guiat del vehicle.

Si les coordenades estan en un sistema global, és necessari tenir en compte l'orientació del vehicle per obtenir (Δx). En efecte, si el vehicle està en les coordenades globals (x,y*)* amb orientació θ (Figura 6.), i el punt objectiu sobre el camí està a les coordenades globals (x_{ob}, y_{ob}) llavors tindrem:

$$\Delta x = (x_{ob} - x)\cos\theta + (y_{ob} - y)\sin\theta \qquad \text{(Eq.6)}$$

Un mètode pràctic per aplicar la llei de control consisteix en obtenir, en cada període de control, el punt (x_{obm}, y_{obm}) del camí objectiu que està més proper al vehicle (x,y) i escollir el punt objectiu (x_{ob}, y_{ob}) a una distància fixa *s* sobre el camí tornada en el sentit en que es desplaça el robot a partir de (x_{obm}, y_{obm}), tal i com s'il·lustra a la Figura 8. A continuació calculem:

$$L = \sqrt{(x_{ob} - x)^2 + (y_{ob} - y)^2} \qquad \text{(Eq.7)}$$

I amb (Eq.6), s'aplica la llei de control (Eq.5).

En qualsevol cas, l'elecció del paràmetre de distància al punt objectiu és crítica per a l'eficiència del controlador de persecució pura.

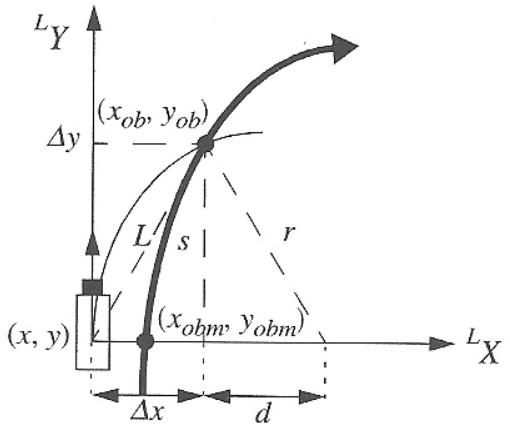

Figura 8. Aplicació del mètode de la persecució pura amb distància fixa sobre el camí.

Existeixen circumstàncies en les que la simple aplicació de la llei de control anterior presenta problemes. Així, quan el punt objectiu es troba molt allunyat (L és molt gran), l'actuació subministrada per la llei de control pot resultar massa petita. En aquest cas s'ha de substituir la distància L al punt objectiu per una distància màxima.

Un altre cas a tenir en compte és quan el vehicle es troba sobre el camí (Δx molt petita), però orientat en direcció contrària al camí, situació en la qual s'obtindria un valor molt petit de la curvatura que no permetria girar al vehicle fins a orientar-se segons el camí. Si es detecta aquesta condició, és necessari substituir la curvatura obtinguda per una de valor més gran.

4.3. Disseny de la trajectòria a través de l'equació de la recta

En el nostre cas hem adaptat el mètode a les condicions que ens han semblat més apropiades i a la vegada útils pel que realment necessitem. Per això, el que hem fet és aproximar totes les trajectòries per trams rectes. Per tal de poder definir i dissenyar aquesta part del programa s'han realitzat una sèrie de proves prèviament amb el programari de Matlab ®.

Per començar a definir el mètode ens hem basat evidentment en la simple equació que defineix una recta (Eq.8):

$$y = (m \cdot x) + n \qquad \text{(Eq.8)}$$

On:

y : component y del punt que passa per la recta.

m : pendent de la recta.

x : component x del punt que passa per la recta.

n : ordenada a l'origen de l'equació de la recta.

A continuació hem definit els paràmetres que caracteritzen aquesta recta com són el pendent i l'ordenada a l'origen mitjançant les següents equacions (Eq.9) i (Eq.10):

$$m = \frac{y_0 - y_{final}}{x_0 - x_{final}} \qquad \text{(Eq.9)}$$

$$n = y_0 \quad \frac{y_0 - y_{final}}{x_0 - x_{final}} \quad x_0 \qquad \text{(Eq.10)}$$

Gràcies a aquestes equacions hem pogut definir cada punt dels que formen la trajectòria de la recta.

Per aconseguir-ho hem creat una taula de punts que ens definiran la trajectòria. Simplement hem creat una variable índex que creï nous punts de la recta i emmagatzemi el valor anterior d'aquest.

No cal dir que com més punts emmagatzemem major serà la precisió que obtindrem de la trajectòria rectilínia. En el nostre cas per les distàncies que cobrim amb el nostre robot hem considerat que 50 punts serien suficients.

Un cop hem comprovat el correcte funcionament d'aquest procés en el Matlab® l'hem aplicat en el programa amb llenguatge C++.

Hem definit una funció coneguda com a "CalculaTrajectòria" que serà la que realitzarà tot aquest procés per a el programa.

```
void GotoAgent::CalculaTrajectoria(int max_val)
{

	double m,n, difx;
	double pi=3.141596;
	int i;

if((pos_ini.getX()-setpoint.getX())==0)
	{
```

```
	difx=(setpoint.getY()-pos_ini.getY())/max_val;
	for(i=0;i<max_val;i++)
		{
		x_camino[i]=pos_ini.getX();
		y_camino[i]=pos_ini.getY()+(i*difx);
		}
	}
else
{
m=(pos_ini.getY()-setpoint.getY())/(pos_ini.getX()-setpoint.getX());
n=pos_ini.getY()-((pos_ini.getY()-setpoint.getY())/(pos_ini.getX()-
setpoint.getX()))*pos_ini.getX();
difx=(setpoint.getX()-pos_ini.getX())/max_val;

	for(i=0;i<max_val;i++)
	{
	x_camino[i]=pos_ini.getX()+(i*difx);
	y_camino[i]=((m*x_camino[i])+n);
	}
}
}
```

Amb aquesta funció no en farem prou però, ja que tant sols ens serveix per definir els punts teòrics del camí que hauria de seguir el robot però el que necessitem és la manera de fer que aquests punts facin reorientar i redirigir el robot cap a la trajectòria correcte que això fet per a cada punt el que acaba aconseguint no és res més que el robot movent-se per la trajectòria o camí adequat. És per això que el mètode es coneix com a mètode de persecució pura (Pure pursuit).

4.4. Aplicació del mètode del pure pursuit al nostre cas

Un cop hem aconseguit dissenyar la recta que generen els dos punts entre els quals s'ha de moure el robot necessitem que aquesta trajectòria faci reorientar el robot cap al camí correcte. Per això ha estat necessari la implementació en llenguatge C++ del ja mencionat en diversos apartats anteriors mètode de la persecució pura o pure pursuit.

Per fer-ho hem creat una altra funció dins l'agent GoTo, la qual hem anomenat geometric. Aquesta funció, no és res més que l'encarregada de realitzar els càlculs necessaris en tots els punts per saber, en cada instant en el que es troba el robot, quina ha de ser la velocitat angular que se li ha d'enviar, per tal que mantingui una orientació adequada per arribar al destí desitjat.

La primera part de la funció serveix per declarar algunes de les variables que no són globals que farem servir dins la funció.

```
MSpeeds GotoAgent::geometric(double Lmax, double s,double vel_lin) // totes en mm
{
MSpeeds veld;
int max_val=50;
int i;
float x_vehiculo,y_vehiculo, x_ob, y_ob; //tenir en compte que estan en mm
float phi_vehiculo; // en graus!
double dist[100];
double acum[100];
int j, k; //j per posar l'index del valor més petit
double val; // per posar el valor mes petit
double incremento_x,incremento_y, L,curvatura;
double pi=3.141596;
double vel_ang;
double signe;
double Kpp, der_Dx;
double dx,dy;
QString string,cad;
double m,n;
```

Com es pot veure la majoria de variables són de tipus double ja que la majoria dels valors que utilitzarem necessitaran decimals i és la variable adient.

A continuació hi ha la part del programa que executa per primera vegada la funció.

```
if (prinv==TRUE)    //primera vegada que fa la funcio
       {
              pos_ini.setX(pos.getX());
              pos_ini.setY(pos.getY());
              pos_ini.setTh(pos.getTh());
              CalculaTrajectoria(max_val);
```

En aquest pas la única cosa que fem és obtenir de la informació que disposem la posició inicial del robot i l'orientació d'aquest mateix en aquell instant.
A la vegada cridem per primera vegada la funció que calcularà la trajectòria de la recta.

Un cop hem fet això entra en acció la següent part de la funció que és la que es pot veure a continuació.

```
prinv=FALSE;
K_ANT=1;

for(i=0;i<max_val;i++)
{
	acum[i]=0;

	for(k=i;k<max_val-1;k++)
	{
		val=sqrt(pow((x_camino[k]-x_camino[k+1])/1000.0,2) + pow((y_camino[k]-
		y_camino[k+1])/1000.0,2));
		acum[k+1]=acum[k]+val;
	}
	for(k=i;k<max_val;k++)
	{
		dist[k]=fabs(acum[k]-s);
	}
	val=dist[i];
	j=i;
	//nomes interessa j

	for(k=i; k<max_val;k++)
	{
		if(dist[k]<val) // si el valor en dist es mes petit que val
		{
			val=dist[k]; //poso a val el valor mes petit
			j=k;	// poso en j l'index del valor mes petit
		}
	}
	// a la sortida d'aquest for tinc el valor mínim en val i l'index a j
	INDEX[i]=j;
}
```

En aquesta part del programa el que s'ha fet és fer un mapa de punts depenent de la posició en la que es troba el robot en cada instant. Per cada posició, es calcula la distància que hi ha entre aquest punt i cadascun dels punts que s'han creat en el mapa de la trajectòria.
Posteriorment el que fem és buscar quina és la distància més curta, tot això per saber quin és el punt de la trajectòria que es troba més a prop del punt en el que està en aquest instant el robot.
Un cop sabem quina és la distància més curta el que fem és emmagatzemar el valor de l'índex o de la posició en la que es troba aquesta distància mínima.

A continuació el programa segueix. Per introduir la part de la funció que segueix la seva utilitat no és altra que seguir calculant les distàncies a les que es troba el robot dels punts de la trajectòria. Podem seguir veient l'estructura del programa en els annexos del document.

En aquesta part de la funció el programa segueix calculant les distàncies entre el robot i els punts de la trajectòria sobre el camí i acumula el valor de l'índex amb distància mínima.

Arriba un moment però, que hi ha un punt que està més a prop que l'anterior i per tant el valor en el que ens hem de fixar ja no és aquest sinó un altre. Això provoca un nou canvi de càlcul en les distàncies i a la vegada un canvi de valor de l'índex amb distància mínima.

Finalment tenim la part del programa en que tots aquest aspectes es veuen reflectits en valors físics que no provoquen altre cosa que portar el robot a moure's.
Tot això ho aconseguim gràcies a aquesta part del programa, que calcula, a través les fórmules vistes anteriorment de manera teòrica, les variables que provoquen el moviment del robot.

En el cas de la persecució pura la velocitat lineal és un paràmetre fix, però en el nostre cas, l'obtemin a través d'un PID que dóna un valor d'aquesta variable, en funció de la distància que queda a l'objectiu.

Amb això finalitzem el que és la part del programa que genera les variables que fan moure el robot a la velocitat adequada i pel camí adequat.
Aquest increment de x no és res més que la distància lateral que hi ha des del punt en el que es troba el robot fins al punt objectiu d'aquest mateix. Comentari: si canvies les formules aixo no fa falta, s'entendra mes be. Si cal canvies al programa incremento_x per incremento_y i ja esta.

El següent pas no és altre que calcular la variable L que és la distància que hi ha des del punt en el que es troba el vehicle fins al punt objectiu. Aquest procés inclou una limitació de una L màxima per tal de no desorbitar els càlculs en alguns punts i que això provoqui alteracions en la trajectòria del robot.

A continuació calculem el que coneixem com a curvatura. Aquesta variable no és altra que la relacionada amb la velocitat lineal, i ens permetrà conèixer el valor de la velocitat angular.

Un cop calculem la velocitat angular podem dir que obtenim el valor que és l'artífex de reconduir el robot cap a la trajectòria que desitgem. Per aconseguir resultats més bons, i com que tant la velocitat angular com la lineal estan limitades a valors màxims per les condicions del robot i del terra on es mou, s'ha fet una funció que limita les velocitats quan són més grans que el valor màxim permés, de manera que es modifica el mòdul però es manté la relació entre les velocitats per aconseguir l'orientació desitjada.

Finalment assignem els valors obtinguts de velocitat angular i lineal als paràmetres de l'agent GoTo que aquest agent envia al Robot Agent i per tant, al robot.

Cal dir que aquestes funcions no són cap agent de l'arquitectura del robot, sinó, una sèrie de funcions que hem integrat dins de l'agent GoTo, que és l'agent encarregat de gestionar el moviment del robot a la posició desitjada. Aquestes funcions que hem dissenyat, el que sí fan és obtenir les variables que proporcionen la informació a l'agent per tal que aquest les enviï al robot i d'aquesta manera obtinguem de manera física el moviment desitjat del robot.

5. Disseny empíric d'un controlador PID pel funcionament en superfícies rugoses

El segon objectiu d'aquest projecte és el d'implementar un segon controlador per l'agent GoTo, però aquesta vegada utilitzant una metodologia de disseny totalment oposada.
En aquest cas, resultava complicat realitzar les simulacions per tal d'aconseguir trobar el controlador adequat, ja que resulta actualment inviable amb les eines de treball que es disposen, poder simular un entorn físic pel robot, amb unes condicions específiques en el terra pel que es desplaçarà. Per tant hem utilitzat una metodologia totalment empírica i hem hagut de realitzar les proves amb el robot real i en condicions reals sobre terrenys rugosos.
El robot de moment ha treballat sempre amb agents que han estat dissenyats amb controls pensats per a que el robot treballi en una superfície gairebé llisa (el passadís dels edificis de la universitat), per tant per tal de poder millorar les prestacions del robot en aquestes superfícies hem treballat sobre el terreny real.

Per fer-ho hem utilitzat una de les formes de control més utilitzades en el món de l'automàtica, la robòtica i l'electrònica en general, un control PID (Proporcional Integral Derivatiu) per tal de controlar les velocitats del robot.

5.1. La fricció

La fricció és l'element que produeix la causa del disseny d'aquest controlador.
Es defineix com la força de fregament o força de fricció, entre dos superfícies en contacte, a aquella que s'oposa al moviment entre ambdues superfícies (força de fricció dinàmica) o a la força que s'oposa al inici del moviment (força de fricció estàtica). Es genera a causa de les imperfeccions, especialment microscòpiques, entre las superfícies en contacte. Aquestes imperfeccions fan que la força normal entre ambdues superfícies no sigui perfectament perpendicular a aquestes, sinó que forma un angle φ amb la perpendicular (l' angle de fregament). Per tant, la força resultant es composa de la força normal (perpendicular a las superfícies en contacte) y de la força de fregament, paral·lela a les superfícies en contacte.

Llavors pel nostre cas veiem que aquesta força de fricció tant per la condició estàtica com per la condició dinàmica es veurà augmentada per tant dit d'una altre manera el robot es veurà frenat. Per això per tal de superar aquestes condicions el robot necessitaria segurament un increment de la velocitat aplicada a les rodes d'aquest per tal de poder funcionar en unes condicions acceptables. A la vegada segurament si aquestes condicions que treballarem sobre terres rugosos, les apliquéssim sobre els terres on s'han dissenyat els controladors fins ara el robot no tindria un funcionament adequat ja que la força de fricció seria molt menor i la que li aportarien les rodes seria excessiva.

En la següent figura veiem el perquè de la diferència entre la força de fregament estàtica i la força de fregament dinàmica.

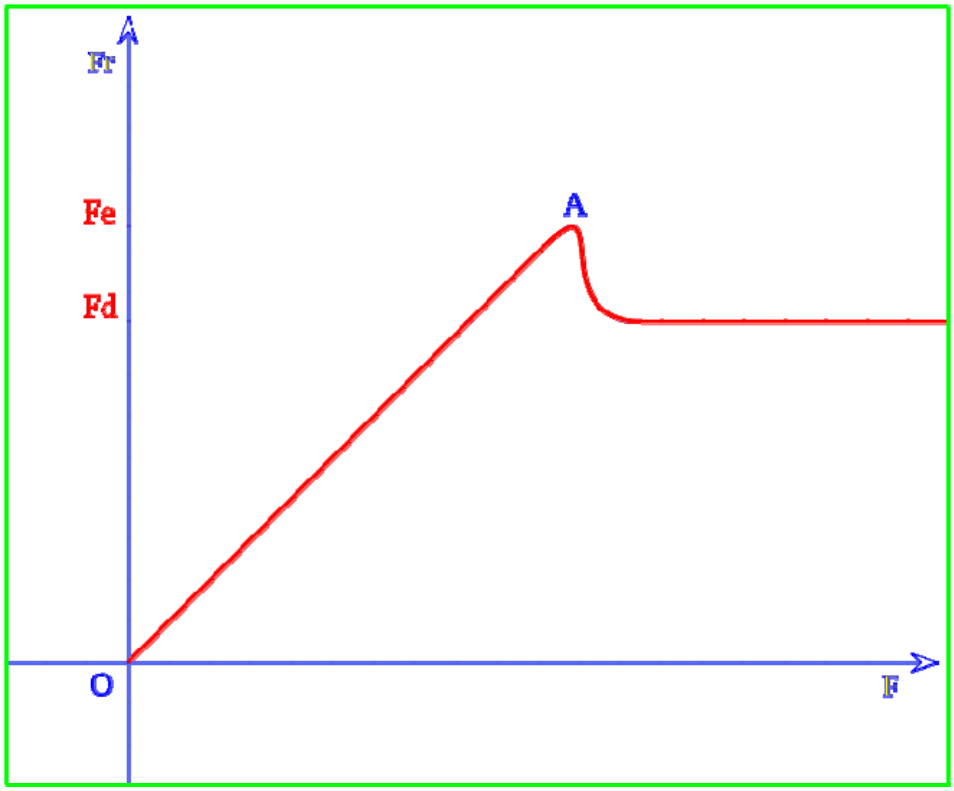

Figura 9. Diagrama de forces en el temps.

Observant aquest diagrama veiem ràpidament que la força de fricció augmenta fins a un màxim que és la força de fricció que tenim al inici del moviment, el que coneixem com a força de fricció estàtica. Un cop superem aquesta força veiem que aquesta es redueix fins a establir-se en un valor que és el que coneixem com a força de fricció dinàmica i que es redueix gràcies a la inèrcia que té en aquest cas el robot un cop té una certa velocitat.

En la següent figura veiem el diagrama de forces que se'ns presenta al inici del moviment, abans que el robot obtingui velocitat.

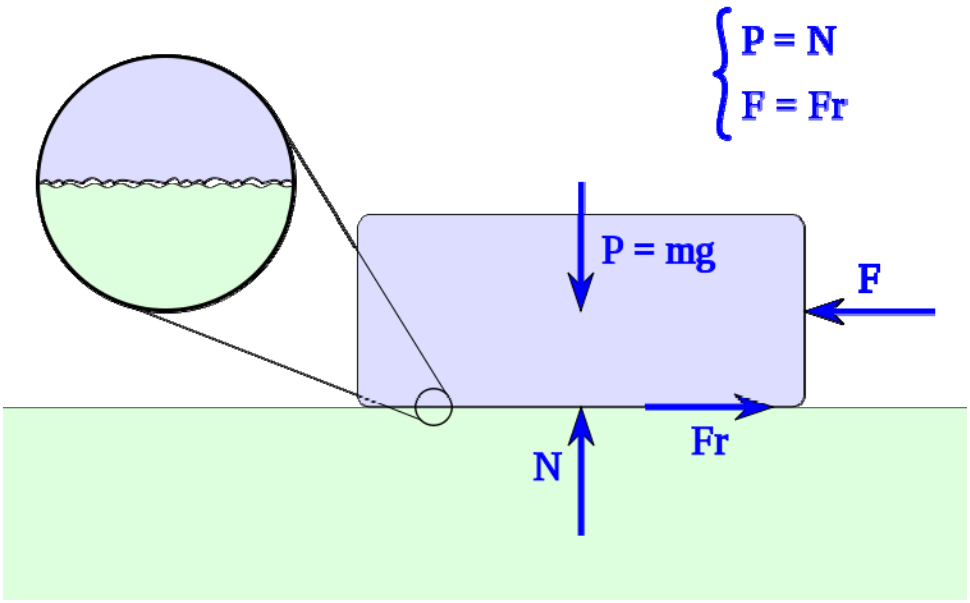

Figura 10. Diagrama de forces amb el robot sense inèrcia o velocitat, rodament estàtic.

On:

F: la força aplicada.

F_r: la força de fricció entre la superfície de recolzament i el cos, i que s'oposa al moviment

P: el pes del propi cos igual a la seva massa per l'acceleració de la gravetat.

N: la força normal, amb la que la superfície reacciona sobre el cos sostenint-lo

.

