EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

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

Títol: CONTROL DE POSICIÓ DEL ROBOT WIFIBOT

Document: 1 Memòria

Alumne: Rafael Hesse

Director/Tutor: Albert Figueras Coma
Departament: Enginyeria Elèctrica, Electrònica i Automàtica
Àrea: ESA

Convocatòria (mes/any): juliol / 2008

ÍNDEX

1. INTRODUCCIÓ

1.1. Antecedents

La robòtica és una branca de l'enginyeria actualment en ple desenvolupament. L'aparició constant de noves tecnologies de comunicació, sensors i mecànica fa possible la construcció de robots cada vegada més complexos per tota mena d'aplicacions.

D'entre les línies de desenvolupament de la robòtica podem distingir dues branques diferents. Per una banda es desenvolupen robots "senzills" cada vegada més petits dissenyats a efectuar una tasca concreta i determinada amb mínim cost i tamany i màxim rendiment. Per altra banda es desenvolupen robots amb cada vegada més capacitat de càlcul capaços de realitzar tasques diverses i complexes com ara la presa de decisions davant determinades circumstàncies i interactuar amb l'humà.

A partir del treball amb diversos tipus i models de robots de gamma mitja-baixa com ara els robots futbolistes i el robot Rogi el departament d'Electrònica i Automàtica de la Universitat de Girona ha adquirit un nou robot de gamma superior.

Aquest nou robot anomenat Wifibot 4G de Roboserv disposa d'entre altres de comunicació per WiFi, això i la seva estructura més complexa en varis nivells el conforma com un robot de gamma mitja-alta.

A diferència dels tipus de robot tractats fins ara en el departament el Wifibot 4G destaca per sobre de tot en un punt, la possibilitat de programar i controlar en tres nivells diferents. El nou nivell introduït és la placa microcontrolada que gestiona les comunicacions mitjançant un sistema operatiu Linux, donant un molt ampli ventall de noves possibilitats donada la potència de càlcul d'aquest sistema.

Val destacar que tot i que es tracta d'un robot amb un ventall de molt ampli de possibilitats de control i programació la documentació subministrada pel fabricant es limita als conceptes bàsics del robot.

Aquesta documentació està formada per diverses fulles de característiques del propi robot així com dels seus principals components. A més conté dos senzills programes que permeten guiar el robot mitjançant un joystick des d'un ordinador amb sistema Windows i un altre per al sistema Linux. Per altra banda la documentació no especifica l'estructura

precisa del hardware ni dóna instruccions sobre les eines necessàries per a la programació en els diversos nivells.

1.2. Objecte

Primerament aquest projecte pretén presentar de forma clara i detallada l'estructura i el funcionament del robot així com dels components que el conformen. Aquesta informació és de vital importància a l'hora de desenvolupar aplicacions per al robot.

Un cop descrites les característiques del robot s'analitzaran les eines necessàries i/o disponibles per poder desenvolupar programari per cada nivell de la forma més senzilla i eficient possible.

Posteriorment s'analitzaran els diferents nivells de programació i se'n contrastaran els avantatges i els inconvenients de cada un. Aquest anàlisi es començarà fent pel nivell més alt i anirà baixant amb la intenció de no entrar en nivells més baixos del necessari. Baixar un nivell en la programació suposa haver de crear aplicacions sempre compatibles amb els nivells superiors de forma que com més es baixa més augmenta la complexitat.

A partir d'aquest anàlisi s'ha arribat a la conclusió que per tal d'aprofitar totes les prestacions del robot és precís arribar a programar en el nivell més baix del robot.

Finalment l'objectiu és obtenir una sèrie de programes per cada nivell que permetin controlar el robot i fer-lo seguir senzilles trajectòries.

1.3. Especificacions i abast

Donant que durant l'anàlisi de les possibilitats de control s'ha arribat a la conclusió de que es necessari arribar al nivell més baix de programació s'ha plantejat realitzar petites modificacions en l'arquitectura del robot.

Aquestes modificacions es limiten al control dels motors i a l'aprofitament de tots els ports disponibles dels controladors del nivell més baix. D'aquesta manera augmenten les prestacions del robot en quant a les possibilitats de control i de funcionament semiautònom.

Aquest projecte presenta el disseny d'aquestes modificacions que en cap cas modifiquen l'estructura general del robot. A partir d'aquest punt es presenten els programes dissenyats per a cada nivell.

Els programes presentats permeten mitjançant càlculs d'odometria fer anar el robot a una consigna de posició determinada. El disseny de les comunicacions entre els diferents nivells permetrà incorporar altres eines de localització en els nivells superiors que interactuaríen amb els càlculs d'odometria. Les comunicacions es dissenyaran de forma que no serà necessari programar en el nivell més baix del robot agilitzant el disseny de noves aplicacions.