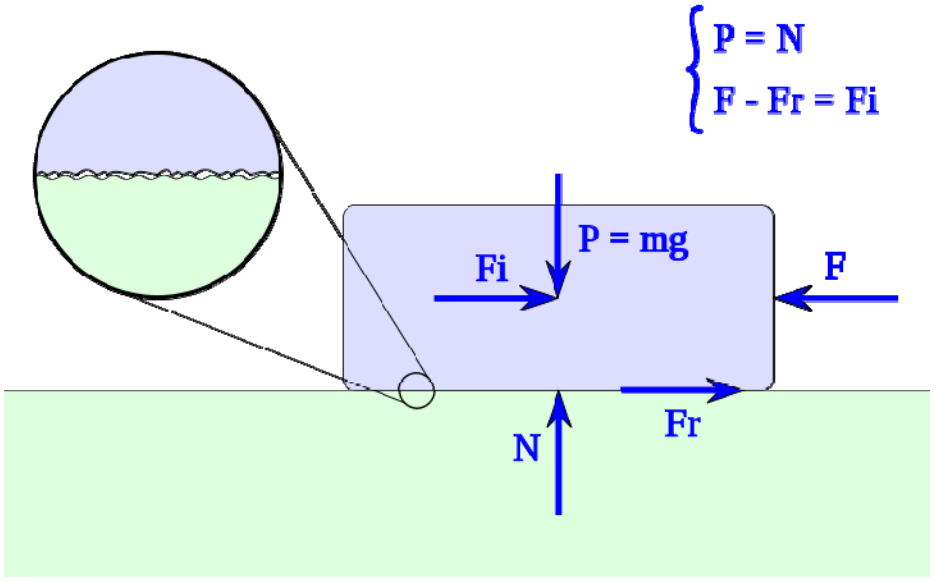

Figura 11. Diagrama de forces amb el robot amb inèrcia o amb velocitat, rodament dinàmic.

On:

F: la força aplicada.

F_r: la força de fricció entre la superfície de recolzament i el cos, i que s'oposa al moviment.

F_i: força d'inèrcia, que s'oposa a l'acceleració de cos, i que és igual a la massa del cos m per l'acceleració que pateix a.

P: el pes del propi cos igual a la seva massa per l'acceleració de la gravetat.

N: la força normal, amb la que la superfície reacciona sobre el cos sostenint-lo

Per tant doncs la fricció podem concluir que és l'element que provocarà una variació a l'hora del disseny del controlador ja que mostra una oposició al moviment al que estava habitualment acostumat en condicions normals el robot.

Per fer aquest controlador hem utilitzat un dels reguladors més comuns que existeixen en el món de l'electrònica i l'automàtica en general, el controlador de tipus PID.

5.2. Controlador PID

El nom de "controlador PID" respon a a l'acrònim PID que significa Proporcional-Integral-Derivatiu. Aquest controlador es fa servir dins del camp del control automàtic per a realitzar les operacions bàsiques de l'àlgebra lineal: el producte per un escalar, la suma i la resta. Aquest controlador és molt usat perquè ha estat molt estudiat i existeixen una bona quantitat de tècniques per a "sintonitzar-lo" (trobar els valors adients dels paràmetres que formen el PID) de tal manera que no cal ser molt expert en control per arribar, en molts casos, a un resultat força satisfactori pel que fa a la resposta del sistema de control. Algunes fonts assenyalen que aproximadament un 80% dels controladors que es poden trobar a la indústria són del tipus PID (estariem parlant també de P, PI, PID, i en molt baix nombre PD).

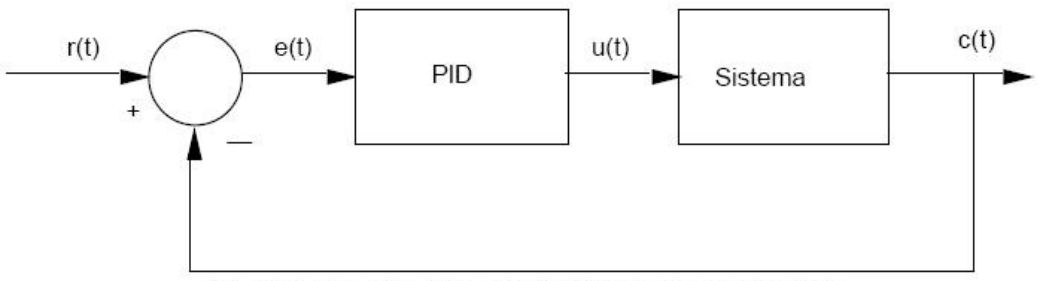

Figura 12. Sistema en llaç tencat amb el controlador PID en sèrie.

Un PID com hem comentat anteriorment realitza tres operacions:

Proporcional: efectua l'operació:

$$u_1 = K_p \cdot e(t) \quad \text{(Eq.11)}$$

Integrador: efectua l'operació:

$$u_2(t) = \frac{1}{T_i} \int_0^t e(\tau)d\tau \quad \text{(Eq.12)}$$

Derivador: efectua l'operació:

$$u_3(t) = T_d \frac{de}{dt} \quad \text{(Eq.13)}$$

On $u_i(t)$ és la sortida de cadascun dels termes del controlador, e(t) és el senyal d'error, K_p és la constant proporcional, T_i és el temps integral i T_d és el temps derivatiu.

El controlador PID té diverses estructures que veurem tot seguit.

L'anomenada estructura bàsica és la que correspon a l'equació:

$$u(t) = K_p[e(t) + K_i \int_0^t e(\tau)d\tau + T_d \frac{de}{dt}] \quad \text{(Eq.14)}$$

La qual es pot reescriure de la següent manera:

$$u(t) = K_p e(t) + \frac{K_p}{Ti} \int_0^t e(\tau)d\tau + K_p T_d \frac{de}{dt} \quad \text{(Eq.15)}$$

o també com:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} \quad \text{(Eq.16)}$$

Així doncs trobem que:

$$K_i = \frac{K_p}{T_i} \quad \text{(Eq.17)}$$

$$K_d = K_p T_d \quad \text{(Eq.18)}$$

Per tant ara estem en disposició de parlar de sintonitzar el PID pel funcionament correcte del robot en condicions per terres rugosos. Quan parlem de sintonitzar no vol dir altre cosa que donar-li o trobar uns valors òptims per a la terna Kp, Ti i Td o, si es vol, com serà pel nostre cas, tenim l'alternativa de sintonitzar els valors de Kp, Ki i Kd.

A continuació veurem una taula on es mostren els efectes de les accions de control en els reguladors PID:

ACCIÓ DE CONTROL	EFECTE EN EL RÈGIM TRANSITORI I ESTABILITAT	EFECTE EN EL RÈGIM PERMANENT	ACCIÓ DAVANT PERTORBACIONS
PROPORCIONAL	Disminueix el temps de Resposta del sistema, fent-lo més ràpid. En sistemes d'ordre 2 o Superior, tendeix a fer la resposta més oscil·lant i inclús inestable.	En augmentar el guany, disminueix l'error estacionari en sistemes de primer ordre (tipus 0). En sistemes de tipus 1 o superior no millora el règim permanent.	No es capaç d'eliminar l'efecte d'una pertorbació que es manté en el temps, malgrat que pot reduir-lo augmentant el guany.
INTEGRAL	EMPITJORA EL TRANSITORI Augmenta el sobrepic i el temps d'establiment del sistema, fent-lo encara més oscil·lant o inclús inestable.	MILLORA L'ESTACIONARI Elimina totalment l'error estacionari a una entrada graó.	Elimina l'efecte de pertorbacions sostingudes en el temps.
DERIVATIU	MILLORA EL TRANSITORI Redueix el temps d'establiment. Augmenta l'esmorteïment i disminueix el sobrepic màxim.	No actua en règim permanent (derivada nula) per tant no pot corregir l'error estacionari.	No exerceix cap acció davant de pertorbacions constants, per tant no és capaç d'eliminar el seu efecte.

Taula 2. Efecte de les accions de control en els reguladors PID.

5.3. Sintonització del PID

Les característiques d'un bon control són difícils de definir de manera genèrica i depenen molt del sistema sobre la que estem treballant. No obstant, podem afirmar que en un sistema de control realimentat s'han de verificar les següents qüestions:

El sistema que regulem ha de ser estable. El sistema de regulació ha de ser suficientment ràpid, ha d'estar adequadament amortitzat, el sistema ha de tenir una determinada precisió en règim estacionari i ha d'atenuar els efectes dels sorolls, pertorbacions i canvis de càrrega mentre es manté la planta amb una consigna constant.

Per aconseguir aquests objectius, el controlador PID és el més utilitzat en regulació de processos industrials. Es calcula que, a l'actualitat, al voltant d'un 90% dels llaços de control a la indústria utilitzen alguna forma o variant d'aquest regulador PID.
En moltes dificultats ens soluciona els problemes: elimina l'error estacionari amb l'acció integral i pot millorar el transitori amb l'acció derivativa.
Un cop hem decidit que utilitzarem un regulador de tipus PID s'ha d'efectuar l'ajust dels paràmetres (sintonització) perquè la resposta en llaç tancat tingui unes determinades característiques. Es calcula que un 30% dels llaços de control industrials oscil·len a causa d'un mal ajustatge dels paràmetres del controlador.

Existeixen varis mètodes de disseny i ajustatge de controladors PID: analítics, per proba i error i empírics. Els mètodes empírics no necessiten un model de la planta i d'aquí la importància que tenen en el món industrial.

Per sintonitzar el PID hem fet servir un mètode de poba i error. Per això, hem utilitzat una sèrie de regles simples per la sintonització del controlador.
Obtenir una resposta del procés i determinar que és necessari millorar en el seu comportament, en termes de especificacions de control. Per exemple fixar-nos en la velocitat del robot a l'hora de realitzar el trajecte, la precisió d'aquest, les oscil·lacions que pateix en el trajecte, etc.
Provar amb un control proporcional P i verificar si es compleixen les especificacions elevant progressivament el guany proporcional Kp. Si no es verifiquen, ajustar el guany per millorar el temps de resposta.
Afegim un control integral per eliminar l'error en règim permanent. Es possible que empitjorem el transitori.
Afegim el control derivatiu D per a millorar el transitori, reduint el sobrepic i el temps d'establiment.
Per tal de realitzar totes aquestes proves hem treballat sobre una superfície rugosa semblant a l'asfalt de qualsevol carretera o del que podríem trobar per exemple a l'aparcament de l'edifici PII de la Politècnica de l'UdG.

Per tant hem dissenyat un agent GoTo que controla les velocitats lineals i angulars del robot, que són les que aporten el moviment i l'orientació del robot. Tot això ho hem fet mitjançant la implementació d'un controlador PID. Per fer-ho hem utilitzat altre vegada l'estructura de programació en C++ igualment que pel controlador anterior pel mateix motiu que abans. L'estructura del programa amb arquitectura en C++ s'adjunta a l'annex de la memòria per poder consultar-ne la forma si resulta necessari.

6. Resultats

6.1. Introducció

Amb l'objectiu de comprovar la bondat dels agents GoTo dissenyats, hem fet un seguit d'experiments tant per al mètode de persecusió pura (simulació i real) com per al PID per superfícies rugoses (real).

Per al cas del GoTo de persecusió pura, hem considerat els mateixos mapes que els emprats per a l'agent GoTo col·laboratiu per obtenir resultats simulats, de manera de poder comparar els dos agents dissenyats per a controlar el moviment del robot a l'interior. Aquests mapes representen gràficament i amb claredat l'assolimnet dels objectius proposats a l' inici d'aquest projecte.

El primer mapa de tots és el GoTo Map, dissenyat sense cap tipus d'obstacle que ens permet veure que no em perdut traçabilitat canviant el mètode de càlcul de les variables que defineixen l'agent i que el comportament del robot per anar a un punt determinat amb l'orientació desitjada s'aconsegueix.
Per a realitzar aquestes simulacions no només ens hem quedat amb la representació de la trajectòria del simulador, sinó que hem anat més enllà i hem extret les trajectòries generades pel simulador mitjançant la base de dades que crea aquest mateix, i les hem implementat en Matlab ®, tot això per tal de poder comparar el resultat obtingut amb el de la trajectòria ideal.

El altres escollits, són una sèrie de mapes que representen la interacció del nou agent GoTo amb els altres agents de l'arquitectura de control, que era un dels objectius que ens marcàvem a l'inici del projecte. En aquest cas és la interacció amb els agents Avoid i GoThrough que gestionen el comportament del robot amb l'aparició d'obstacles en la trajectòria d'aquest en el primer, i la circulació del robot per passadissos estrets pel segon.

Seguidament hem fet alguns d'aquests experiments amb el robot real, no per comparar amb l'antic GoTo sinó per comprovar que el robot té el comportament adequat en el món real.

Per al cas del GoTo dissenyat empíricament i degut a la impossibilitat de modificar els paràmetres que representen el terreny al simulador, només s'han fet els experiments amb el robot real.

A continuació es presenten els resultats per als dos controladors, en simulació i real i per als diferents escenaris (mapes).

6.2. Resultats per trajectòries sense obstacles

Per comprovar aquestes trajectòries hem realitzat diverses simulacions en diferents condicions per tal d'assegurar-nos que la resposta era l'adequada.

A la vegada com hem comentat abans, la resposta s'ha traspassat, mitjançant la base de dades creada, a l'entorn Matlab® en Windows® per així poder veure amb major claredat com segueix la trajectòria el robot.

En el primer cas hem buscat segurament el més senzill que li podríem donar al robot i és una situació de partida en el punt [0,0] amb un punt objectiu de [7000,7000]. Diem que és senzill ja que el pendent és sempre 1 i per tant és una trajectòria relativament senzilla.

Primerament veurem la simulació directament amb el Mobile Sim per veure com és el camí que descriu el robot.

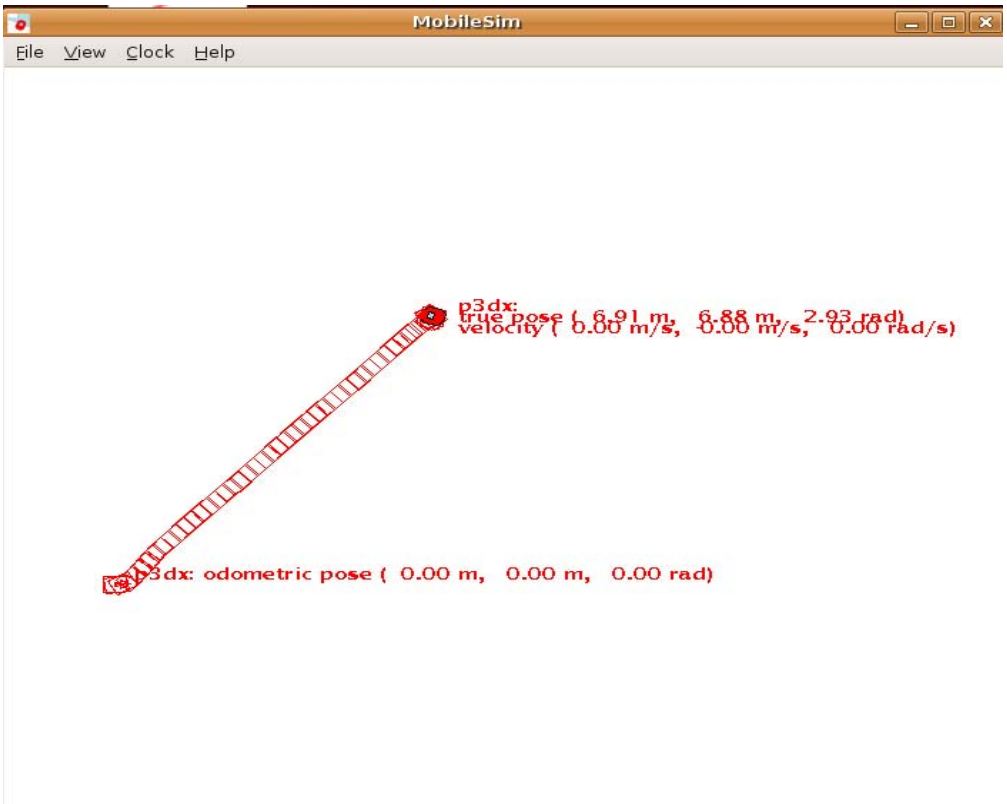

Figura 13. Simulació en Mobile Sim de la trajectòria per un punt objectiu [7000,7000].

Seguidament mostrem la comparació del resultat de la trajectòria obtinguda amb la trajectòria que hauria de ser realment.

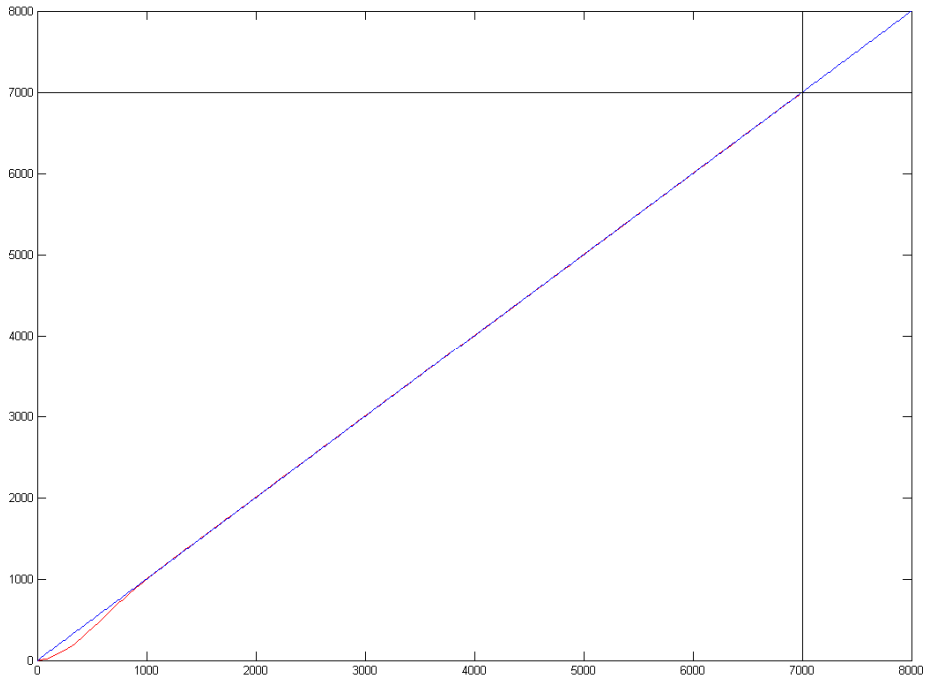

Figura 14. Simulació en Matlab ® de la trajectòria obtinguda amb la trajectòria teòrica.

Com es pot veure a la figura 14, la trajectòria és gairebé igual a la teòrica i es mou en la major part dels moments sobre la mateixa línia. Únicament es veu com al inici del moviment fa una petita corba per aproximar-se a el camí correcte, llavors segueix perfectament el camí.

Pel que fa a la precisió a l'hora d'acostar-se al punt objectiu també podríem dir que hi ha una precisió acceptable, com es mostra en el detall ampliat a la figura 15.

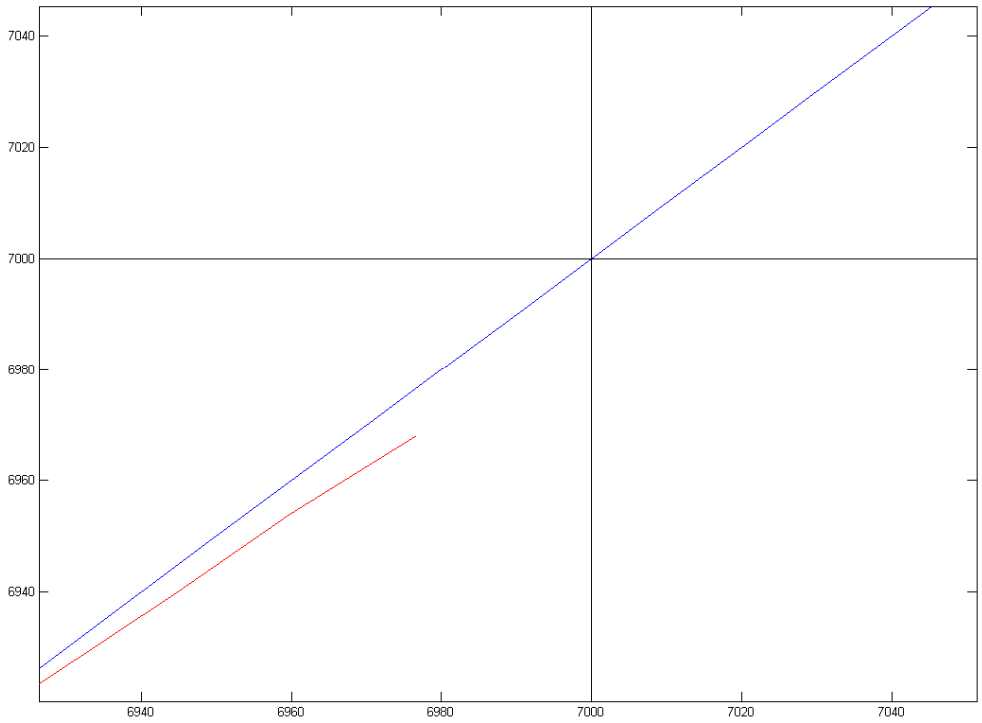

Figura 15. Aproximació al punt objectiu.

Podem considerar que tenim un error força acceptable ja que per la banda de la component x l'error és de uns 20 mm i per part de la component y l'error no supera els 30 mm. Tenint en compte les mides del robot l'error seria gairebé inapreciable en la situació real.

Hem realitzat una segona i una tercera simulació de característiques semblants però amb diferents punts aquesta vegada amb pendents diferents a 1.

En el següent cas estem parlant d'un punt objectiu que es troba a una distancia de 2000 mm en component x i a 8000 mm en component y i les respostes en el simulador són les següents.