2. DESCRIPCIÓ WIFIBOT 4G

Aquest apartat descriu el funcionament i l'estructura del robot tal i com va ser adquirit al fabricant. Les modificacions realitzades en el control dels motors no es reflecteixen en aquest apartat per tal d'il·lustrar la necessitat de la realització d'aquestes modificacions.

El Wifibot 4G és un robot mòbil sobre quatre rodes d'altes prestacions. En la Figura 1 veiem el seu disseny 4x4 que permet el seu ús en múltiples situacions. L'estructura mecànica del robot està dividida en dues parts unides per un eix mòbil. D'aquesta manera el robot s'adapta a terrenys irregulars i li permet superar petits obstacles.

Figura 1. Robot Wifibot 4G.

2.1. Processador Mesh Cube

L'element central del robot és la unitat wifi (Mesh Cube de 4G Systeme). Aquest bloc format per un processador MIPS AMD Alchemy Au1500 a 400 MHz (Figura 2) actua com a pont entre l'usuari i els nivells més baixos del robot. Es tracta d'un processador d'arquitectura mipsel (*little endian*) amb 64 MB de RAM i 32 MB de memòria flash.

Figura 2. Processador Au1500

La Taula 1. Prestacions del AMD Au15000 representa les principals característiques del processador tot i que la disponibilitat d'aquestes ve donada per la circuiteria en la qual està implementat, en el nostre cas el Mesh Cube.

Velocitat del nucli: 400 MHz
Set d'instruccions MIPS32
Arquitectura 32 bits
16kB caché per instruccions i 16kB caché per dades
Tensió de treball del nucli: 1.5 – 1.8 V
Tensió per ports d'entrada i sortida: 3.3 V
Consum: 1.2 W
Controlador PCI 32-bits (PCI 2.2) a 33/66 MHz
Dos controladors ethernet 10/100 kb
Controlador USB per actuar com a dispositiu "host"
Controlador USB per dispositius externs
2 ports UART
Controlador AC-97
Controlador per PCMCIA
Controlador per SDRAM a 100/125 MHz
Controlador per SRAM/Flash EPROM

Taula 1. Prestacions del AMD Au15000

La Figura 3. Esquema del processador mostra de forma esquemàtica l'estructura del processador i dels elements descrits en la taula anterior.