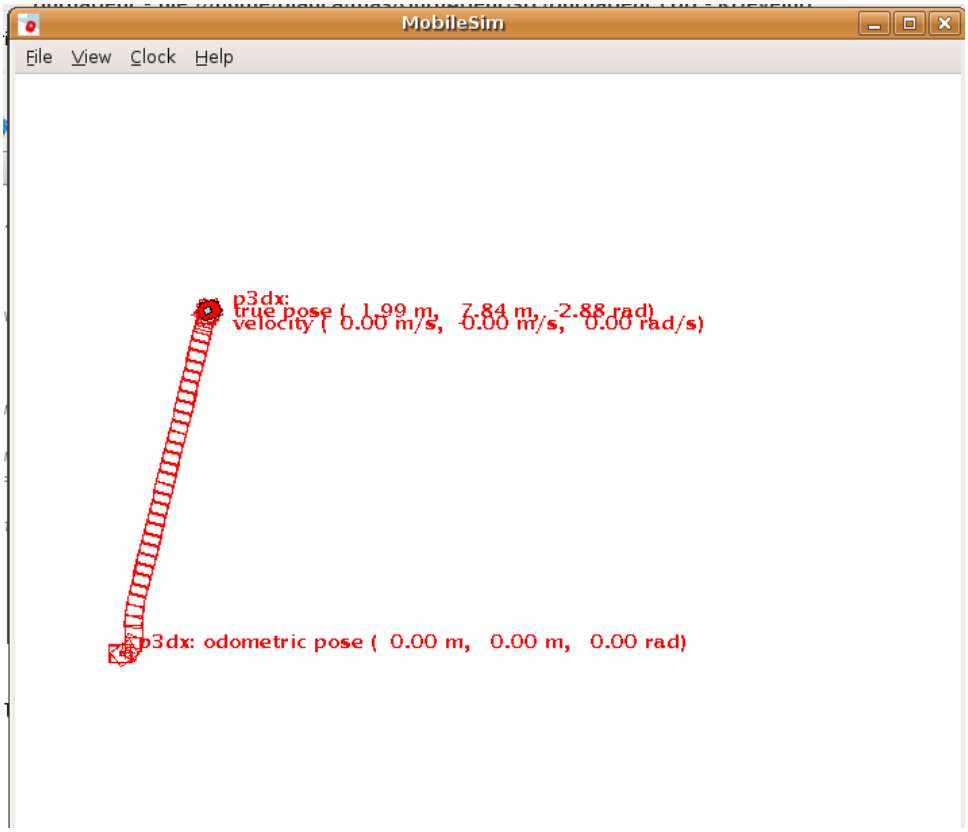

Figura 16. Simulació en Mobile Sim de la trajectòria per un punt objectiu [2000,8000].

Seguidament veiem la resposta obtinguda comparada amb la resposta ideal.

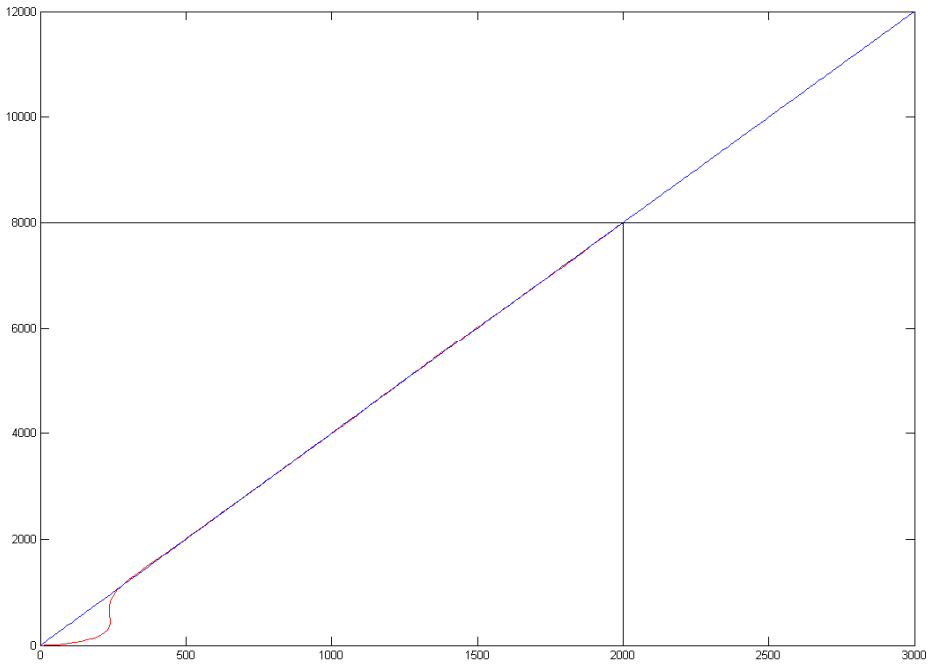

Figura 17. Simulació en Matlab ® de la trajectòria obtinguda amb la trajectòria teòrica.

Igualment que en el cas anterior la resposta obtinguda és força fidel a la teòrica per tant podem dir que és un resultat acceptable pel que fa al seguiment del camí correcte.

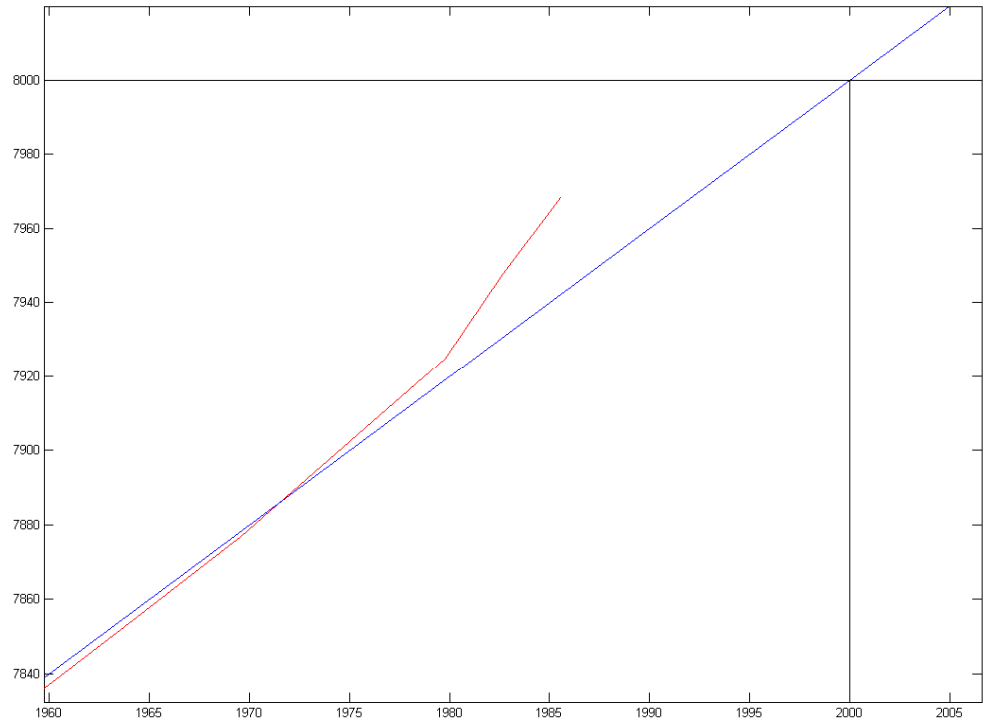

Figura 18. Aproximació al punt objectiu.

Si mirem l'error respecte al punt objectiu també considerem que és molt acceptable ja que per part de component x és aproximadament d'uns 15 mm i per part de component y d'uns 25 mm. Tal i com hem comentat abans són errors gairebé inapreciables.

També s'ha realitzat la mateixa operació per un punt objectiu de [8000,4000] i per tant amb un pendent més baix i les respostes han estat les següents.

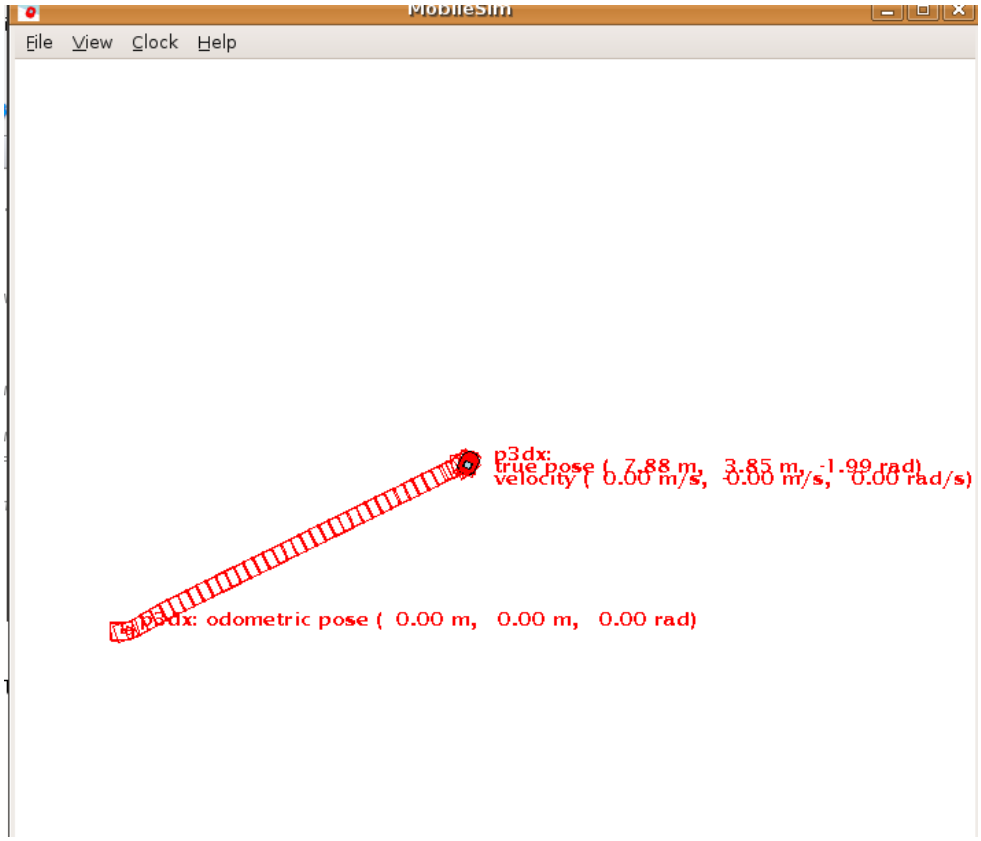

Figura 19. Simulació en Mobile Sim de la trajectòria per un punt objectiu [8000,4000].

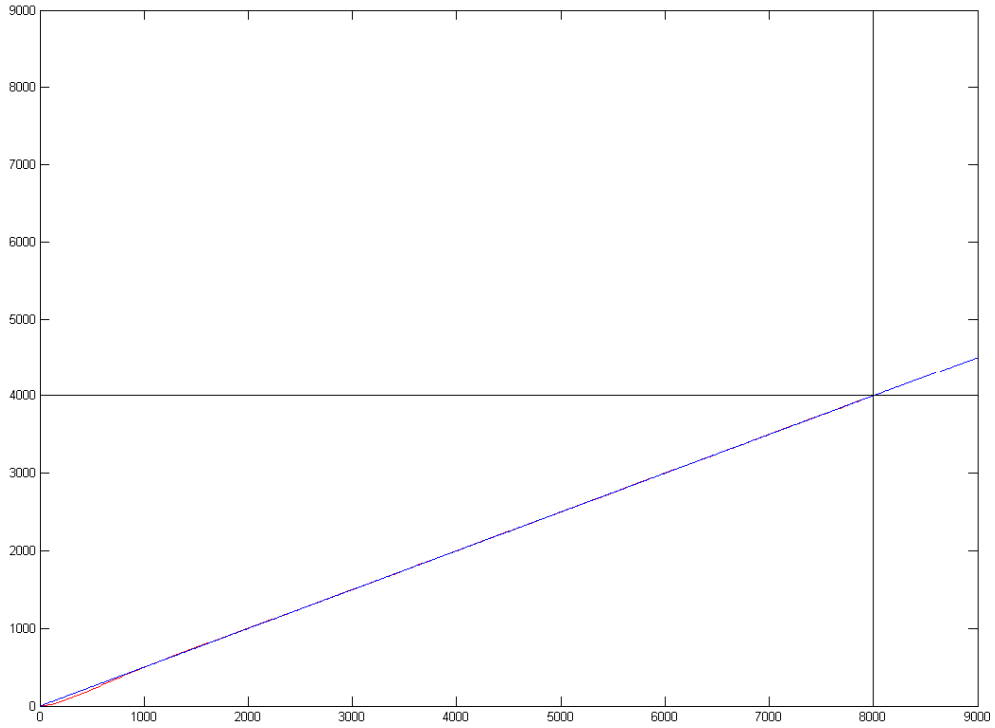

Figura 20. Simulació en Matlab ® de la trajectòria obtinguda amb la trajectòria teòrica.

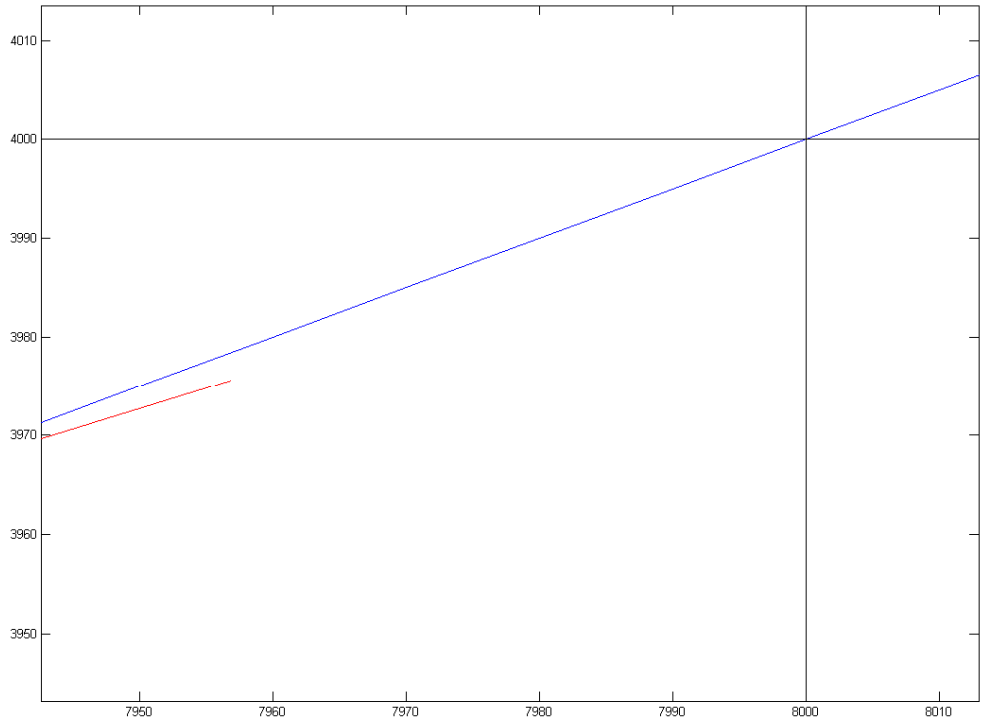

Figura 21. Aproximació al punt objectiu.

Tant pel que fa a resultat com pel que fa a error el resultat és molt pròxim. L'error en x és d'uns 40 mm mentre que en y és d'uns 35 mm. És un error un pèl més elevat que els dos anteriors però segueix essent petit.

A continuació hem realitzat dos assajos més una mica especials. No són especials per la dificultat en quant a la trajectòria sinó perquè al llarg del procés d'investigació del projecte ens hem adonat que quan el robot es mou sobre els eixos de coordenades acostuma a donar més problemes.

En el primer cas ens centrem en el moviment sobre l'eix x, és a dir quan la trajectòria del robot ha de ser totalment rectilínia. Els resultats que hem obtingut són per un punt objectiu de coordenades [7000,0].

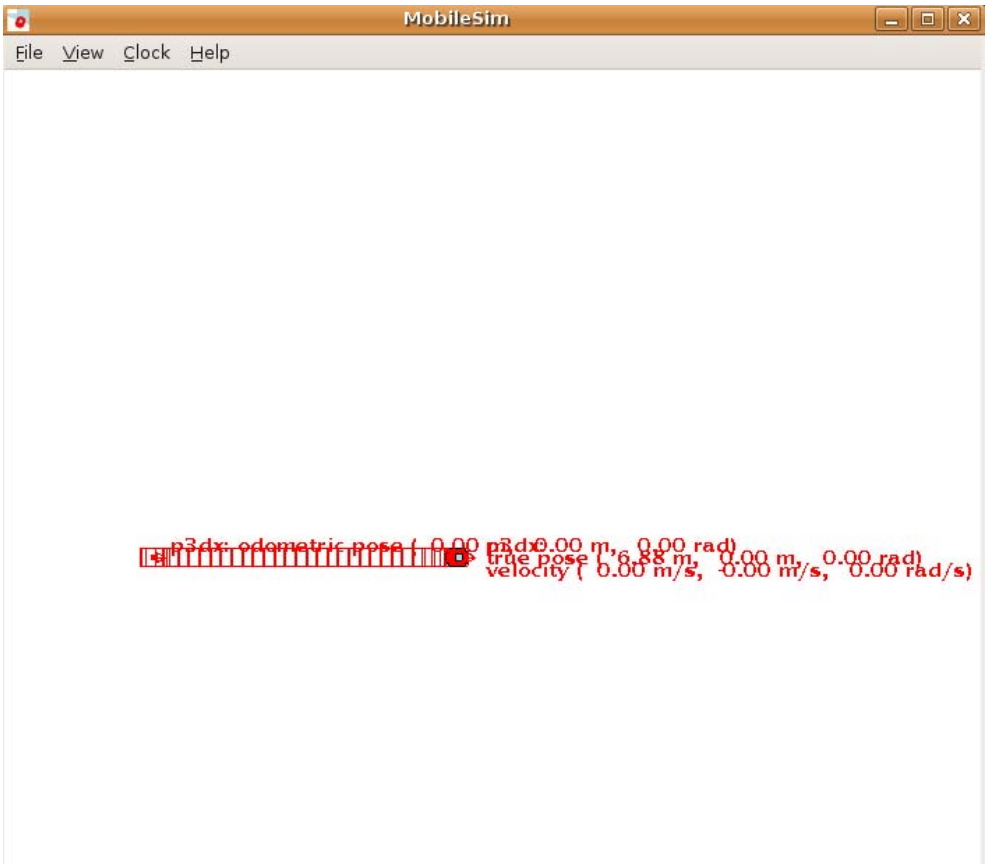

Figura 22. Simulació en Mobile Sim de la trajectòria per un punt objectiu [7000,0].

Fixant-nos només en el resultat en simulació ja podem observar que la trajectòria és aparentment rectilínia i no pateix cap alteració ni desviació mentre hi ha desplaçament. El que si veiem és que hi ha l'habitual error en la precisió final, però com hem dit en casos anteriors és un error gairebé inapreciable tenint en compte les dimensions del robot.

Per aquest cas també hem realitzat les corresponents gràfiques amb Matlab ® per poder comprovar la fidelització amb la trajectòria teòrica. Els resultats obtinguts han estat els següents.

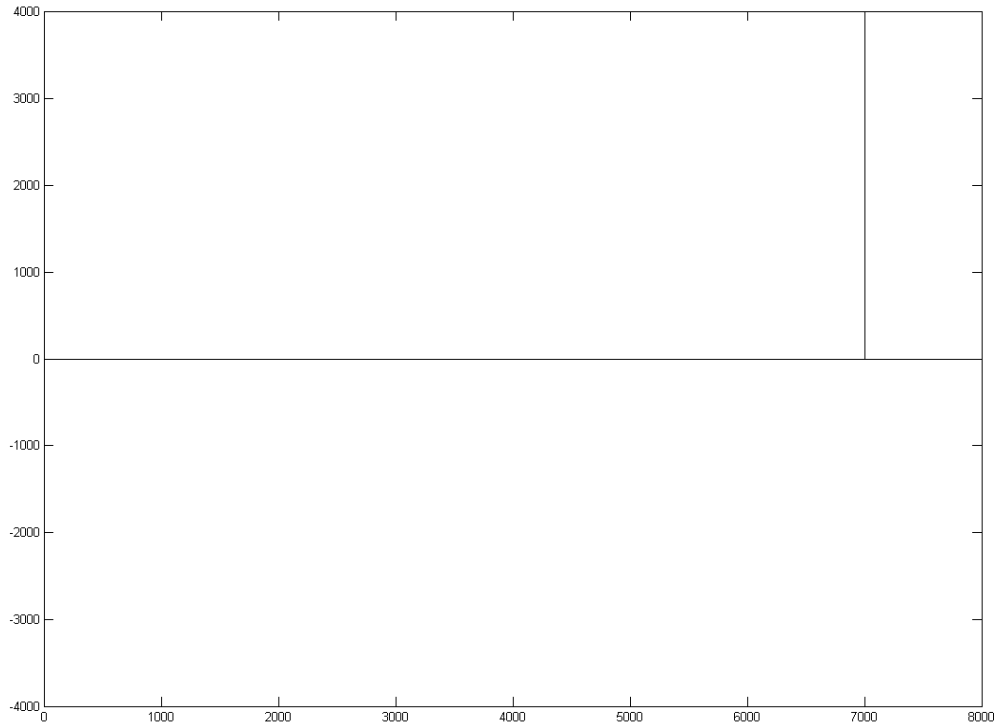

Figura 23. Simulació en Matlab ® de la trajectòria obtinguda amb la trajectòria teòrica.