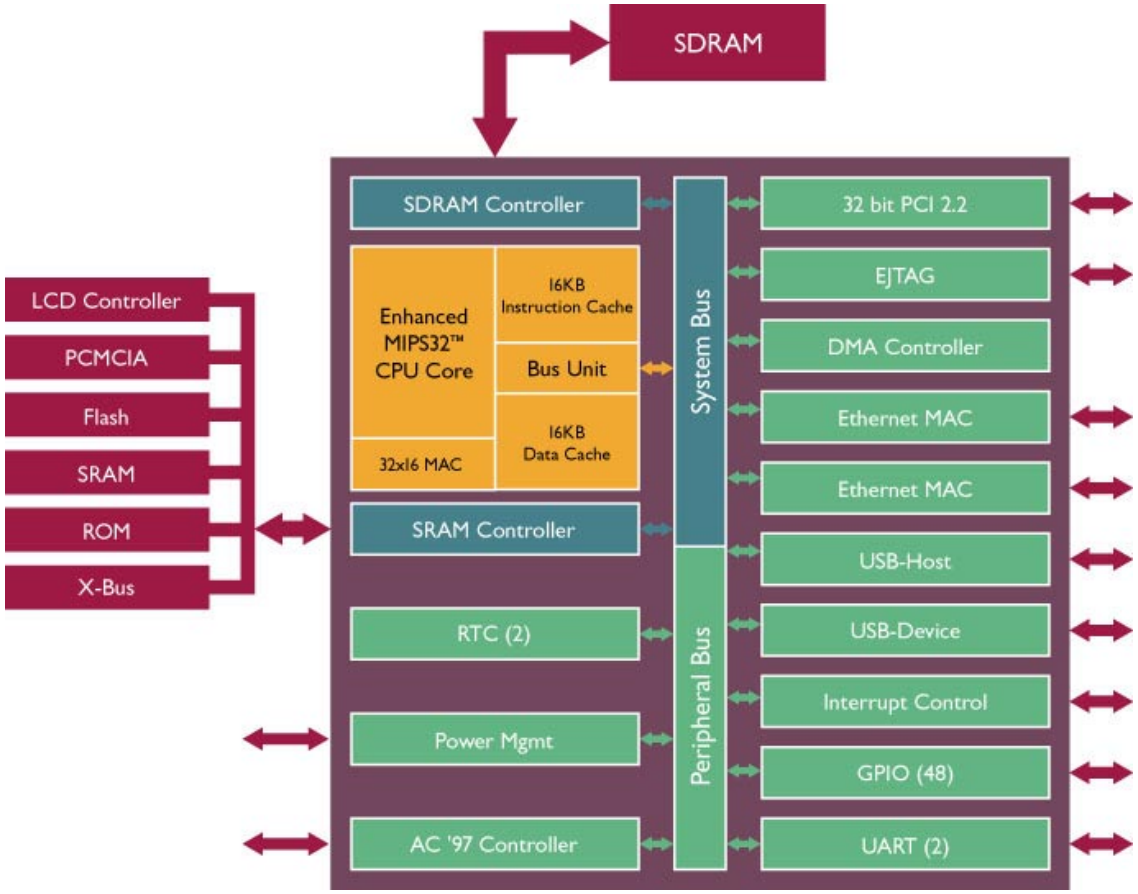

Figura 3. Esquema del processador

Com ja s'ha comentat el processador descrit s'inclou en el Mesh Cube (Figura 4. Mesh Cube) el qual es pot entendre com un punt d'accés wifi (router). Per tal de gestionar el trànsit wifi així com la resta de comunicacions en el cub corre un sistema operatiu Linux. Nylon, la distribució Linux que corre en el cub està dissenyada específicament per aplicacions wifi portant ja incorporades les funcions de gestió de comunicacions. En els propers capítols d'aquest projecte ens referirem a aquesta unitat com "el cub" per tal de referir-nos al processador central del robot.

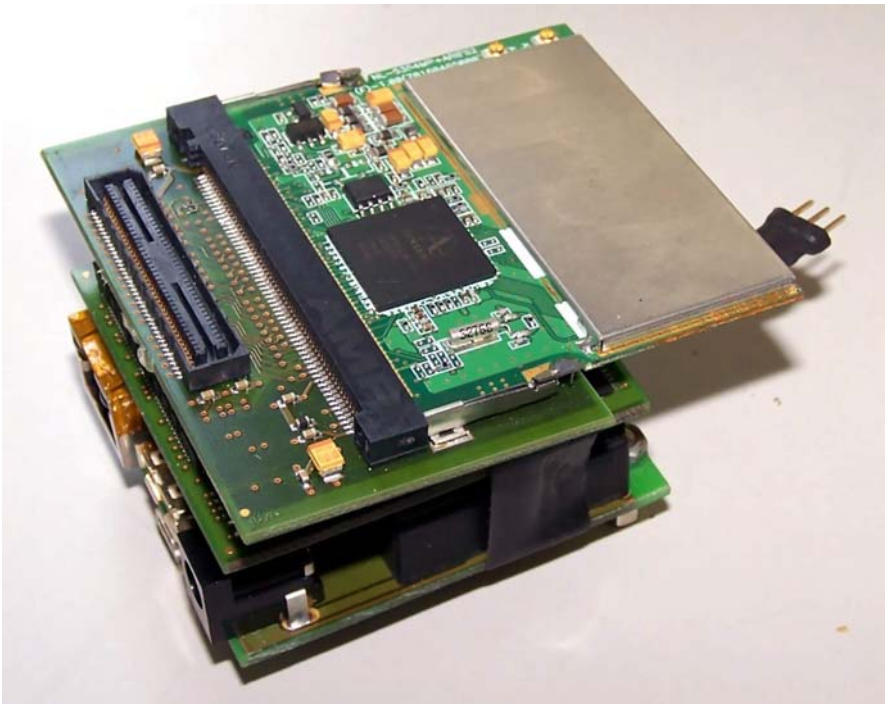

Figura 4. Mesh Cube

A part de disposar de comunicació wifi el cub conté un punt d'accés de xarxa LAN que li permet comunicar-se amb altres elements del robot.

La Taula 2 detalla les principals aplicacions que conté el sistema tot i que aquestes son les que conté de fàbrica podent-se actualitzar i afegir noves aplicacions via internet.

Linux Kernel 2.4.27
Mesh Routing
Web server
DNS Server
DHCP Server / DHCP Client
Firewall
Perl

Taula 2 Principals aplicacions del processador

Cal destacar que a part dels dispositius que es descriuran a continuació el cub disposa d'un port mini PCI no accessible des de l'exterior del robot. Amb els adaptadors adequats aquest port es pot ampliar fins un màxim de 4 ports. La Figura 5 mostra un esquema general de les connexions del robot, en base a aquest esquema s'explicaran els diferents dispositius.

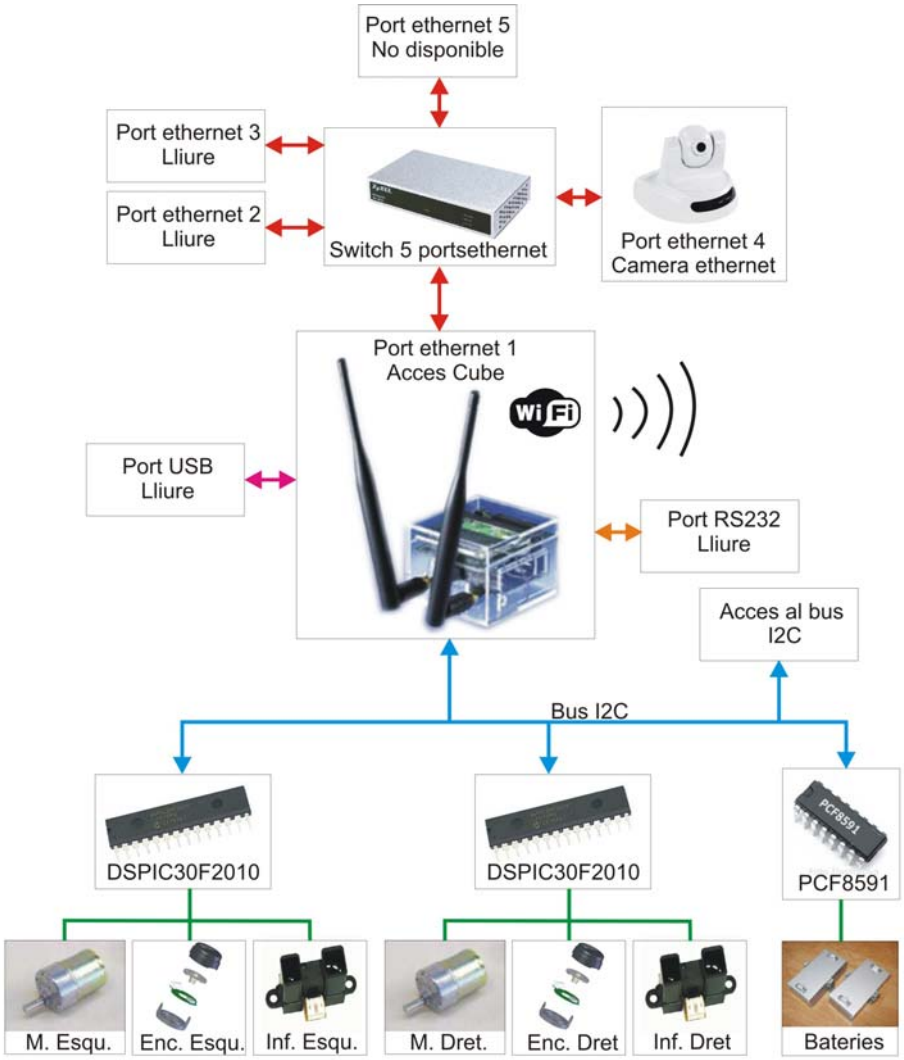

Figura 5 Esquema general del robot

2.2. El port sèrie

Es tracta d'un port sèrie estàndard RS232. Aquest port comunica de forma directa amb el processador del cub, es troba accessible des de l'exterior del robot a través d'un connector estàndard. El propòsit d'aquest port és connectar perifèrics addicionals al robot ja siguin estàndards o autofabricats. La Figura 6 mostra en quins pins del connector DB-9 es troben les senyals.

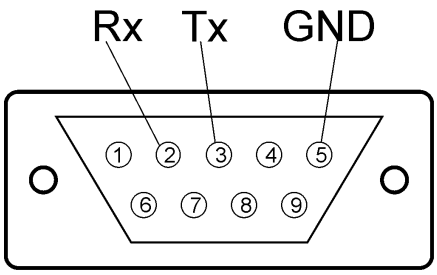

Figura 6. Connexionat RS232

Tal com es detalla en les especificacions del processador aquest disposa de dos ports UART de forma que en cas de necessitat es podria afegir un accés al segon port

2.3. El bus I^2C

El bus I^2C permet la comunicació entre el cub i els nivells més baixos, d'aquesta manera el processador fa la funció de pont entre els nivells més baixos i l'usuari. En el bus hi ha connectats tres microcontroladors que es detallen més endavant La Taula 3 detalla els elements connectats i la corresponent direcció en format hexadecimal. Externament tenim un connector DB-9 que permet afegir nous dispositius I^2C al robot fins a un total de 127 sent el processador sempre el mestre del bus. El límit de 127 elements en el bus ve donat per la tipologia de les adreces que son de 7 bits, tot i que existeixen dispositius que suporten adreces de 10 bits amb la qual cosa augmenta el nombre màxim de dispositius del bus, no es pot aplicar aquest sistema en el robot donat que el conversor analògic-digital no suporta aquest sistema.