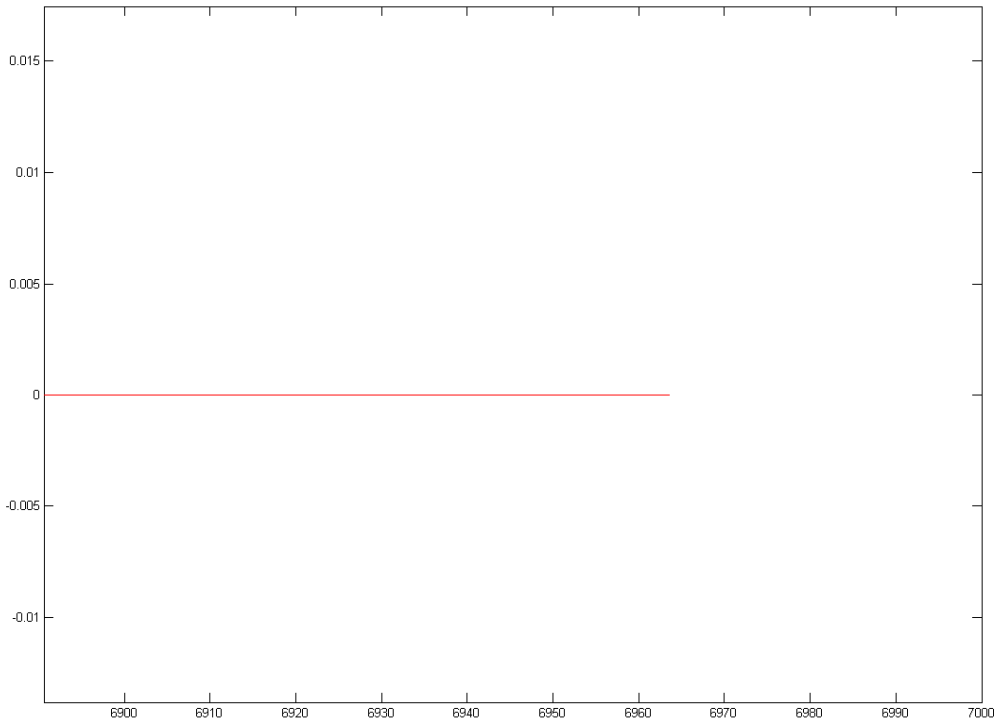

Figura 24. Aproximació al punt objectiu.

Si observem la figura 23 de seguida veiem que la trajectòria és exactament la que ha de tenir ja que és impossible diferenciar la teòrica de la obtinguda per tant compleix perfectament les expectatives inicials.
Pel que fa a l'aproximació al punt objectiu la diferència s'aproxima als 30 mm per tant és gairebé inapreciable per les mides que estem tractant.

Per acabar hem realitzat el mateix assaig que l'anterior amb la mateixa distància de recorregut però aquesta vegada amb desplaçament sobre l'eix y. Ara el robot no tindrà inicialment la orientació adequada així que el resultat previsiblement ha de ser lleugerament diferent al cas anterior. Els resultats obtinguts han estat els següents.

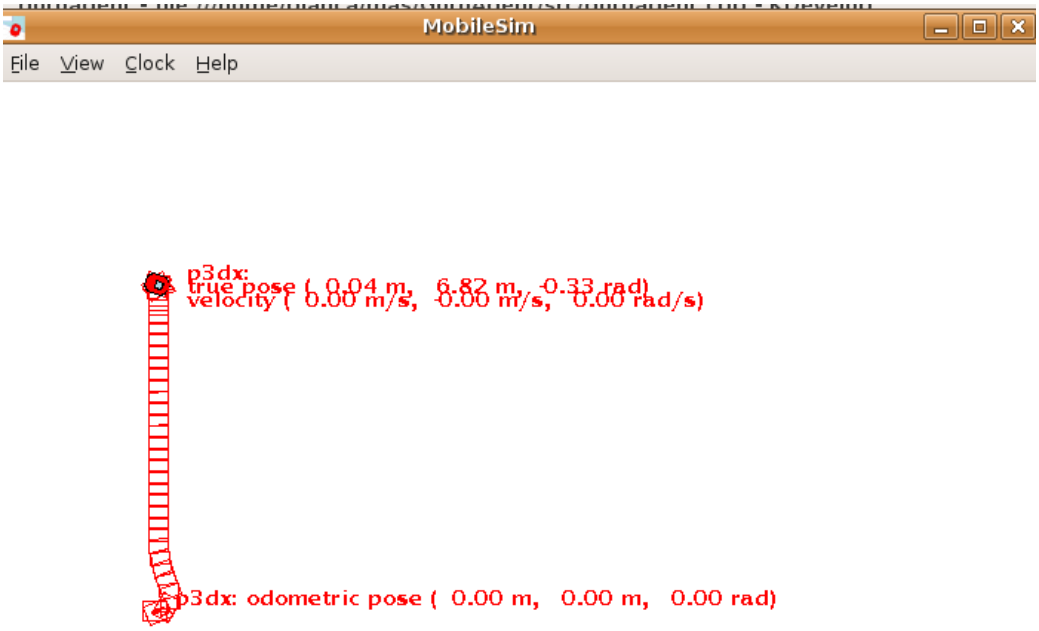

Figura 25. Simulació en Mobile Sim de la trajectòria per un punt objectiu [0,7000].

Observant a simple vista ja veiem que el robot s'ha reorientat sobre l'eix y sense majors problemes i a més a més ho ha fet amb una rapidesa molt acceptable. Podrem veure amb major precisió les dades representades en l'entorn Matlab ®.

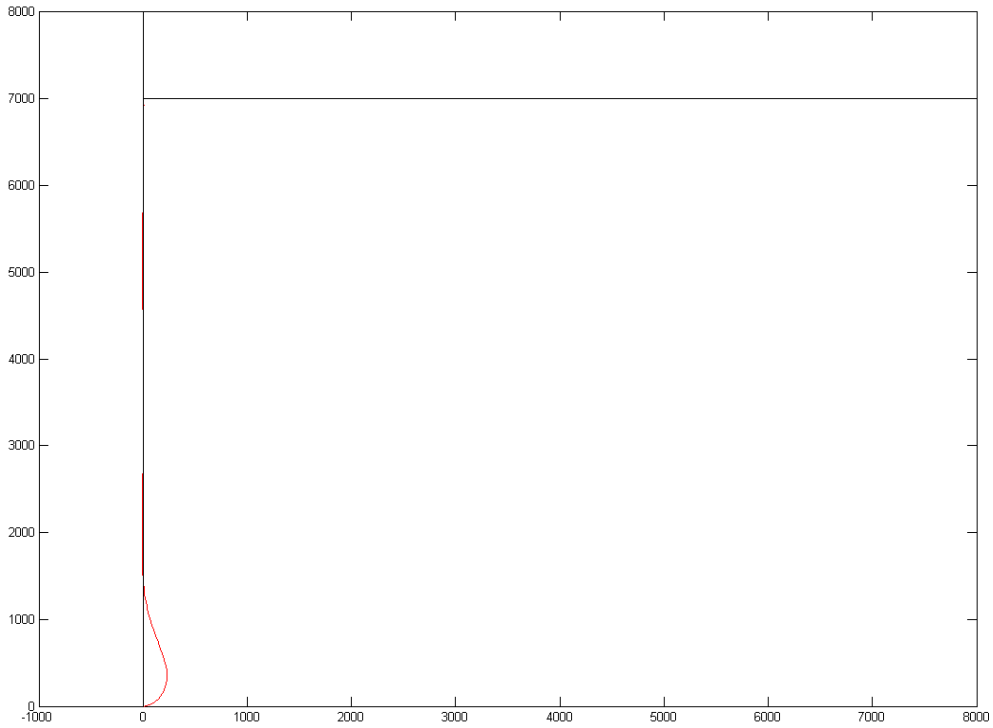

Figura 26. Simulació en Matlab ® de la trajectòria obtinguda amb la trajectòria teòrica.

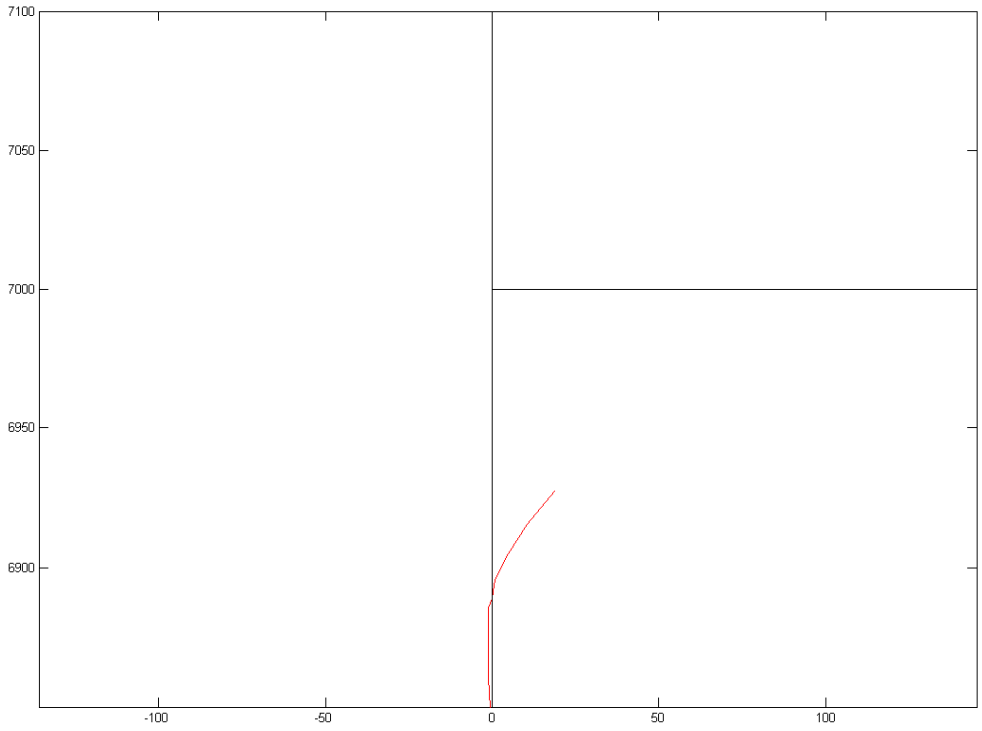

Figura 27. Aproximació al punt objectiu.

Observant la figura 27 veiem que l'aproximació al punt objectiu també és correcte ja que tant sols hi ha una diferència d'uns 50 mm en component i segueix essent un error acceptable per les distàncies que parlem.

6.3. Altres resultats en simulació per a diferents mapes

6.3.1. Mapa avoid

Després de comprovar el funcionament del robot per a trajectòries sense obstacles el següent pas ha estat comprovar el funcionament en el moment en que el vehicle es troba un obstacle al mig del seu camí.

Dit d'una altre manera s'ha comprovat la correcta interacció de l'agent GoTo dissenyat amb els altres agents que formen l'estructura multi-agent del robot. Per a fer-ho hem utilitzat un mapa en el simulador en el que hi col·loquem un objecte de forma rectangular que bé podria ser un banc, una jardinera o qualsevol altre objecte de forma similar. Evidentment en el cas real ens interessaria que el robot esquivés aquest element i que fos capaç de reconduir-ne la trajectòria i així poder arribar al seu punt destí.

L'agent de l'estructura que actuarà en aquestes condicions és l'Avoid. Per tant el que hem fet és comprovar la correcta interactuació de l'agent GoTo amb l'agent Avoid en el moment en que els sensors del robot detecten un obstacle i el resultat ha estat el següent.

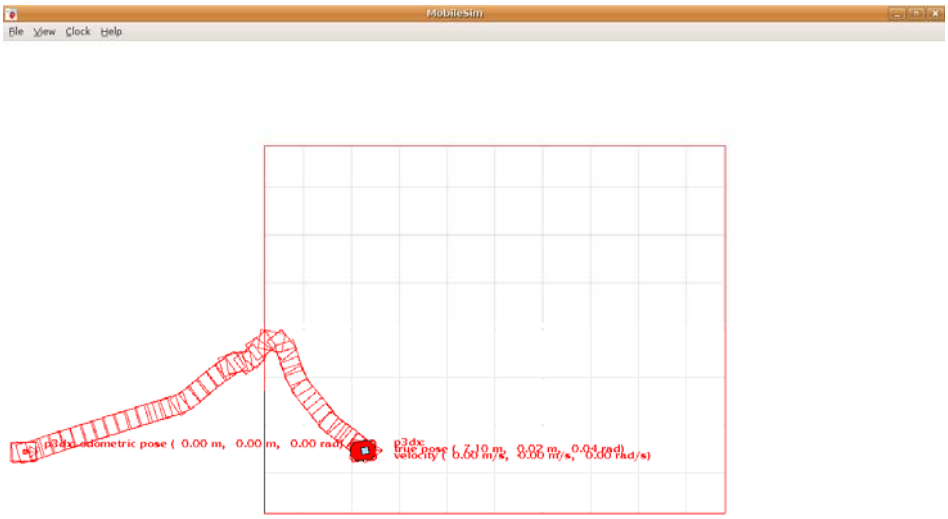

Figura 28. Recorregut del robot esquivant un obstacle.

El robot genera la seva trajectòria normalment mitjançant l'agent GoTo el qual intenta dirigir el robot en línia recta fins al punt. Arriba un moment però en que els sensors del robot detecten un obstacle a la proximitat requerida per tal que l'agent Avoid entri en funcionament.

En aquest instant, l'agent GoTo comença a interaccionar amb l'agent Avoid (ja implementat en la plataforma de control). El que fa és generar una trajectòria alternativa que voreja l'obstacle i evita el xoc amb aquest mateix.

Finalment el robot segueix el seu curs fins al punt objectiu i s'atura amb un error gairebé mínim tenint en compte les mides del robot.

També s'ha realitzat la simulació per un mapa de característiques similar però aquesta vegada introduint un objecte amb certa profunditat i el resultat obtingut ha estat el següent.

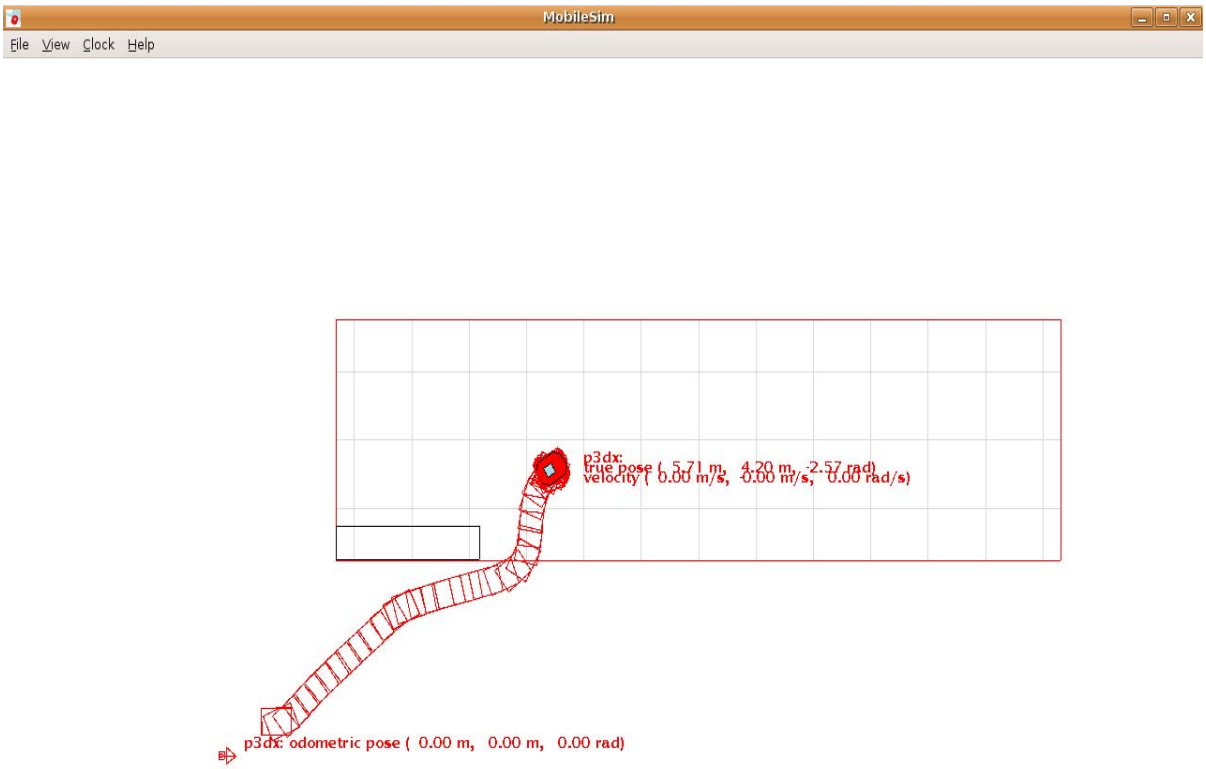

Figura 29. Recorregut del robot esquivant un obstacle amb profunditat.

El punt objectiu en aquest mapa estava situat a la posició (5700,4200) amb l'inconvenient que en aquest cas trobem un obstacle de dos metres de longitud amb certa profunditat i que

es troba al mig de la trajectòria. Tal i com podem observar a la figura 29 el robot recondueix la seva posició per tal de no xocar amb l'obstacle i poder així arribar a la seva destinació.

6.3.2. Mapa GoThrough

A la vegada que hem realitzat una sèrie de simulacions en un mapa on hi aparegui un obstacle, com és el cas anterior, s'ha comprovat també el correcte funcionament del robot per un mapa on aquest hagi de circular pel mig de dos obstacles, el que en terrenys reals podria ser un passadís, per posar un exemple clar.

En el mapa en qüestió ens trobem amb que el robot es troba a l'extrem d'un passadís i ha de circular fins a l'altre extrem ja que el seu punt objectiu es troba al final del passadís.
El cas en particular del que parlem és el que podrem observar a la figura 30.

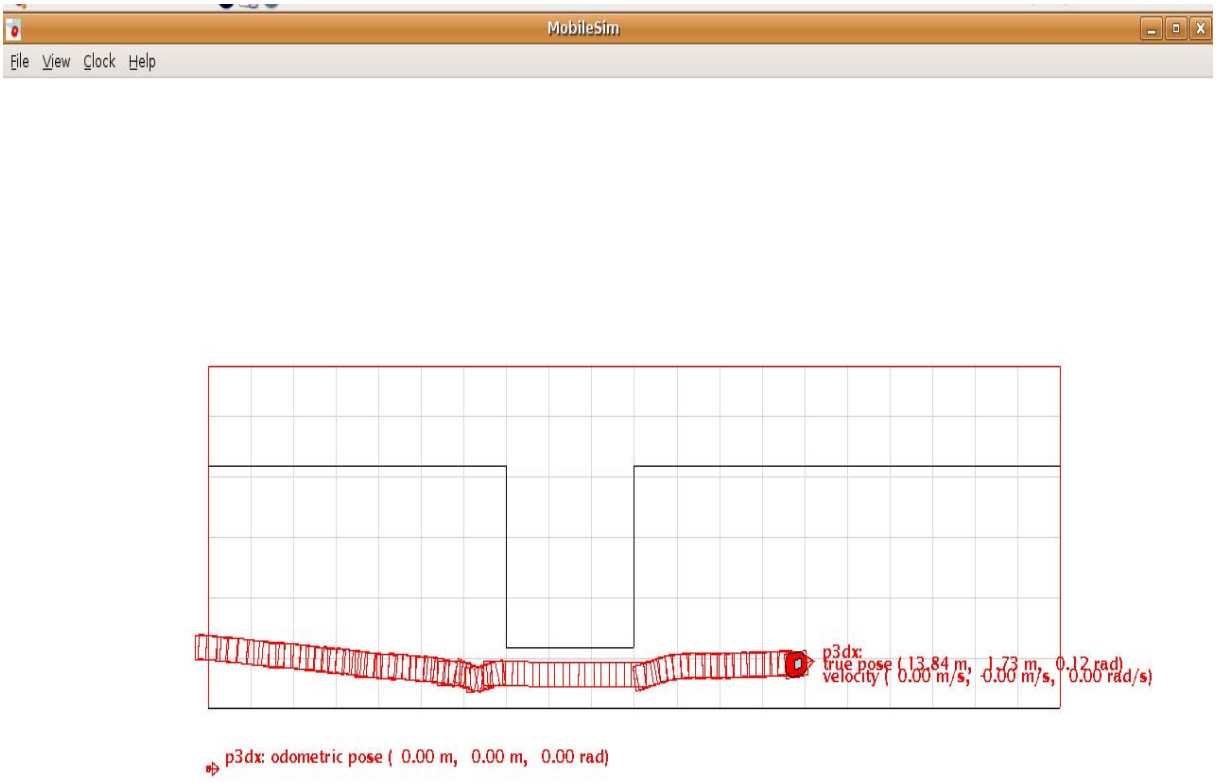

Figura 30. Recorregut del robot passant per un passadís estret.

Si observem el mapa veiem que el robot s'acosta fins a l'entrada del passadís, on només intervindrà l'agent GoTo, i un cop detecta la presència d'obstacles a un costat i a l'altre del robot aquest interactua amb l'agent GoThrough que el fa circular pel mig del passadís sense

problemes. Un copa aquest arriba al final i els sonars del robot deixen de detectar obstacles als costats torna a actuar únicament l'agent GoTo fins a arribar al punt objectiu.

6.3.3. Mapa habitació

Per acabar de simular de manera inicial en diferents mapes, hem complicat una mica més la situació. Hem utilitzat un mapa en el qual hi trobem diferents habitacions que es troben connectades entre elles mitjançant un passadís central.

El que hem demanat al robot en aquest cas és que sortís de l'habitació en la que es troba inicialment, sortejant l'obstacle que hi ha en aquesta mateixa, passés pel passadís central d'una punta a l'altre i que acabés entrant en una de les habitacions.
El resultat que hem obtingut és el que es pot veure a la següent figura.

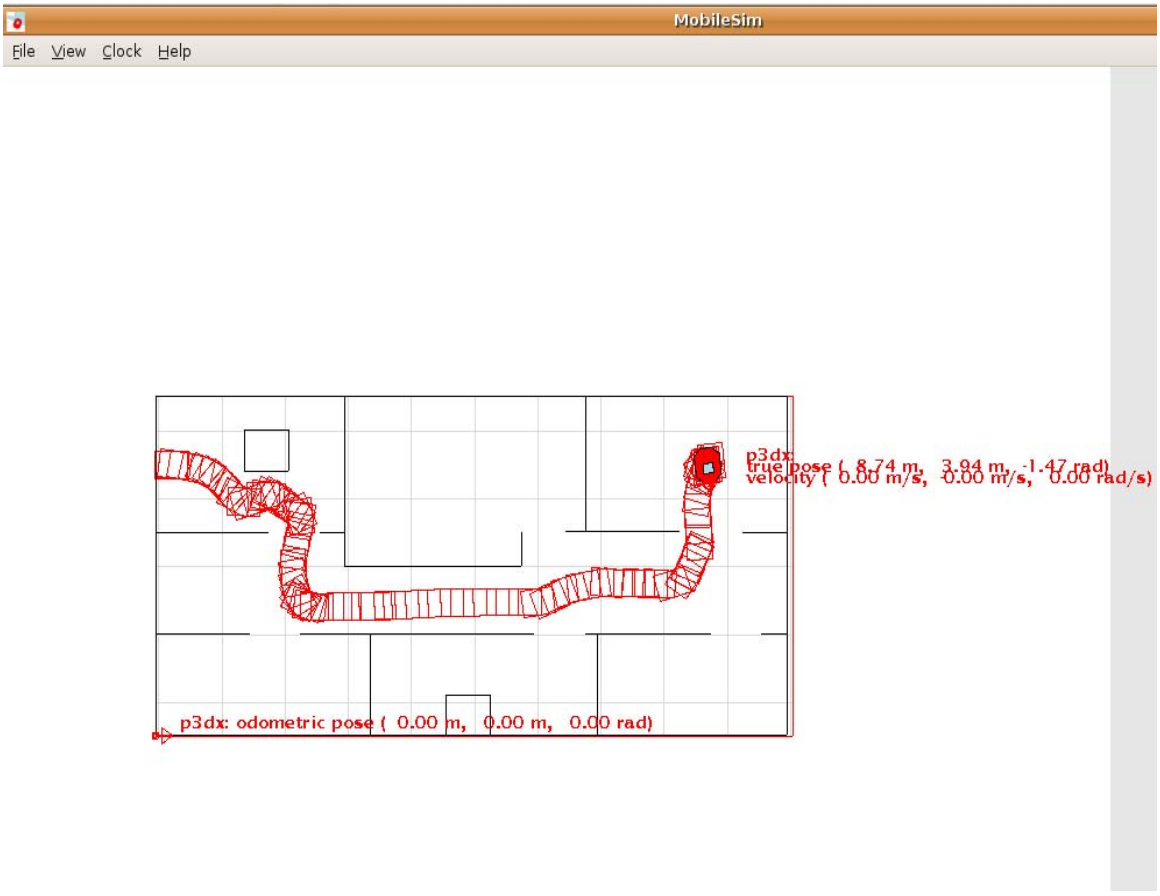

Figura 31. Recorregut del robot dins el mapa amb habitacions.

Com es pot observar el robot parteix d'una habitació on per tal de sortir-ne prèviament ha d'esquivar un obstacle. Un cop evita aquest obstacle amb la intervenció de l'agent Avoid, el robot surt de l'habitació i enfila en direcció a l'habitació on hi ha el punt objectiu. Prèviament passa pel passadís central on ha d'intervenir l'agent GoThrough per tal que el robot es

mogui de manera correcte. Finalment enfila direcció al punt objectiu entrant a l'habitació evitant les parets d'aquesta mateixa.

6.3.4. Mapa habitació amb petit obstacle (persona) en la trajectòria

Un altre cas real que es podria trobar el robot i que no està de més comprovar, és el que hem realitzat per acabar. Hem comprovat quina seria la resposta que tindria el robot en el cas que es trobés un petit obstacle en el camí, en altres paraules el que passaria si per qüestions incontrolables en aquell moment aparegués una persona, una motxilla o qualsevol altre petit objecte que no estava previst.

La resposta que hem obtingut es pot considerar satisfactòria ja que l'ha esquivat sense majors problemes i ha seguit la trajectòria desitjada sense majors conseqüències en el resultat final.

La resposta obtinguda és la que trobem en la figura següent.

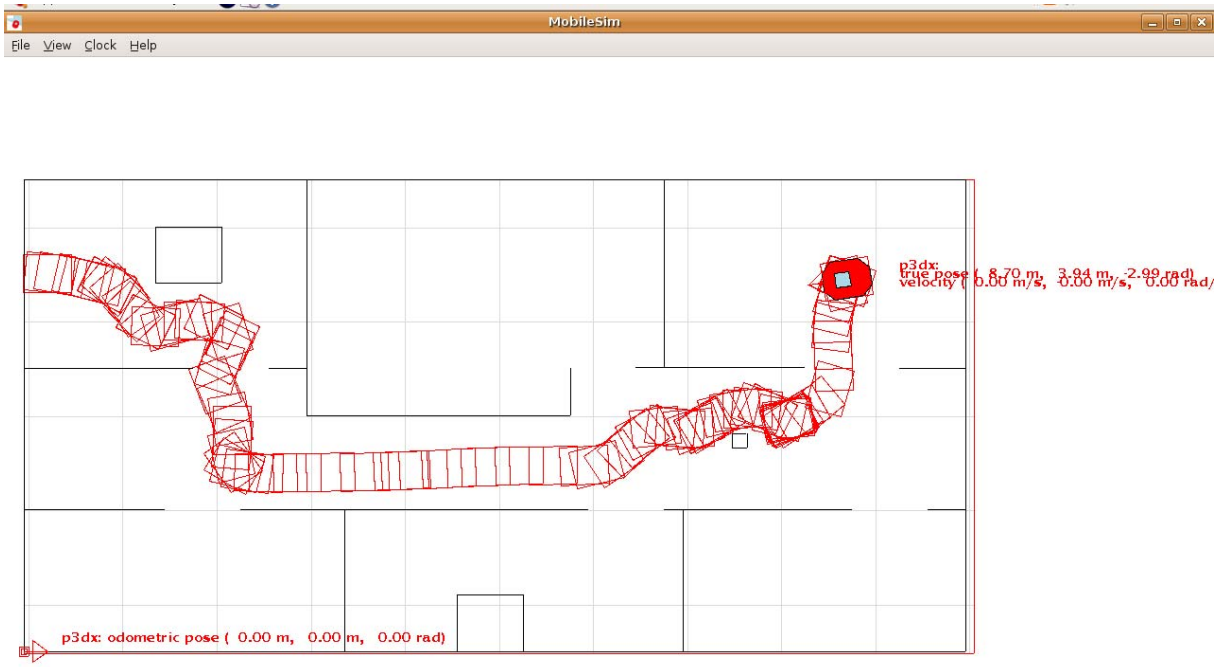

Figura 32. Resposta del robot amb l'aparició d'un petit obstacle.

Com es pot observar el robot en detectar la presència del petit obstacle recondueix la seva posició evitant així el xoc amb aquest. D'aquesta manera podem afirmar que el robot també és capaç de solucionar aquest tipus d'incidències.

6.4. Comparació de resultats actuals amb resultats dels controladors anteriors

Ja per acabar amb totes les simulacions per aquest controlador, hem agafat quatre escenaris model, pels quals anteriorment s'havien realitzat proves (amb el GoTo col·laboratiu) i hem comparat els resultats amb els obtinguts per al controlador dissenyat mitjançant de seguiment de trajectòria.

Per fer-ho hem utilitzat altre vegada el Matlab® i mitjançant una sèrie d'eines hem pogut obtenir un conjunt de dades que expliquen els resultats obtinguts. Les característiques d'aquestes dades les descriurem més endavant.
Hem realitzat les proves per a quatre mapes o quatre situacions diferents. Per a cada mapa s'han realitzat cinc simulacions per tal de poder ampliar la precisió dels resultats que obtenim.

Els mapes pels quals s'han fet aquestes proves són els següents: el mapa GoTo ja que és un mapa sense cap mena d'obstacle simplement amb un punt origen i un punt destí, el segon és el que coneixem com a mapa avoid en el que apareix un obstacle de dos metres que el robot ha d'evitar per arribar a la seva destinació, el tercer és el que anomenarem mapa 7 que és una barreja del mapa anterior i el del GoThrough i per últim el mapa Nova Planta que és un dels realitzats en anterioritat.

6.4.1. Resultats mapa Go to

Per aquest primer mapa les trajectòries descrites pel robot en els cinc casos estan descrites en la següent figura.

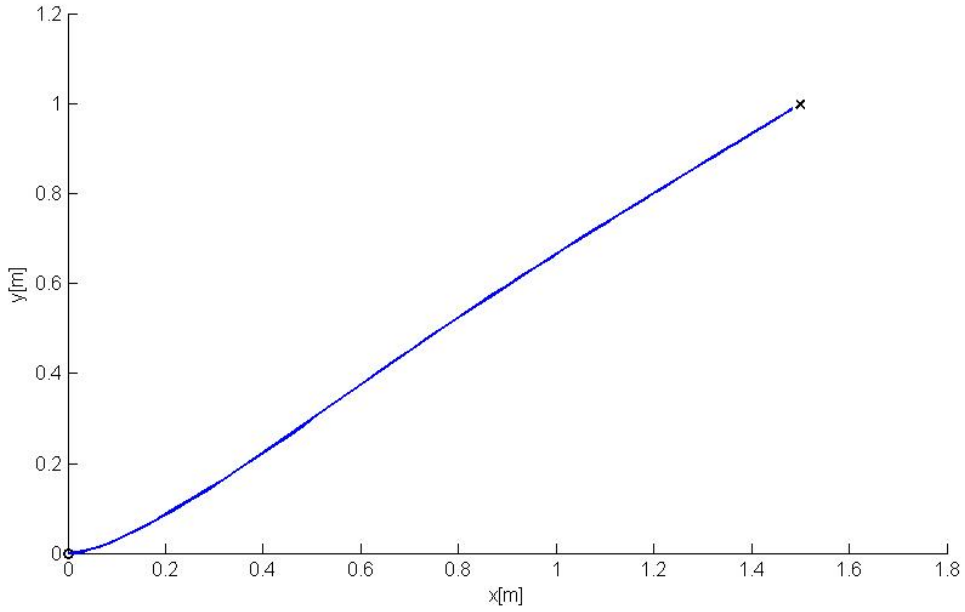

Figura 33. Representació de les trajectòries pel mapa Go to.

En aquesta situació el robot partia d'una posició inicial (0,0) i ha d'arribar a un punt objectiu amb posició (1500,1000), tot això sense l'aparició de cap obstacle.

Com hem comentat a l'inici a més a més de descriure les trajectòries obtingudes també hem pogut obtenir una sèrie de valors informatius que descriuen com s'ha definit aquesta trajectòria recorreguda.
Els valors que descriuran aquestes característiques són els següents:

TD: Distància total recorreguda en metres.
FO: Orientació final en graus.
TT: Temps total en recòrrer la trajectòria en segons.
P: Precisió.
TG: Percentatge de temps que ha estat funcionant l'agent GoTo.
TA: Percentatge de temps que ha estat funcionant l'agent Avoid.
TGt: Percentatge de temps que ha estat funcionant l'agent GoThrough.

Aquests són els valors que descriuran els nostres resultats.

Per aquest primer cas els resultats que hem obtingut són els següents:

	Go to nou		Go to anterior	
	Valor	**Desviació**	**Valor**	**Desviació**
TD [m]	1,78	± 0,01	5,11	± 0,01
FO [º]	17,95	± 9.67	1,95	± 0,71
TT [s]	8,56	± 0,54	16,56	± 0,44
P [%]	98,12	± 0,72	99,41	± 0,22
TG [%]	100	± 0	100	± 0
TA [%]	0	± 0	0	± 0
TGt [%]	0	± 0	0	± 0

Taula 3. Comparació de resultats pel mapa Go to.

Si ara ens parem a observar els resultats que obtenim per aquest mapa podem veure a simple vista que ja hem aconseguit una millora considerable. La principal millora és que hem acurtat el temps que triguem a realitzar la trajectòria de manera considerable i a més a més això no ha comportat una pèrdua excessiva de precisió a l'hora d'arribar a l'objectiu. A la vegada la distància que recorre el robot per arribar a l'objectiu és molt menor que abans, ja que amb el nostre controlador sempre generem trajectòries rectes, per tant les distàncies més curtes. Llavors podem dir que per un mapa sense obstacles el controlador dissenyat millora considerablement en alguns aspectes sense empitjorar en excés en altres.

Per altre banda si mirem les altres dades veiem que l'únic controlador que entra en funcionament evidentment és el GoTo ja que no es troba cap obstacle en cap moment, per tant el 100% de la trajectòria és definida per l'agent GoTo.

6.4.2. Resultats mapa Avoid

Per al següent mapa, les cinc trajectòries descrites en l'aparició de l'obstacle de dos metres, són les que s'observen en la figura següent.

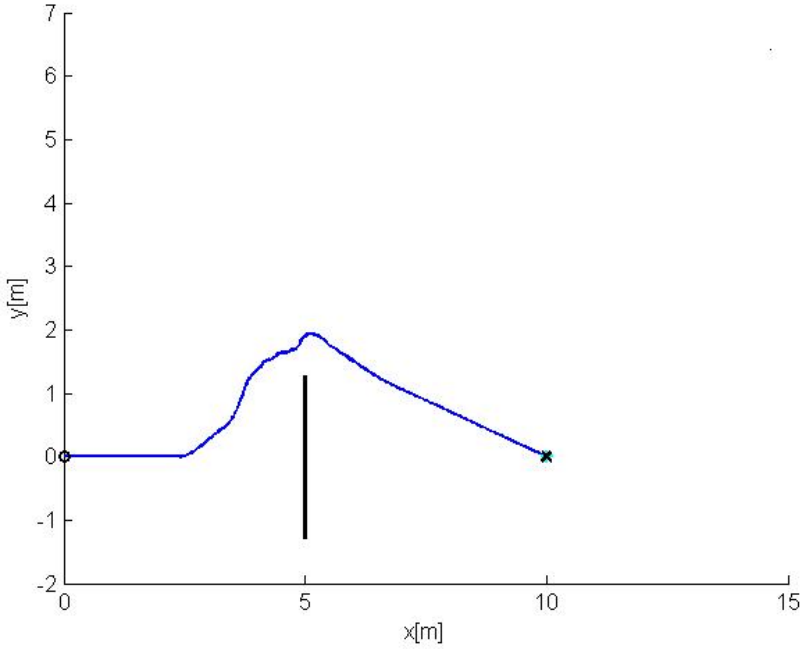

Figura 34. Representació de les trajectòries pel mapa Avoid.

En aquesta situació el robot partirà d'una posició inicial (0,0) i ha d'arribar a un punt objectiu amb posició (10000,0). Aquesta vegada però apareix al mig de la trajectòria un obstacle en posició vertical de dos metres de longitud.

Un cop realitzades les proves corresponents els valors que defineixen el resultat són els que trobem a la següent taula.

	Avoid nou		**Avoid anterior**	
	Valor	**Desviació**	**Valor**	**Desviació**
TD [m]	11,20	± 0,07	12,46	± 0,66
FO [º]	9,56	± 2,04	1,76	± 0,84
TT [s]	40,23	± 3,54	27,76	± 1,22
P [%]	99,61	± 0,14	99,67	± 0,06
TG [%]	84,71	± 2,85	88,40	± 2,29
TA [%]	1,90	± 1,00	11,60	± 2,29
TGt [%]	13,40	± 2,36	0	± 0

Taula 4. Comparació dels resultats pel mapa Avoid.

Si ens centrem en la primera dada ja podem comprovar que la distància que recorre el robot per arribar al destí és lleugerament inferior a la que recorria abans, per tant ja trobem una certa millora. Pel que fa al temps que tarda en recórrer aquesta distància en aquest cas veiem que augmenta. Probablement aquest increment ve produït per aquesta reducció de la distància, ja que en passar el robot més a prop de l'obstacle, reduint així els metres emprats per arribar al punt, entra en actuació l'agent Avoid i l'agent GoThrough per evitar aquest element i això rellenteix la marxa del robot.

Pel que fa a la precisió els resultats pràcticament no es veuen alterats. Si mirem però ara els percentatges de funcionament dels agents veiem clarament que el GoTo segueix funcionant pràcticament el mateix temps que abans però a diferència d'abans, l'agent GoThrough té el control del robot durant un cert temps.

6.4.3. Resultats Mapa 7

Per al següent mapa veiem l'aparició d'un obstacle amb profunditat i que a la vegada es troba situat entre dos parets o dos obstacles. La resposta per a les cinc simulacions necessàries ha estat la següent.

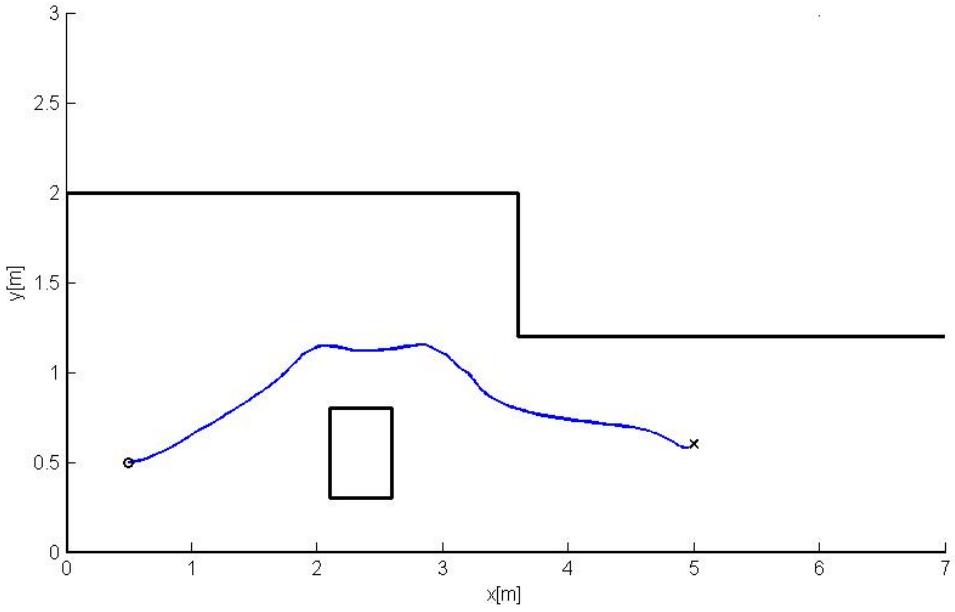

Figura 35. Representació de les trajectòries pel Mapa 7.

En aquesta situació el robot partirà d'una posició inicial (500,500) i ha d'arribar a un punt objectiu amb posició (5000,600). Aquesta vegada però apareix al mig de la trajectòria un obstacle quadrat amb unes mides aproximades de 500x500 mm que fa rectificar la trajectòria del robot cap amunt. Un cop rectifica la trajectòria, per arribar al punt objectiu, a

més a més haurà de passar pel mig de dos obstacles amb la qual cosa podem preveure l'actuació de l'agent GoThrough tal i com veurem en els següents resultats.

Un cop realitzades les proves corresponents els valors que defineixen el resultat són els que trobem a la següent taula.

	Mapa 7 nou		**Mapa 7 anterior**	
	Valors	**Desviació**	**Valors**	**Desviació**
TD [m]	4,77	± 0,03	5,08	± 0,30
FO [º]	2,78	± 2,38	1,97	± 1,01
TT [s]	20,96	± 2,15	27,90	± 1,35
P [%]	98,35	± 0,52	99,55	± 0,16
TG [%]	26,26	± 4,41	58,68	± 4,27
TA [%]	28,44	± 3,80	41,32	± 4,27
TGt [%]	45,30	± 6.33	0	± 0

Taula 5. Comparació dels resultats pel Mapa 7.

Si ens fixem en la distància recorreguda altre vegada aconseguim millorar el resultat ja que com hem comentat abans generem trajectòries rectes i això produeix sempre que el robot recorri el camí més curt. A la vegada també hem aconseguit que el robot sigui molt més ràpid alhora de realitzar el trajecte, redueix en més de un trenta per cent el temps que tardava abans. Com en les altres situacions on veiem afectats els resultats una mica negativament és en la precisió que es veu reduïda en gairebé un dos per cent.