Dispositiu	Adreça
dsPIC30F2010 (costat esquerra)	0x51
dsPIC30F2010 (costat dret)	0x52
PCF8591 (conversor A/D)	0x48

Taula 3. Adreces del bus I^2C

Cal destacar que la direcció del conversor analògic-digital no es correspon amb la direcció que descriu el fabricant en la documentació del robot. Aquesta direcció no es pot modificar donat que ve fixada per la circuitería de l'integrat i no és programable. Per altra banda la direcció dels dsPIC's es fixa per programació i pot ser modificada segons les necessitats.

L'alimentació del bus s'obté a partir de la font del cub de 3.3 volts de manera que si es volen afegir nous dispositius aquests hauran de ser compatibles amb aquest nivell de tensions. Molt sovint els nivells requerits per sensors o microcontroladors amb I^2C son de

5 volts. La Figura 7 mostra el connector I^2C del cub amb les línies SDA, SCL i l'alimentació.

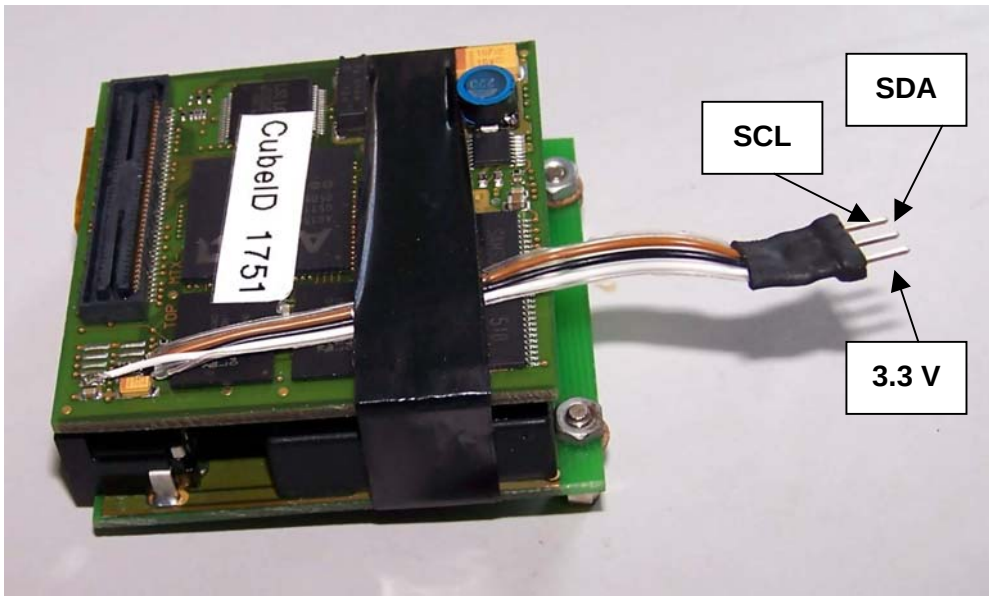

Figura 7. Port I^2C del cub

La configuració del bus es pot definir com mono-mestre donat que l'únic mestre del bus es el cub. De fet el programa que corre en el cub no permet que cap altre element actuï com a mestre en el bus donat que el cub ocupa permanentment el bus, inclòs quan no transmet dades. Més endavant veurem que aquesta programació es pot modificar per tal de permetre una configuració multimestre.

La Figura 8 mostra el connexionat per al connector del bus, aquest es troba en el costat dret del robot, davant del connector RS232.

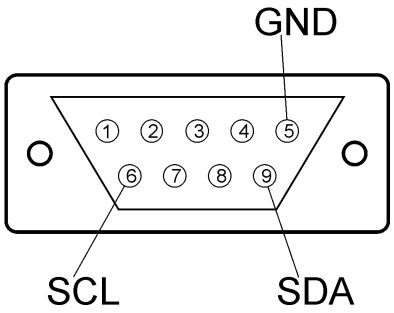

Figura 8 Connexionat del bus I2C

2.4. Els ports USB

El cub disposa de dos connectors USB els quals són per propòsits diferents. Per una banda tenim un connector mini USB que connecta amb el device controller del

processador, amb aquest podem connectar el cub a un PC actuant com a perifèric del PC.