Finalment si ens fixem en els percentatges de funcionament dels agents veiem que en el nostre cas entra en funcionament i durant la major part del recorregut l'agent GoThrough que en canvi en el cas anterior no entrava en funcionament en cap moment. Pel que fa al GoTo queda pràcticament reduïda a la meitat la seva participació mentre que l'Avoid també redueix la seva aportació però en menor mesura.

6.4.4. Resultats pel mapa Planta 4

Finalment hem realitzat el mateix procediment que amb els altres tres mapes però aquesta vegada amb el mapa Planta 4 que és el que inclou diferents habitacions amb un passadís central per arribar a totes les habitacions.

Els resultats de les cinc simulacions realitzades per aquest mapa són les que mostrem a la següent figura.

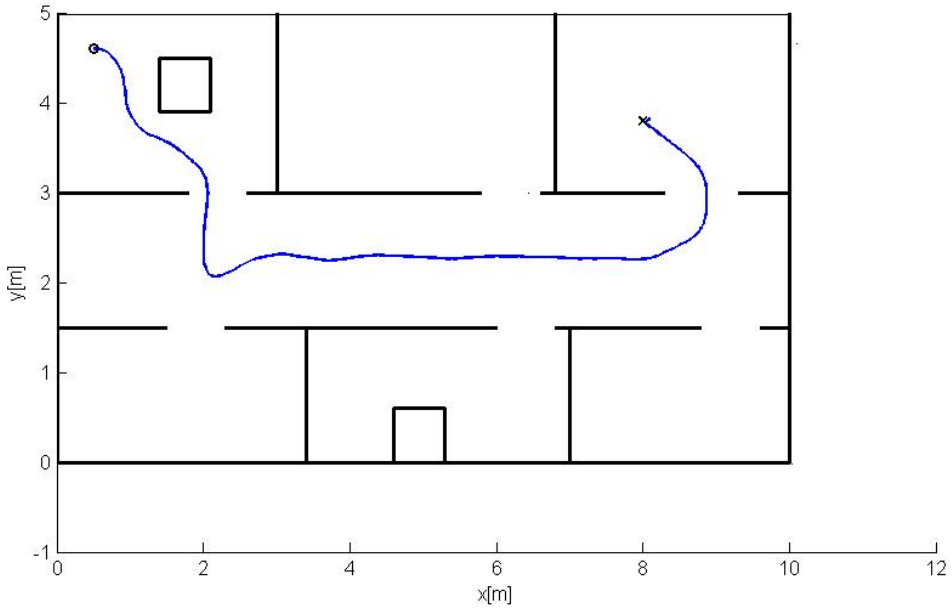

Figura 36. Representació de les trajectòries pel mapa Planta 4.

En aquesta situació el robot partirà d'una posició inicial (500,4600) i ha d'arribar a un punt objectiu amb posició (8000,3800). Aquesta vegada a més a més d'aparèixer un obstacle només iniciar-se el moviment, aquest es troba dins d'una habitació i haurà de sortir-ne.
Posteriorment haurà de travessar un passadís on segurament entrarà en acció l'agent GoThrough i finalment el robot girarà per entrar en una altra habitació on es troba el punt objectiu.

Un cop realitzades les proves corresponents els valors que defineixen el resultat són els que trobem a la següent taula.

	Mapa Planta 4 nou		Mapa Planta 4 anterior	
	Valors	**Desviació**	**Valors**	**Desviació**
TD [m]	12,05	± 0,20	12,55	± 0,44
FO [º]	12,15	± 2,59	1,77	± 1,63
TT [s]	54,24	± 3,23	66,06	± 8,79
P [%]	99,62	± 0,10	99,78	± 0,13
TG [%]	47,79	± 1,61	66,79	± 2,31
TA [%]	28,62	± 3,55	33,21	± 2,31
TGt [%]	23,49	± 3,05	0	± 0

Taula 6. Comparació dels resultats pel mapa Nova Planta.

Novament veiem que la distància per arribar al punt es veu reduïda com en gairebé la majoria de casos que hem tractat fins al moment.

Pel que fa al temps emprat per arribar al punt també es redueix ostensiblement, el retallem en gairebé 12 segons per tant podem considerar que és un resultat molt positiu pel que fa en aquest aspecte.

Si mirem la precisió veiem que es veu reduïda en un gairebé 0,5% però podem considerar que és un resultat força acceptable veient els aspectes en els que sí aconseguim una millora.

Finalment fixant-nos en els períodes de funcionament dels diferents agents, veiem que el temps que abans s'emprava en el GoTo es redueix en certa mesura i aquest temps passa a ser controlat per l'agent GoThrough que en el cas anterior no s'utilitzava. En canvi l'agent Avoid pràcticament entra en funcionament en la mateixa mesura que abans.

6.4.5. Conclusió final dels resultats de les proves

Fixant-nos en els resultats obtinguts a trets generals en les diferents proves podem veure que generalment amb el controlador dissenyat aconseguim normalment reduir el temps que tardem en recórrer la distància i a la vegada també retallem aquesta mateixa.

També veiem que en el nostre cas ja utilitzem l'agent GoThrough i això suposa també una ajuda a l'hora improvar aquests resultats.

6.4.6. Resultats en el món real

Per a cadascun dels dos controladors dissenyats s'han realitzat les corresponents proves per tal d'assegurar-nos que el funcionament d'aquest era correcte, no només per a la simulació sinó també per al món real.

Per tant doncs, hem dividit les proves en dos parts, la primera per al controlador dissenyat mitjançant el mètode de la persecució pura i la segona per al controlador de disseny empíric.

Si ens centrem en el primer cas hem realitzat una prova en un dels passadissos de l'edifici PIV de la Universitat de Girona i en un mateix experiment hem pogut comprovar el funcionament en diferents condicions. Ha intervingut el GoTo en algun moment, en altres l'Avoid per tal d'esquivar un obstacle i en altres el GoThrough pel fet que en una part del recorregut es mou per un passadís.

La resposta que hem obtingut és la que podem observar a la següent figura.

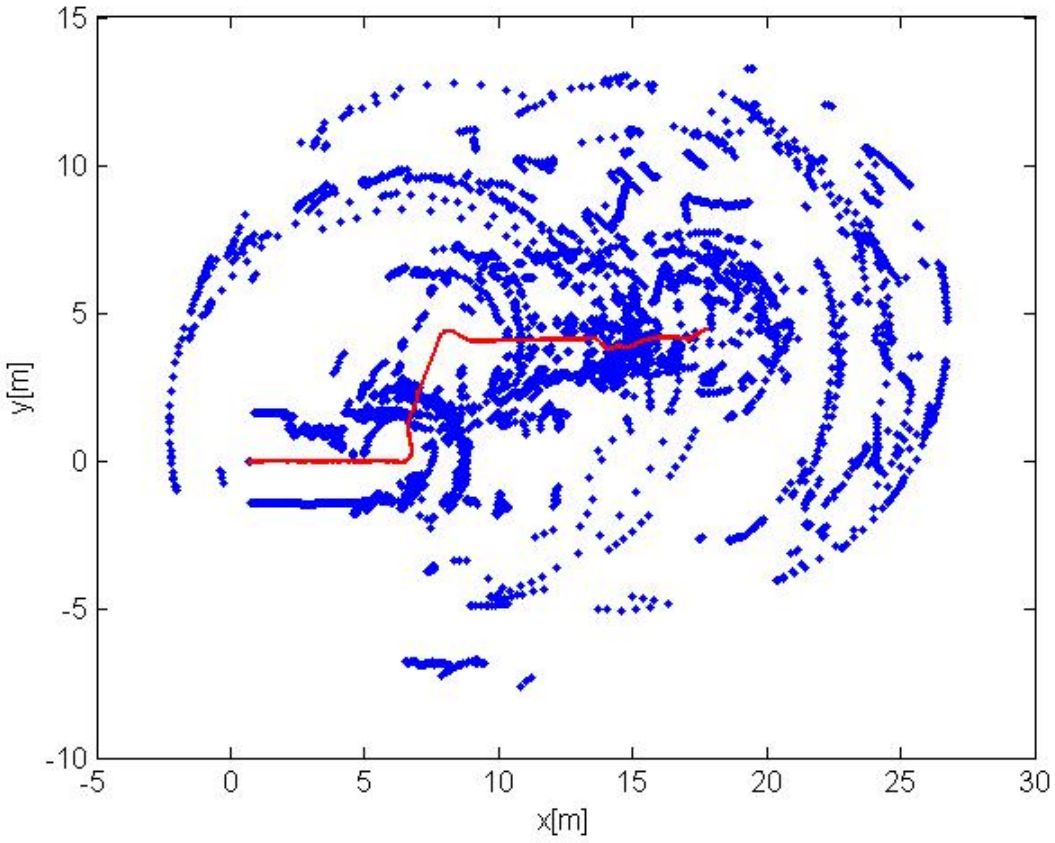

Figura 37. Resposta del robot en el passadís de l'edifici PIV de la UdG amb sonars.

En aquesta figura observem la interacció dels sonars del robot amb els obstacles que es va trobant en el camí, tant pel que fa a les parets que formen el passadís com algun que altre obstace que s'hagi pogut trobar pel camí. A l'inici veiem que el robot es mpu per un passadís fins que gira a l'esquerra per tal d'afrontar la recta final del seu recorregut fins arribar a la seva destinació. Prèviament ha anat evitant alguns obstacles que s'ha trobat pel camí com són: persones que han aparegut espontàneament, extintor, paperera,etc. A continuació mostrem el recorregut sense els senyas del sonar.

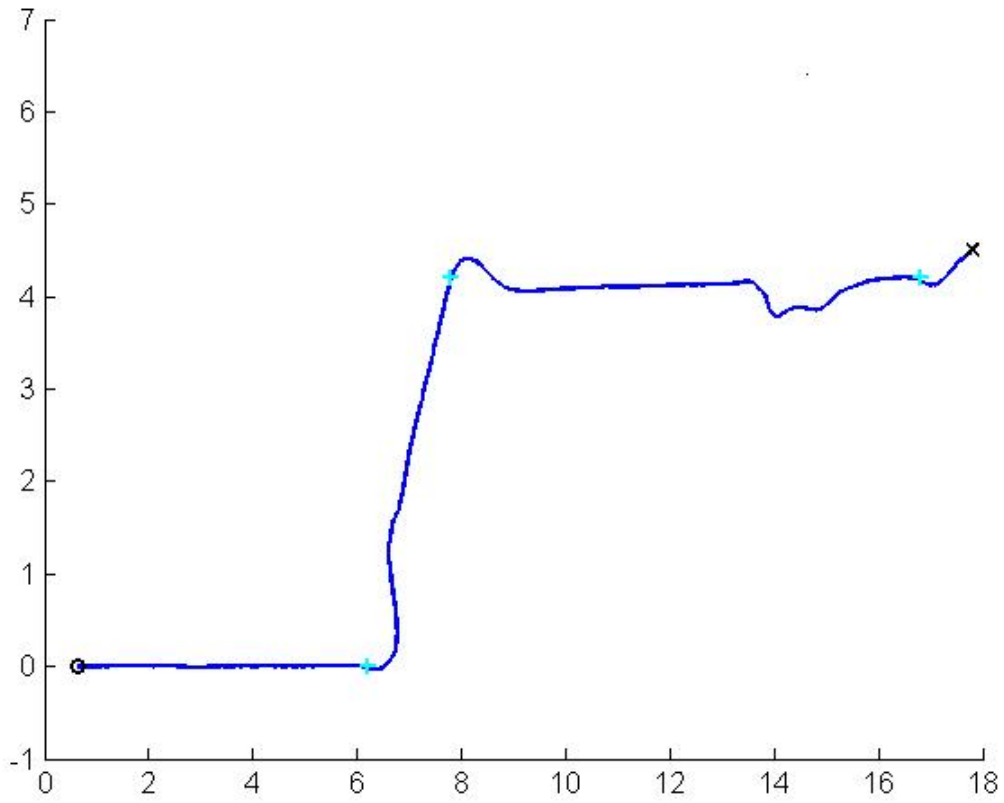

Figura 38. Resposta del robot en el passadís de l'edifici PIV de la UdG amb sonars.

La resposta també la podem definir amb termes numèrics i d'aquesta manera hem obtingut els següents resultats on:

TD: Distància total recorreguda en metres.

FO: Orientació final en graus.

TT: Temps total en recòrrer la trajectòria en segons.

P: Precisió.

TG: Percentatge de temps que ha estat funcionant l'agent GoTo.

TA: Percentatge de temps que ha estat funcionant l'agent Avoid.

TGt: Percentatge de temps que ha estat funcionant l'agent GoThrough.

	Valor
TD [m]	21
FO [º]	2,90
TT [s]	72,90
P [%]	99,94
TG [%]	84
TA [%]	9,45
TGt [%]	5,34

Taula7. Valors obtinguts per dins.

Si passem ara a la resposta en el món real pel segon controlador, amb un PID de disseny empíric, s'han realizat diverses proves per a superfícies exteriors amb característiques de fricció alta i amb rugositat més gran.

En aquest cas s'han seleccionat dues proves. La primera d'elles el robot es dirigeix fins a una distància de 5 metres en component x i 5 metres amb component y sense trobar-se cap obstacle en el seu camí.
La segona prova realitzada el robot realitza a mateixa distància i el mateix recorregut que en el cas anterior però en aquest cas es troba un obstacle al mig del recorregut i es pot observar com el detecta i com rectifica la seva trajectòria per tal d'evitar-lo.

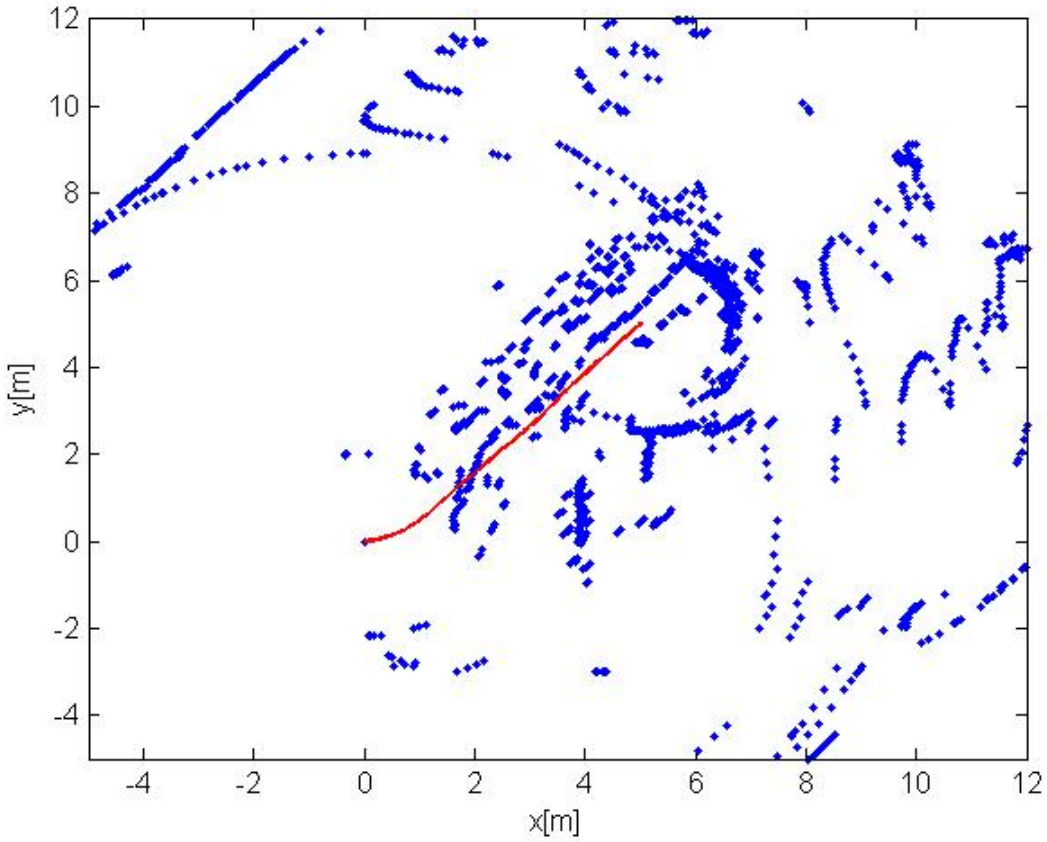

Figura 39. Resposta del robot en una superfície rugosa sense obstacle.

La resposta per a l'exterior sense cap obsacle és la que tenim a la figura anterior i els obstacles que veiem que detecta el robot són la persona que realitza la prova que camina al costat del robot i a la part dreta que hi trobem un petit muntatge arquitectònic que provoca aquestes formes en el sonar.

La resposta sense els sonars és la següent.

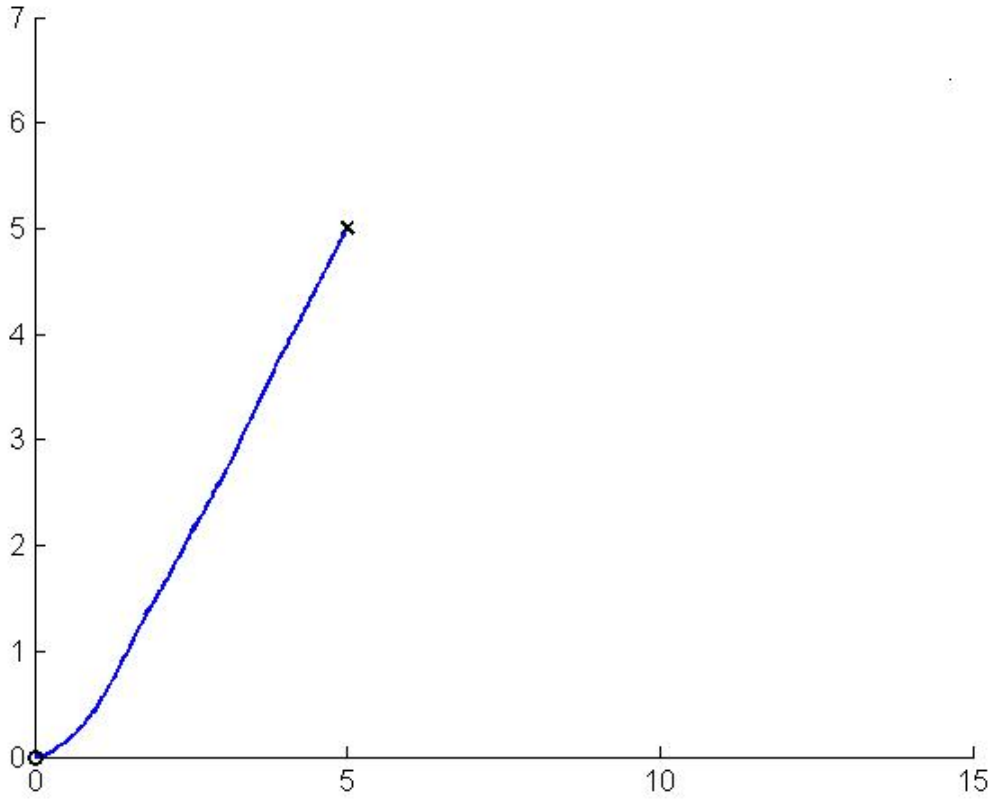

Figura 40. Resposta sense els sonars en l'exterior i sense obstacle.

Com en el cas anterior hem realitzat una definició amb valors de les condicions de la resposta, i són les següents:

La resposta per una superfície rugosa amb la intervenció d'un obstacle en el camí és la següent.

	Valor
TD [m]	7,22
FO [º]	13,97
TT [s]	31,7
P [%]	99,50
TG [%]	96,85
TA [%]	3,14
TGt [%]	0

Taula 8. Valors obtinguts per fora sense obstacle.

Teòricament en aquest apartat no hauria d'existir la intervenció del Avoid però intervé mínimament pel fet que caminem al costat del robot.

Si realitzem el mateix que abans però amb un obstacle a mig camí, la resposta amb sonars és la següent.

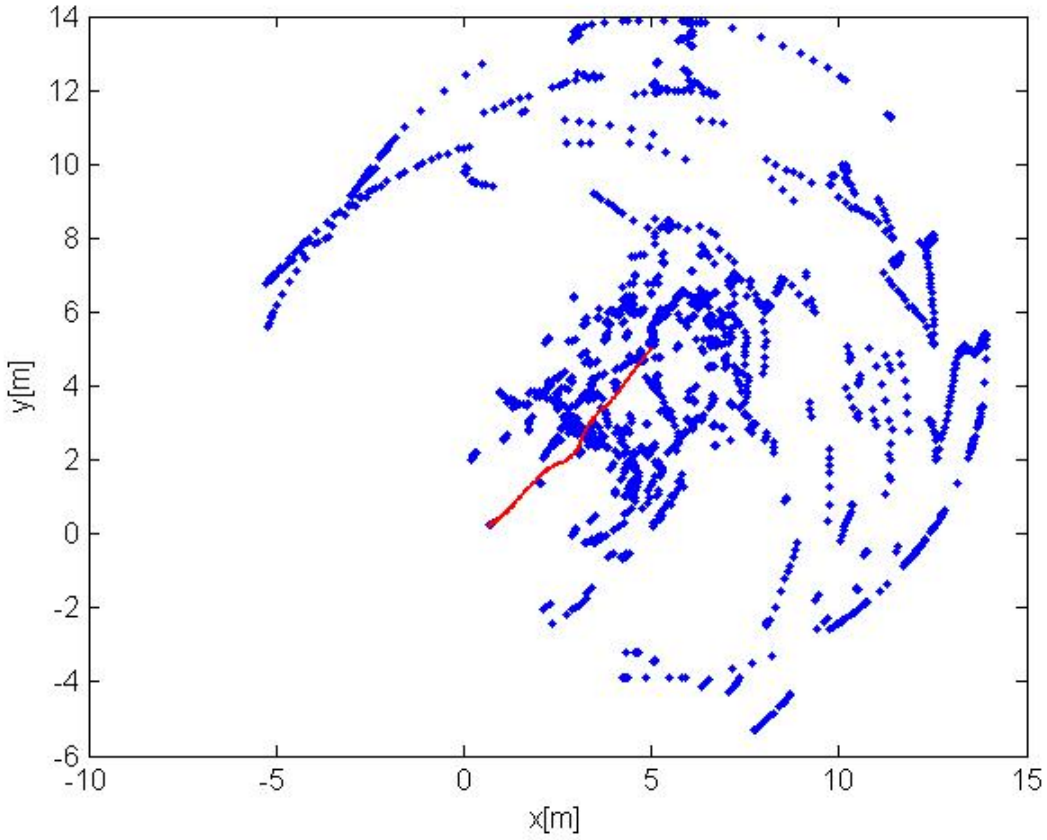

Figura 41. Resposta del robot en una superfície rugosa amb obstacle.