Per altra banda tenim el connector USB normal que connecta amb el host controller del cub. En aquest port podem connectar dispositius que actuaran com a perifèrics del cub. La Figura 9 mostra els dos connectors situats en la part inferior del cub.

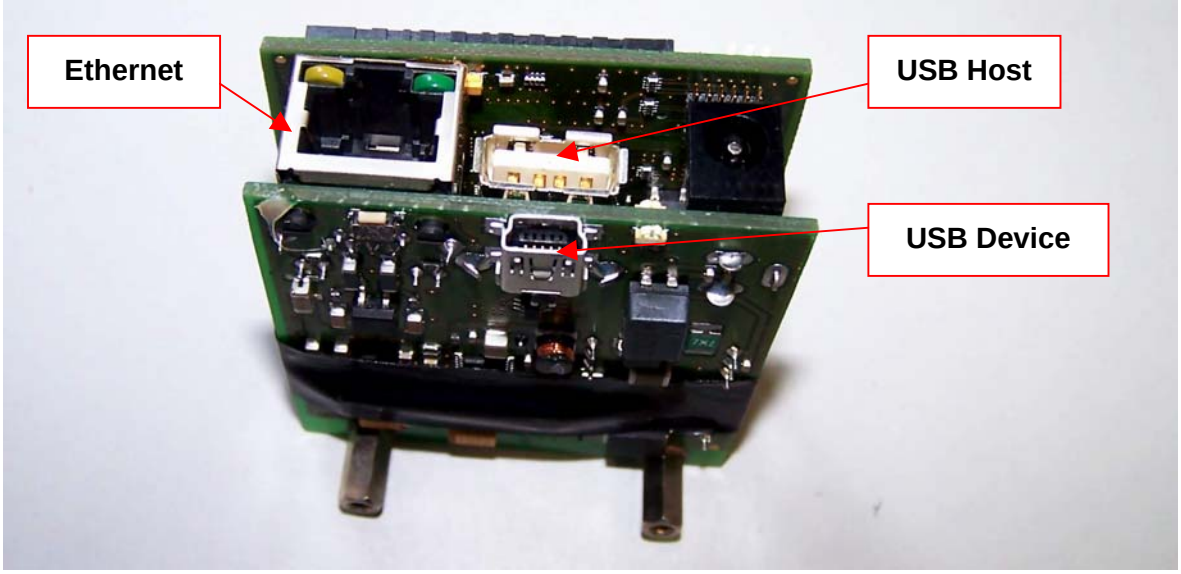

Figura 9. Connectors USB del cub

El connector USB per perifèrics porta una extensió per tal de fer-lo accessible des de fora del robot. Actualment no hi ha cap aplicació específica per aquest port per la qual cosa no hi ha controladors per dispositius USB instal·lats en el cub. No obstant es podrien connectar tot tipus de dispositius USB com per exemple memòries sempre i quan es disposi dels controladors compatibles amb Linux.

2.5. Els motors i rodes

El robot es mou sobre quatre rodes accionades per motors de corrent contínu. Cada roda està acoblada directament a l'eix d'un motor (GHM-04 de Lynxmotion). Els motors porten incorporat un engranatge de reducció amb una relació de 50:1. La Figura 10 mostra l'aspecte del motor i del reductor acoblat.

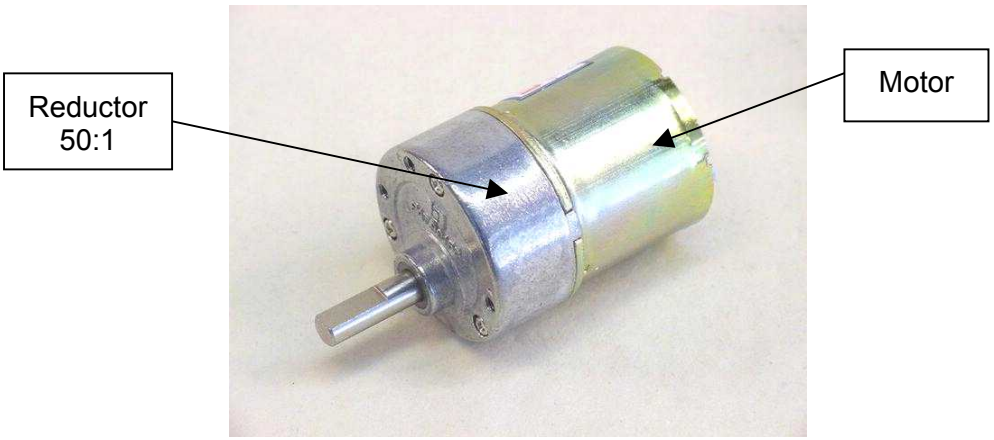

Figura 10. Motor del robot

En la Taula 4 es resumeixen les prestacions d'aquest motor donades pel fabricant.

Model:	HN-GH7.2-2414T - 50:1
Rang de voltatge:	6 – 7.2 Vdc
Reductor	50:1
Consum a màxima càrrega	3.6 A
Velocitat en buit	175 rpm
Diàmetre de l'eix	6 mm
Càrrega màxima	1 kg-cm

Taula 4. Prestacions del motor

La velocitat dels motors es controla mitjançant modulació de polsos els quals són controlats per una parella de dsPIC's. Tot i que tenim quatre motors independents la velocitat d'aquests no es controla de forma independent. Cada dsPIC controla els dos motors d'un costat del robot de forma simultània.

A l'hora de programar cal tenir en compte que aquests motors no permeten canvis de polaritat instantanis, segons el fabricant cal inserir un temps mort abans d'invertir la polaritat de l'alimentació.

Cada parella de motors connectats en paral·lel es controlen a través d'un driver de potència (L298) el qual també te connectat en paral·lel les seves dues entrades i sortides. D'aquesta manera s'assegura que cada driver suporta el corrent de dos motors.

Les rodes de goma del robot son del mateix fabricant que els motors (Figura 11) es tracta del model "soft" donat el pes relativament baix del robot. Aquestes s'acoblen a l'eix del motor mitjançant un accessori mostrat al costat dret de la figura.

Figura 11. Rodes del robot i sistema de muntatge

La Taula 5 detalla les especificacions de les rodes

Model	TRC-02
Material de la base	Nylon
Material de la rodadura	Goma
Material del interior	Espuma
Diàmetre	5” (12.7 cm)
Ample	2.25” (5.7 cm)

Taula 5. Especificacions de les rodes

2.6. Els encoders

Els dos motors de les rodes davanteres van equipats amb un encoder òptic en quadratura (E4 Miniature Optical Kit Encoder de US Digital). Les rodes de darrera no porten encoder donat que com hem comentat la velocitat d’aquestes serà la mateixa que les de davant, tret que no estigui bloquejada alguna d’elles.

En aquest cas la documentació del fabricant discrepa amb la configuració real donat que el fabricant descriu que el robot conté un encoder en cada roda i diferent model.

L’encoder s’acobla al motor per la part de darrera de forma que compta directament amb el gir del motor i no del reductor o les rodes. La Figura 12 mostra en detall l’encoder i el muntatge en el motor.

Figura 12. Encoder. Detall i muntatge

A continuació es detallen les característiques d'aquests encoders.

Màxima vibració (5-20 kHz)	20 g
Alimentació	5 Vdc
Màxima acceleració	25.000 rad/s^2
Nº de sectors	120
Nº de canals	2 (desfasament 90º)
Diàmetre per l'eix	2 mm

Taula 6. Especificacions dels encoders

El disc dels encoders està dividit en 120 sectors que es detecten mitjançant dos sensors òptics. Aquests sensors llegeixen el disc amb un desfasament de 90º proporcionant una parella de senyals com es detalla a la Figura 13.

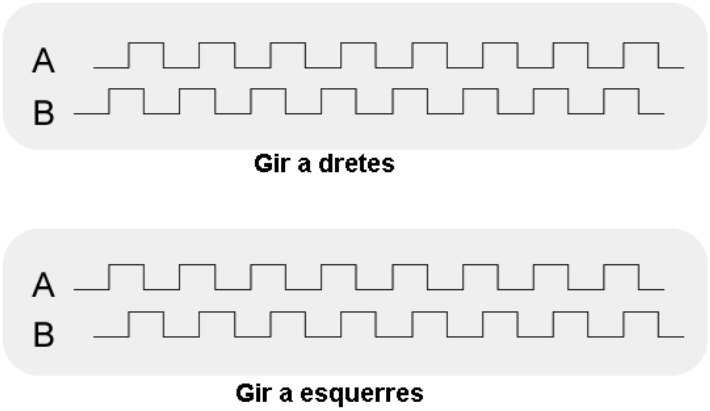

Figura 13. Lectura dels encoders

L'ordre en que es succeeixen els polsos de cada canal ens permet determinar el sentit de gir. La resolució final del encoder depèn de la forma en que es conten els polsos. Per una banda podem aprofitar només un dels canals o tots dos i per altra banda podem comptar només flancs en un sentit o en ambdós sentis. La Taula 7 mostra les resolucions obtingudes.