En aquest cas podem veure com el robot detecta a través dels sonars del robot l'obstacle i es recondueix per tal de no impactar amb aquest.
La resposta sense la intervención dels sonars és la següent.

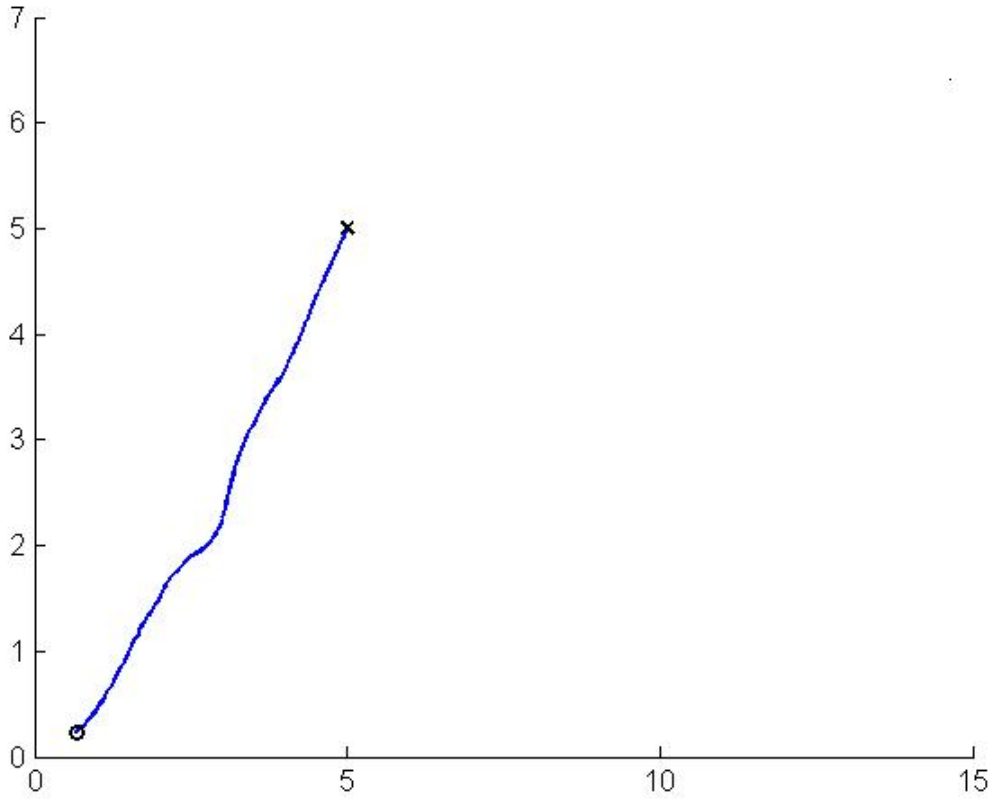

Figura 42. Resposta amb obstacle per a l'exterior sense els sonars.

A mig camí es veu com el robot evita l'obstacle i segueix la seva trajectòria normalment.

Els valors que defineixen la resposta són els següents:

	Valor
TD [m]	6,56
FO [º]	0,09
TT [s]	28,30
P [%]	99,94
TG [%]	91,20
TA [%]	7,39
TGt [%]	1,41

Taula 9. Valors obtinguts per fora amb obstacle

7. Resum del pressupost

La finalitat d'aquest projecte ha estat la de dissenyar nous agents GoTo que siguin òptims per a la estructura multi-agent que està instaurada en el robot Grill (Pioneer 2DX) del laboratori d'eXiT de la Universitat de Girona, on l'import total ha estat d'onze mil euros sense IVA.

8. Conclusions

El projecte ha complert tots els objectius proposats en tots els seus termes.
Amb l'exposat en la present memòria i en els seus Annexos adjunts, així com la resta dels documents que integren el projecte, queden suficientment definits tots i cadascun dels elements que són precisos per a la utilització per futures investigacions.

Totes les conclusions particulars de cada fase d'aquest projecte queden recollides en els seus apartats corresponents dintre de la memòria descriptiva.

S'espera que aquest projecte serveixi de base pels futurs projectes en aquest àmbit. Com a possibles projectes futurs es podria considerar la possibilitat de contemplar l'opció de instaurar elements en el robot que li permetin saber quan ha d'entrar en funcionament el segon controlador, per tal de detectar quan el robot es desplaça per superfícies rugoses. D'aquesta manera no seria necessària l'intervenció de la persona per tal de definir-li en quina superfície es mourà i per tant el robot ho podria decidir automàticament.

Gerard Alarcón Rodríguez
Enginyer Tècnic Industrial especialitzat en electrònica industrial

Girona, 8 de desembre de 2010

9. Relació documents

El projecte consta d'una sèrie d'apartats on es descriu tot el treball realitzat. El formen un total de quatre documents: la memòria, el plec de condicions, l'estat d'amidaments i el pressupost.

10. Bibliografia

CPLUSPLUS.COM. (http://www.cplusplus.com/doc/tutorial/, 10 d'octubre de 2010).

GONZALEZ E. Implementació d'un agent per l'arquitectura del robot Grill utilitzant raonament difús. Projecte/ Treball Fi de Carrera. Enginyeria Industrial. Escola Politècnica Superior.Universitat de Girona. 2009.

INNOCENTI, BIANCA. A Multi-agent Architecture with Distribuited Coordination for an Autonomous Robot. PhD Tesis, Universitat de Girona, 2008.

OLLERO BATURONE, A.Robótica:Manipuladores y robots móviles.España: Marcombo,S.A., cop 1ª Edición. 2001.

A. Codi en C++

Abans d'introduir el codi per als dos agents dissenyats cal dir que tant un disseny com l'altre han estat afegits en format .cpp en una carpeta dins la carpeta Memòria de l'arrel del CD per tal de poder ser consultats i utilitzats en qualsevol moment.

A.1 Codi pel mètode de la persecució pura

```
#include "gotoagent.h"

GotoAgent::GotoAgent(Q_UINT16 server_port, QString hostname,Q_UINT16 df_port,
QString df_hostname)

 : BasicAgent(server_port,hostname)
{
connect(this,SIGNAL(SigRegister()),SLOT(Register()));
connect(this,SIGNAL(SigPosition()),SLOT(Position()));
connect(this,SIGNAL(SigDPosition()),SLOT(DesiredPosition()));
connect(this,SIGNAL(SigCloseAgent()),SLOT(CloseAgent()));
connect(this,SIGNAL(SigUtility()),SLOT(Utility()));

//estableix la informacio propia

selfinfo.SetMutex(false);
selfinfo.addSuppService(QString("DESIRED_LINEAR_SPEED"));
selfinfo.addSuppService(QString("DESIRED_ANGULAR_SPEED"));
selfinfo.addSuppService(QString("UTILITY"));
selfinfo.addReqService(QString("DESIRED_COORDINATES"));
selfinfo.addReqService(QString("DESIRED_HEADING"));
selfinfo.addReqService(QString("ABSOLUTE_COORDINATES"));
selfinfo.addReqService(QString("ABSOLUTE_HEADING"));
selfinfo.addResource(QString("ROBOTAGENT"));

df=CreateClient(df_hostname,df_port);

// Es registra al DF

RegisterAgent(df,AGENTNAME,selfinfo.getAllSuppService(),selfinfo.
getAllReqService(),selfinfo. getAllResources(),selfinfo.GetMutex());

       c1.IniciaErrors();
       c1.LimitsVel(400.0,80.0);
       c1.ValorsConstants(0.5,0.2,0.0,0.1,1);//(1.5,0.8,0.0,0.1,1);//PID pos kp=5 i
       ki=3
```

```
	c1.ValorsConstants(1.0,0.5,0.0,0.1,0);//(1.0,1.2,0.0,0.1,0)//;//PID	head
	kp=2, ki=1

	canvi=false;
	control=1;
	smooth=true; // si false abrupt
	fp=fopen("goto.txt","a+");
	fu=fopen("posicions.txt","a+");

	pos_ini.setX(0.0);
	pos_ini.setY(0.0);
	pos_ini.setTh(0.0);
	vel_ang_ant=0.0;
	curv_ant=0.0;
	prinv=true;
	DX_ant=0;
}

GotoAgent::~GotoAgent()
{
	fclose(fp);
	fclose(fu);
	CloseClientConnection(df);

	QValueList<MInfo>::iterator itai;

	for(itai=agentinfo_list.begin();itai!=agentinfo_list.end();itai++) //per cada
	agent connectat
	{
		CloseClientConnection((*itai).getTCPClient());
	}
}

void GotoAgent::CloseAgent()
{
//	printf("emit closed\n");
	emit Closed();
}
/*void GotoAgent::CanviControl()
{
	canvi=true;
	vela.setX(utility.getX()); //la utilitat
	vela.setY(utility.getY()); //la velocitat lineal
	vela.setTh(utility.getTh()); //la velocitat angular
	ctf=0.0;
}*/
```

```
void GotoAgent::Register()
{
       ConnectToServer(selfinfo);
       //printf("\n CONNECTAT\n");
}

void GotoAgent::DesiredPosition()
{
//     flag=0;
       prinv=true;
}

void GotoAgent::Utility()
{
       UpdateUtility();
       //printf("Utility()\n");
}

void GotoAgent::Position()
{
    /* QStringList::iterator itlist;
       QValueList<MInfo>::iterator itai;
       QStringList mes;

       for(itai=agentinfo_list.begin();itai!=agentinfo_list.end();itai++) //per cada
       agent connectat
       {
              mes=(*itai).getAllMessages();
              //busquem al llistat d'utilitats la posicio de l'agent que envia la
              utilitat

              for(itlist=mes.begin();itlist!=mes.end();itlist++)
              {
                     fprintf(fp,(*itlist));
                     fprintf(fp,(*itai).getAgentName());
                     fprintf(fp,"\n");
              }
       }
    */ run();
}
void GotoAgent::run()
{
//     double  pGoto=0.0  ;  //variables  que  contendran  la  prioridad  del  Goto  y
       delAvoid
//     double Ts=0.1;// temps de mostreig del robot.
```

```
	//-------------------------------------------------------------------------

	//                        CALCULAR VELOCIDADES

	//-------------------------------------------------------------------------

//	double velocitat, velocitatR;
	double gir,distancia,edist,egir; // consigna de giro y distancia
	double dx,dy,mu_aprop;

	MSpeeds velc1;
	MUtil self_util;
	double L_max;
	double s_m=0.5;

	//distancia que queda para llegar al punto deseado

	dx=setpoint.getX()-pos.getX();
	dy=setpoint.getY()-pos.getY();
	distancia = sqrt(pow(dx,2)+pow(dy,2));
	//fp=fopen("goto.txt","a+");

	edist=distancia;
	// calcula l'angle de gir
	gir=atan2(dy,dx)*180.0/3.141596;

	if(distancia<100.0)
	{
		gir =setpoint.getTh();
	}
	//	printf("ed=%f,gir=%f\n",edist,gir);
		egir=gir-pos.getTh();

	if(egir>180)
		egir=egir-360;
	else if(egir<-180)
		egir=360+egir;

	if(fabs(egir)>90)
		edist=0.0;

	velc1=c1.AccioControl(distancia,egir);
	printf("distancia=%f, vel_lin=%f\t",distancia,velc1.getVl());
	//		Lmax   s
	edist=sqrt(pow((pos.getX()-pos_ini.getX()),2)+pow((pos.getY()-
	pos_ini.getY()),2));
```

```
L_max=Vary.calcula_minmax(edist, 1000,10000.0,50.0,100.0);

velc1=geometric(4.0, s_m,velc1.getVl()); //vel es una variable que te dos
camps, Vl (velocitat lineal) i Va(velocitat angular).
printf("vel_ang=%f\n",velc1.getVa());

if(distancia<50.0)
{
//Si no ponemos esto en algunos casos cuando ya ha llegado al punto correcto
//Se desvia pq se sigue moviendo aunque sea poco.
	velc1.setVl(0.0);

	if(fabs(setpoint.getTh()-pos.getTh())<10)
	{
		velc1.setVa(0.0);
	}
}

if(distancia<100.0)
{
	//Si no ponemos esto en algunos casos cuando ya ha llegado al punto
	correcto
	//Se desvïa pq se sigue moviendo aunque sea poco.
	velc1.setVl(0.0);

	if(fabs(setpoint.getTh()-pos.getTh())<10)
	{
		velc1.setVa(0.0);
	}
	else
		velc1=c1.AccioControl(distancia,egir);
		velc1.setVl(0.0);
}

if(isnan(velc1.getVl())!=0)
	velc1.setVl(0.0);

//-------------------------------------------------------------------------

//                      UTILITY CALCULATION

//-------------------------------------------------------------------------

//self_util.setUtil(CalculateUtility(distancia));   //utilitat
self_util.setUtil(0.6);   //utilitat
self_util.setControl(control);
```

```
self_util.setVelAng(velc1.getVa());
self_util.setVelLin(velc1.getVl());

//printf("Utility=%f\n",self_util.getUtil());

//-------------------------------------------------------------------------

//                   RESOURCE EXCHANGE

//-------------------------------------------------------------------------

//si agent te control
//self_util.setVelAng(0.0);
//self_util.setVelLin(0.0);
//printf("control=%d \t", control);

if(control==1)
{
      SendUtility(self_util,QString("UTIL"));
}

bool guanyo=true,ndata=false;
HigherUtility(guanyo,ndata,self_util);

//si tenim el control i hem guanyat enviem les dades al robot i a la resta
d'agents

printf("control=%d,guanyo=%d,newdata=%d\n",control,guanyo,ndata);

if(ndata)
{
      if( (control==1) && guanyo)
      {
            if(canvi)
                  self_util=ResourceExchange(self_util,quiControla);
                  //busquem al llistat      printf("Soc     a     dins\n");
                  d'utilitats la posicio de l'agent que envia la utilitat

            SendSpeeds(self_util);
            printf("control&guanyo:
            vel.li=%f,vel.an=%f\n",vel.getVl(),vel.getVa());

      }
else
      if((control==0) && guanyo)//si no tinc el control i guanyo
      {
```

```
                    canvi=true;
                    ctf=0.0;
                    control = 1;
                    self_util=ResourceExchange(self_util,quiControla);
                    SendSpeeds(self_util);
                    printf("guanyo vel.li=%f,vel.an=%f\n",vel.getVl(),vel.getVa());

                    SendUtility(self_util,QString("UTIL"));

             }
                    else  //si no he guanyat perdo el control
                           if (!guanyo)
                                  control =0;
      }
      else
      {
             if (control==1)
             {
                    if(canvi)
                           self_util=ResourceExchange(self_util,quiControla);
                           SendSpeeds(self_util);
             //printf("ndatafalse: vel.li=%f,vel.an=%f\n",vel.getVl(),vel.getVa());
             }
      }

      SendUtility(self_util,QString("UTILITY"));

      //printf("vel=%f, ang=%f, util=%f\n",velocitat,velocitatR,pGoto);
      //fprintf(fp,"velL=%f, velR=%f, ctf=%f, tf=%f, can   printf           ("Soc a
      dins\n");vi=%d,        control=%d,guanyo=%d\n",        self_util.getVelLin(),
      self_util.getVelAng(), ctf-1, tf, canvi, control,guanyo);
}
// poserem directament geometric perque aqui no hi ha confusio

MSpeeds GotoAgent::geometric(double Lmax, double s,double vel_lin) // totes en mm

{
      MSpeeds veld;
      int max_val=50;
      int i;
      float x_vehiculo,y_vehiculo, x_ob, y_ob; //tenir en compte que estan en mm
      float phi_vehiculo; // en graus!
      double dist[100];
      double acum[100];
      int j, k; //j per posar l'index del valor mÃ©s petit
      double val; // per posar el valor mes petit
```

```
	double incremento_x,incremento_y, L,curvatura;
	double pi=3.141596;
	double vel_ang;
	double signe;
	double Kpp, der_Dx;
	double dx,dy;
	QString string,cad;
	double m,n;

	if (prinv==TRUE)    //primera vegada que fa la funcio
	{
		pos_ini.setX(pos.getX());
		pos_ini.setY(pos.getY());
		pos_ini.setTh(pos.getTh());
		CalculaTrajectoria(max_val);

		/*s=4;
		max_val=11;
		x_camino[0]=0;
		y_camino[0]=0;
		for (i=1;i<11;i++)
		{
			x_camino[i]=x_camino[i-1]+1;
			y_camino[i]=0;
			string=QString("x_camino[%1]=%2 "). arg(i). arg(x_camino[i]);
			fprintf(fp, "%s\n",string.latin1());
		}*/

		prinv=FALSE;
		K_ANT=1;

for(i=0;i<max_val;i++)
	{
		acum[i]=0;
		/*string=QString("acum[%1]= %2").arg(i).arg(acum[i]);
		fprintf(fp, "%s\n",string.latin1());*/

		for(k=i;k<max_val-1;k++)
		{
		val=sqrt(pow((x_camino[k]-x_camino[k+1])/1000.0,2) + pow((y_camino[k]-
		y_camino[k+1])/1000.0,2));
		acum[k+1]=acum[k]+val;

		/*string=QString("acum[%1]=%2,val=%3").arg(k+1).arg(acum[k+1]).arg(val
		);
			fprintf(fp, "%s\n",string.latin1());
```

```
        */
        }
        for(k=i;k<max_val;k++)
        {
                dist[k]=fabs(acum[k]-s);
                /*string=QString("dist[%1]= %2").arg(k).arg(dist[k]);
                fprintf(fp, "%s\n",string.latin1());
        */
        }
        val=dist[i];
        j=i;
        //nomes interessa j

        for(k=i; k<max_val;k++)

        {
               if(dist[k]<val) // si el valor en dist es mes petit que val

                      {
                             val=dist[k]; //poso a val el valor mes petit
                             j=k;       // poso en j l'index del valor mes petit
                      }

        }// a la sortida d'aquest for tinc el valor minim en val i l'index a j

               INDEX[i]=j;

        /*     string=QString("Index[%1]= %2").arg(i).arg(INDEX[i]);
               fprintf(fp, "%s\n",string.latin1());
        */
        }
}
else
{
        CalculaTrajectoria(max_val);
}

x_vehiculo=pos.getX();
y_vehiculo=pos.getY();
phi_vehiculo=pos.getTh();

if(phi_vehiculo>180)
        phi_vehiculo=phi_vehiculo-360;

else if(phi_vehiculo<-180)
        phi_vehiculo=360+phi_vehiculo;
```

```
	for(i=0;i<max_val;i++)
	{
		m=cos(phi_vehiculo*pi/180.0);
		n=sin(phi_vehiculo*pi/180.0);
		dist[i]=sqrt(pow((x_camino[i]-x_vehiculo),2)+pow((y_camino[i]-
		y_vehiculo),2))/1000.0;

		/*string=QString("dist[%1]= %2").arg(i).arg(dist[i]);
		fprintf(fp, "%s\n",string.latin1());
		string=QString("x_vehiculo=%1
		y_vehiculo=%2phi_vehiculo=%3").arg(x_vehiculo).arg(y_vehiculo).arg(phi
		_vehiculo);

		fprintf(fu, "%s\n",string.latin1());
	*/
	}
	j=K_ANT;
	val=dist[K_ANT];

	for(i=K_ANT; i<max_val;i++)
	{
	if(dist[i]<val) // si el valor en dist es mes petit que val
		{
		val=dist[i]; //poso a val el valor mes petit
		j=i;		// poso en j l'index del valor mes petit
		}
	}// a la sortida d'aquest for tinc el valor minim en val i l'index a j
	string=QString("K_ANT= %1").arg(j);
	fprintf(fp, "%s\n",string.latin1());

K_ANT=j;
x_ob=x_camino[INDEX[K_ANT]];
y_ob=y_camino[INDEX[K_ANT]];
/*m=cos(phi_vehiculo*pi/180.0);
n=sin(phi_vehiculo*pi/180.0);*/

string=QString("punts x_ob=%1 y_ob=%2 x_vehiculo= %3 y_vehiculo= %4 phi_vehiculo=
%5 cos=%6 sin=%7"). arg(x_ob). arg(y_ob).
arg(x_vehiculo).arg(y_vehiculo).arg(phi_vehiculo).arg(m).arg(n);
fprintf(fp, "%s\n",string.latin1());

//incremento_x=((((x_ob-x_vehiculo)/1000.0)*cos(phi_vehiculo*pi/180.0))+(((y_ob-
y_vehiculo)/1000.0)*sin(phi_vehiculo*(pi/180.0)))); //comprovar que l'equacio sigui
la correcta
```

```
incremento_y=((y_ob-y_vehiculo)/1000.0*cos(phi_vehiculo*pi/180.0))-((x_ob-
x_vehiculo)/1000.0*sin(phi_vehiculo*pi/180.0));

/*incremento_x=((x_ob-x_vehiculo)/1000.0*sin(phi_vehiculo*pi/180.0))+((y_ob-
y_vehiculo)/1000.0*cos(phi_vehiculo*pi/180.0));*/

       string=QString("incremento_y= %1").arg(incremento_y);
       fprintf(fp, "%s\n",string.latin1());

L=sqrt(pow(((x_ob-x_vehiculo)/1000.0),2)+pow(((y_ob-y_vehiculo)/1000.0),2));
//printf("x_ob=%f,     y_ob=%f,     x_vehiculo=%f,     y_vehiculo=%f,phi_vehiculo=%f,
L=%f,incremento_x=%f\n",x_ob,y_ob,       x_vehiculo,       y_vehiculo,phi_vehiculo,L,
incremento_x);

       string=QString("L= %1").arg(L);
       fprintf(fp, "%s\n",string.latin1());

       if(L>Lmax)
              L=Lmax;

       else if(L<=0)
              L=1;

curvatura=((1)*incremento_y*(2.0/(pow(L,2))));
der_Dx=(incremento_x-DX_ant)/0.1;

string=QString("geometric L=%1 curv=%2 inc_x= %3 y_ob=%4 y_vehiculo=%5"). arg(L).
arg(curvatura).arg(incremento_x).arg(y_ob).arg(y_vehiculo);
fprintf(fp, "%s\n",string.latin1());

vel_ang=vel_lin*curvatura*(180.0/(1000.0*pi));

string=QString("curv=%1      velang=      %2      vel_lin=      %3").arg(curvatura).
arg(vel_ang).arg(vel_lin);
fprintf(fp, "%s\n",string.latin1());

double vel_ang_max;

vel_ang_max=100.0; //graus/s

       if(fabs(vel_ang)>vel_ang_max)
              {
                     signe=vel_ang/fabs(vel_ang);
                     vel_lin=(vel_ang_max/fabs(vel_ang))*vel_lin;
                     vel_ang = signe*vel_ang_max;
              }
```

```
string=QString("velang= %1, vellin=%2").arg(vel_ang).arg(vel_lin);
fprintf(fp, "%s\n",string.latin1());

printf("x_ob=%f,     y_ob=%f,     x_vehiculo=%f,     y_vehiculo=%f,phi_vehiculo=%f,
L=%f,incremento_x=%f,curvatura=%f,vel_ang=%f,   der_DX=%f\n",x_ob,y_ob,   x_vehiculo,
y_vehiculo,phi_vehiculo,L, incremento_x,curvatura, vel_ang, der_Dx);

veld.setVl(vel_lin); // mm per segon
veld.setVa(vel_ang);
DX_ant=incremento_x;
return(veld);
}

void GotoAgent::CalculaTrajectoria(int max_val)
{
       double m,n, difx; // millor si es double
       double pi=3.141596;
       int i;
       QString string;

       string=QString("pos_ini.x=%1,pos_ini.y=%2,   setpoint.x=%3,   setpoint.y=%4").
       arg(pos_ini.getX()).        arg(pos_ini.getY()).        arg(setpoint.getX()).
       arg(setpoint.getY());
       fprintf(fp, "%s\n",string.latin1());

       if((pos_ini.getX()-setpoint.getX())==0)
       {
              difx=(setpoint.getY()-pos_ini.getY())/max_val;

              for(i=0;i<max_val;i++)
              {
              x_camino[i]=pos_ini.getX();//5000*sin(2.0*3.6*i*pi/180.0)+5000;   posem
              tot en mm
              y_camino[i]=pos_ini.getY()+(i*difx);

              string=QString("x_camino[%1]=%2,       y_camino[%3]=%4").       arg(i).
              arg(x_camino[i]). arg(i). arg(y_camino[i]);
              fprintf(fp, "%s\n",string.latin1());
              }
       }

       else
       {
              m=(pos_ini.getY()-setpoint.getY())/(pos_ini.getX()-setpoint.getX());
```

```
            n=pos_ini.getY()-((pos_ini.getY()-setpoint.getY())/(pos_ini.getX()-
            setpoint.getX()))*pos_ini.getX();
            difx=(setpoint.getX()-pos_ini.getX())/max_val;

            for(i=0;i<max_val;i++)
            {
                  x_camino[i]=pos_ini.getX()+(i*difx);//5000*sin(2.0*3.6*i*pi/180.
                  0)+5000;
                  y_camino[i]=((m*x_camino[i])+n);
                  string=QString("x_camino[%1]=%2,    y_camino[%3]=%4").    arg(i).
                  arg(x_camino[i]). arg(i). arg(y_camino[i]);
                  fprintf(fp, "%s\n",string.latin1());
            }
      }

}

double GotoAgent::CalculateUtility(double distancia)
{
      double utilval;
      utilval=Vary.calcula_maxmin(distancia,150.0,1000.0,0.4,1.0);
      return utilval;
}

MUtil GotoAgent::ResourceExchange(MUtil winner, MUtil loser)
{
      double difvel,dtl,dtw,vl,va;
      double uLoser,uWinner;
      MUtil resultat;
      resultat=winner;

      if(!smooth) //abrupt
      canvi=false;

      else //smooth
      {
            uLoser=loser.getUtil();
            uWinner=winner.getUtil();

            if(uLoser<0.1)
                  uLoser=0.1;

            if(ctf==0.0)
            {
                  difvel=fabs(winner.getVelLin()-loser.getVelLin());
                  tf=calcula_tf(difvel);
```

```
            }

            if(ctf>=tf)
            {
            //printf("no formula\n");
            canvi=false;
            vl=winner.getVelLin();
            va=winner.getVelAng();
            }

            else
            {
                  ctf++;
                  dtl=Vary.calcula_maxmin(ctf, 0.0, tf, 0.0,uLoser);
                  dtw=Vary.calcula_minmax(ctf,0.0,tf,0.0,uWinner);
                  //printf("dtl=%f,dtw=%f,vl_l=%f,vl_w=%f,uL=%f,uW=%f\n",dtl,dtw,l
                  oser.getVelLin(),winner.getVelLin(),uLoser,uWinner);

                  vl=(dtl*loser.getVelLin()+dtw*winner.getVelLin())/(dtl+dtw);
                  va=(dtl*loser.getVelAng()+dtw*winner.getVelAng())/(dtl+dtw);
            }
            //printf("Tf=%f,ctf=%f,vl=%f,va=%f\n",tf,ctf,vl,va);

            if(vl<0.0)
                  vl=0.0;

            if(isnan(vl)!=0)
                  vl=0.0;

            if(isnan(va)!=0)
                  va=0.0;

            resultat.setVelLin(vl);
            resultat.setVelAng(va);
      }
      return resultat;
}

double GotoAgent::calcula_tf(double d)
{
      double figual, fpetita, fmitjana, fgran; //resultat de la campana de Gauss
      double sigual = 95.49 , spetita = 98.0, smitjana = 107.0,  sgran = 84.15; //
      sigma

      double uigual = 0.0 , upetita = 261.6 , umitjana = 503.0 , ugran = 693.0; //
      mu
```

```
        double temps;
        figual = Vary.calcula_gauss(d, sigual, uigual);
        fpetita = Vary.calcula_gauss(d, spetita, upetita);
        fmitjana = Vary.calcula_gauss(d, smitjana, umitjana);
        fgran  = Vary.calcula_gauss(d, sgran, ugran);
        temps=(figual*0.0+fpetita*3.0+fmitjana*5.0+fgran*10.0)/(figual+fpetita+fmitja
        na+fgran);
        temps=(figual*0.0+fpetita*4.0+fmitjana*7.0+fgran*15.0)/(figual+fpetita+fmitja
        na+fgran);
        //printf("\n           figual=%f,           fpetita=%f,           fmitjana=%f,
        fgran=%f,temps=%f\n",figual,fpetita,fmitjana,fgran,temps);

        return(temps);
}
```

A.2 Codi pel controlador per superfícies rugoses

```
#include "gotoagent.h"

GotoAgent::GotoAgent(Q_UINT16 server_port, QString hostname,Q_UINT16 df_port,
QString df_hostname)
 : BasicAgent(server_port,hostname)
{
        connect(this,SIGNAL(SigRegister()),SLOT(Register()));
        connect(this,SIGNAL(SigPosition()),SLOT(Position()));
        connect(this,SIGNAL(SigDPosition()),SLOT(DesiredPosition()));
        connect(this,SIGNAL(SigCloseAgent()),SLOT(CloseAgent()));
        connect(this,SIGNAL(SigUtility()),SLOT(Utility()));

        //estableix la informacio propia
        selfinfo.SetMutex(false);
        selfinfo.addSuppService(QString("DESIRED_LINEAR_SPEED"));
        selfinfo.addSuppService(QString("DESIRED_ANGULAR_SPEED"));
        selfinfo.addSuppService(QString("UTILITY"));
        selfinfo.addReqService(QString("DESIRED_COORDINATES"));
        selfinfo.addReqService(QString("DESIRED_HEADING"));
        selfinfo.addReqService(QString("ABSOLUTE_COORDINATES"));
        selfinfo.addReqService(QString("ABSOLUTE_HEADING"));
        selfinfo.addResource(QString("ROBOTAGENT"));

        df=CreateClient(df_hostname,df_port);
        // Es registra al DF
        RegisterAgent(df, AGENTNAME,selfinfo.getAllSuppService(),selfinfo.
        getAllReqService(),selfinfo. getAllResources(),selfinfo.GetMutex());
```

```
	// variables pid de posición
	c1.IniciaErrors();
	c2.IniciaErrors();
	//PID ràpid
	c1.LimitsVel(500.0,50.0);
	c1.ValorsConstants(0.4,0.1,0.0,0.1,1);//PID
	c1.ValorsConstants(1.0,1.2,0.0,0.1,0); //PID
	//PID Lent
	c2.LimitsVel(200.0,45.0);
	c2.ValorsConstants(0.5,0.2,0.0,0.1,1); //PID pos
	c2.ValorsConstants(1.0,0.5,0.0,0.1,0); //PID head ki=0.3
	//PID per a distancies curtes
/*		c2.LimitsVel(200.0,45.0);
		c2.ValorsConstants(0.5,0.2,0.0,0.1,1); //PID pos
		c2.ValorsConstants(1.0,0.3,0.0,0.1,0); //PID head
*/
/*		c2.LimitsVel(500.0,60.0);
		c2.ValorsConstants(0.8,0.2,0.0,0.1,1); //PID pos
		c2.ValorsConstants(1.0,0.5,0.0,0.1,0); //PID head
*//*		c2.ValorsConstants(1.0,0.5,0.0,0.1,1); //PID pos kp=5 i ki=3
		c2.ValorsConstants(2.0,1.0,0.0,0.1,0); //PID head kp=2, ki=1
*/
	canvi=false;
	control=1;
	smooth=true; // si false abrupt
//	fp=fopen("goto.txt","a+");
}

GotoAgent::~GotoAgent()
{
//	fclose(fp);
	 CloseClientConnection(df);
	QValueList<MInfo>::iterator itai;
	for(itai=agentinfo_list.begin();itai!=agentinfo_list.end();itai++) //per cada
	agent connectat
	{
		CloseClientConnection((*itai).getTCPClient());
	}
}

void GotoAgent::CloseAgent()
{
//	printf("emit closed\n");
	emit Closed();
}
```

```
/*void GotoAgent::CanviControl()
{
	canvi=true;
	vela.setX(utility.getX()); //la utilitat
	vela.setY(utility.getY()); //la velocitat lineal
	vela.setTh(utility.getTh()); //la velocitat angular
	ctf=0.0;
}*/

void GotoAgent::Register()
{
	ConnectToServer(selfinfo);
	//printf("\n CONNECTAT\n");
}

void GotoAgent::DesiredPosition()
{
//	flag=0;
}

void GotoAgent::Utility()
{
	UpdateUtility();
	//printf("Utility()\n");
}

void GotoAgent::Position()
{

/*	QStringList::iterator itlist;
	QValueList<MInfo>::iterator itai;
	QStringList mes;

	for(itai=agentinfo_list.begin();itai!=agentinfo_list.end();itai++) //per cada
agent connectat
	{
		mes=(*itai).getAllMessages();
		//busquem al llistat d'utilitats la posicio de l'agent que envia la
		utilitat
		for(itlist=mes.begin();itlist!=mes.end();itlist++)
		{
			fprintf(fp,(*itlist));
			fprintf(fp,(*itai).getAgentName());
			fprintf(fp,"\n");
		}
	}
```

```
*/     run();
}

void GotoAgent::run()
{
//     double pGoto=0.0 ; //variables que contendran la prioridad del Goto y
delAvoid
//     double Ts=0.1;// temps de mostreig del robot.
       //-------------------------------------------------------------------------
       //                         CALCULAR VELOCIDADES
       //-------------------------------------------------------------------------
//     double velocitat, velocitatR;
       double gir,distancia,edist,egir; // consigna de giro y distancia
       double dx,dy,mu_aprop;
       MSpeeds velc1,velc2;
       MUtil self_util;

       //distancia que queda para llegar al punto deseado
       dx=setpoint.getX()-pos.getX();
       dy=setpoint.getY()-pos.getY();
       distancia = sqrt(pow(dx,2)+pow(dy,2));
       //fp=fopen("goto.txt","a+");

       edist=distancia;
       // calcula l'angle de gir
       gir=atan2(dy,dx)*180.0/3.141596;

       if(distancia<50.0)
       {
              gir =setpoint.getTh();
       }
//     printf("ed=%f,gir=%f\n",edist,gir);
       egir=gir-pos.getTh();
       if(egir>180)
              egir=egir-360;
       else if(egir<-180)
              egir=360+egir;

       if(fabs(egir)>90)
              edist=0.0;

       //Controlador 1
       velc1=c1.AccioControl(edist,egir);

       //Controlador 2
       velc2=c2.AccioControl(edist,egir);
```

```
	// si es a prop del punt desti
	if(distancia<50.0)
	{
		//Si no ponemos esto en algunos casos cuando ya ha llegado al punto
correcto
		//Se desvía pq se sigue moviendo aunque sea poco.
		velc2.setVl(0.0);
		if(fabs(setpoint.getTh()-pos.getTh())<5)
		{
			velc2.setVa(0.0);
		}
		velc1.setVl(0.0);
		if(fabs(setpoint.getTh()-pos.getTh())<10)
		{
			velc1.setVa(0.0);
		}
	}

	if(distancia<100.0)
	{
		//Si no ponemos esto en algunos casos cuando ya ha llegado al punto
		correcto
		//Se desvía pq se sigue moviendo aunque sea poco.
		velc1.setVl(0.0);
		if(fabs(setpoint.getTh()-pos.getTh())<10)
		{
			velc1.setVa(0.0);
		}
	}

	if(isnan(velc1.getVl())!=0)
		velc1.setVl(0.0);
	if(isnan(velc2.getVl())!=0)
		velc2.setVl(0.0);
	/*if(velc1.getVl()<0.0)
		velc1.setVl(0.0);
	if(velc2.getVl()<0.0)
		velc2.setVl(0.0);*/
	//-------------------------------------------------------------------------
	//                      FUZZY COLLABORATIVE CONTROL
	//-------------------------------------------------------------------------
	mu_aprop=Vary.calcula_maxmin(edist,500.0,2500.0,0.0,1.0); //C1 500 i C2 200
	self_util.setVelAng(mu_aprop*velc2.getVa()+(1.0-mu_aprop)*velc1.getVa());
	//velocitat lineal
	self_util.setVelLin(mu_aprop*velc2.getVl()+(1.0-mu_aprop)*velc1.getVl());
	//velocitat angular
```

```
	//------------------------------------------------------------------------
	//                      UTILITY CALCULATION
	//------------------------------------------------------------------------

	self_util.setUtil(CalculateUtility(distancia));     //utilitat
	self_util.setControl(control);
	//printf("Utility=%f\n",self_util.getUtil());

	//------------------------------------------------------------------------
	//                    RESOURCE EXCHANGE
	//------------------------------------------------------------------------
	//si agent te control
//	self_util.setVelAng(0.0);
	//self_util.setVelLin(0.0);
	//printf("control=%d \t", control);
	if(control==1)
	{
			SendUtility(self_util,QString("UTIL"));
	}

	bool guanyo=true,ndata=false;
	HigherUtility(guanyo,ndata,self_util);
	//si tenim el control i hem guanyat enviem les dades al robot i a la resta
d'agents
	//printf("control=%d,guanyo=%d,newdata=%d\n",control,guanyo,ndata);
	if(ndata)
	{
		if( (control==1) && guanyo)
		{

			if(canvi)
				self_util=ResourceExchange(self_util,quiControla);
					//busquem al llistat d'utilitats la posicio de
l'agent que envia la utilitat
			SendSpeeds(self_util);
		}
		else
			if((control==0) && guanyo)//si no tinc el control i guanyo
			{
				canvi=true;
				ctf=0.0;
				control = 1;
				self_util=ResourceExchange(self_util,quiControla);
				SendSpeeds(self_util);
				SendUtility(self_util,QString("UTIL"));
```

```
                }
                else  //si no he guanyat perdo el control
                       if (!guanyo)
                              control =0;
       }
       else
       {
              if (control==1)
              {
                     if(canvi)
                            self_util=ResourceExchange(self_util,quiControla);
                     SendSpeeds(self_util);
              }
       }
       SendUtility(self_util,QString("UTILITY"));
       //     printf("vel=%f, ang=%f, util=%f\n",velocitat,velocitatR,pGoto);
       //fprintf(fp,"velL=%f, velR=%f, ctf=%f, tf=%f, canvi=%d,
control=%d,guanyo=%d\n", self_util.getVelLin(), self_util.getVelAng(), ctf-1, tf,
canvi, control,guanyo);
}

double GotoAgent::CalculateUtility(double distancia)
{
       double utilval;

       utilval=Vary.calcula_maxmin(distancia,150.0,500.0,0.6,1.0);
       return utilval;
 }

MUtil GotoAgent::ResourceExchange(MUtil winner, MUtil loser)
{
       double difvel,dtl,dtw,vl,va;
       double uLoser,uWinner;
       MUtil resultat;

       resultat=winner;

       if(!smooth) //abrupt
              canvi=false;
       else //smooth
       {
              uLoser=loser.getUtil();
              uWinner=winner.getUtil();
              if(uLoser<0.1)
                     uLoser=0.1;
              if(ctf==0.0)
```

```
		{
			difvel=fabs(winner.getVelLin()-loser.getVelLin());
			tf=calcula_tf(difvel);
		}
		if(ctf>=tf)
		{
			//printf("no formula\n");
			canvi=false;
			vl=winner.getVelLin();
			va=winner.getVelAng();
		}
		else
		{
			ctf++;
			dtl=Vary.calcula_maxmin(ctf, 0.0, tf, 0.0,uLoser);
			dtw=Vary.calcula_minmax(ctf,0.0,tf,0.0,uWinner);

	//printf("dtl=%f,dtw=%f,vl_l=%f,vl_w=%f,uL=%f,uW=%f\n",dtl,dtw,loser.getVelLi
n(),winner.getVelLin(),uLoser,uWinner);
			vl=(dtl*loser.getVelLin()+dtw*winner.getVelLin())/(dtl+dtw);
			va=(dtl*loser.getVelAng()+dtw*winner.getVelAng())/(dtl+dtw);
		}
		//printf("Tf=%f,ctf=%f,vl=%f,va=%f\n",tf,ctf,vl,va);
		if(vl<0.0)
			vl=0.0;
		if(isnan(vl)!=0)
			vl=0.0;
		if(isnan(va)!=0)
			va=0.0;

		resultat.setVelLin(vl);
		resultat.setVelAng(va);


	}
	return resultat;
}

double GotoAgent::calcula_tf(double d)
{
	double figual, fpetita, fmitjana, fgran; //resultat de la campana de Gauss
	double sigual = 95.49 , spetita = 98.0, smitjana = 107.0,  sgran = 84.15; //
	sigma
	double uigual = 0.0 , upetita = 261.6 , umitjana = 503.0 , ugran = 693.0; //
	mu
	double temps;
```

```
        figual = Vary.calcula_gauss(d, sigual, uigual);
        fpetita = Vary.calcula_gauss(d, spetita, upetita);
        fmitjana = Vary.calcula_gauss(d, smitjana, umitjana);
        fgran  = Vary.calcula_gauss(d, sgran, ugran);
        temps=(figual*0.0+fpetita*3.0+fmitjana*5.0+fgran*10.0)/(figual+fpetita+fmitja
        na+fgran);
        //
        temps=(figual*0.0+fpetita*4.0+fmitjana*7.0+fgran*15.0)/(figual+fpetita+fmitja
        na+fgran);
        //printf("\n figual=%f, fpetita=%f, fmitjana=%f,
        fgran=%f,temps=%f\n",figual,fpetita,fmitjana,fgran,temps);
        return(temps);
}
```