Canals llegits	Flancs llegits	Informació de sentit de gir	Res. motor (polsos / rev.)	Res. roda (polsos / rev.)
A o B	Només pujada o baixada	No	120	6.000
A o B	Pujada i baixada	No	240	12.000
A i B	Només pujada o Baixada	Si	240	12.000
A i B	Pujada i baixada	Si	480	24.000

Taula 7. Resolucions del encoder

La finalitat dels encoders principalment és controlar la velocitat de les rodes per tal de tancar el llaç del control de velocitat PID implementat en els dsPIC's.

2.7. Els sensors de distància

En la part davantera el robot porta dos sensors infrarojos (Figura 14) de distància (GP2Y0A02YK de Sharp). Aquests sensors produeixen una sortida analògica de voltatge inversament proporcional a la distància d'un objecte.

Figura 14. Sensor GP2Y0A02YK

El funcionament d'aquest sensor es basa en el temps de rebot sobre un obstacle d'un feix de llum infraroja. Aquest temps és proporcional a la distància de l'objecte. La sortida analògica donada per aquests sensors es llegeix a través de les entrades analògiques del dsPIC on s'haurà d'interpretar el valor llegit en volts per obtenir la distància en centímetres.

La Figura 15 mostra la resposta del sensor, es pot observar que el marge de treball es troba entre 15 i 150 centímetres.

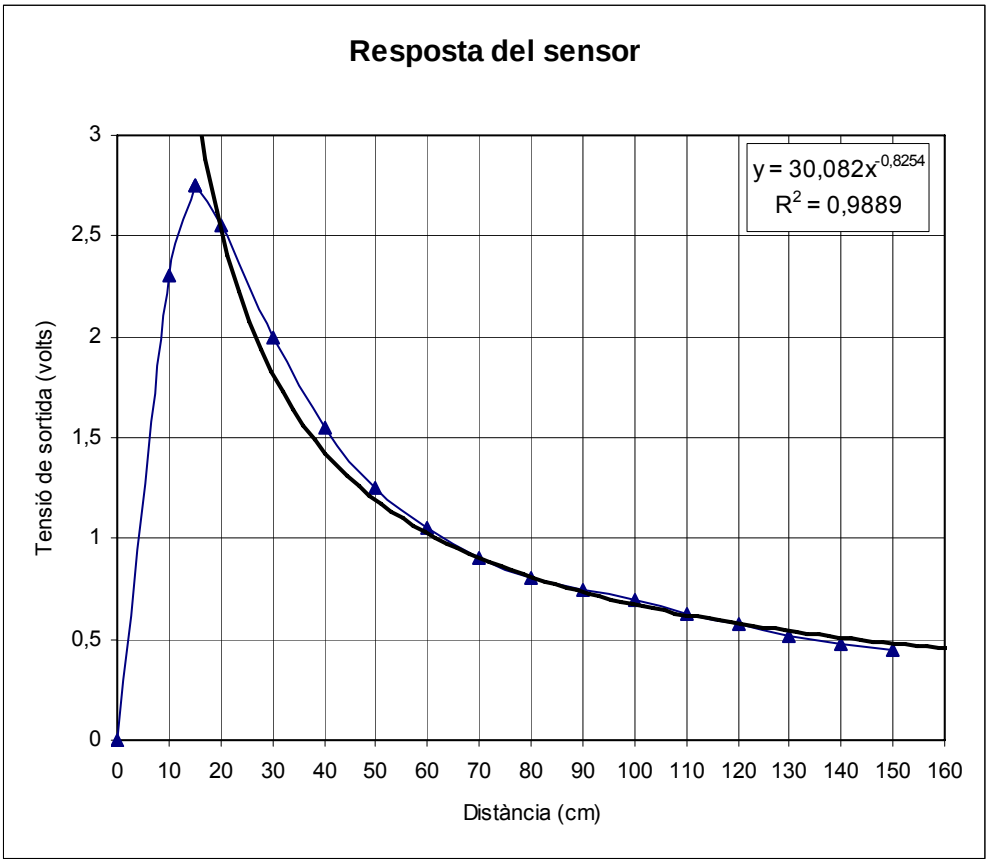

Figura 15. Resposta del sensor de distància

La resposta d'aquests sensors no és lineal, per obtenir un valor proporcional a la distància cal processar la senyal i linealitzar-la.

Per tal d'obtenir el valor de la distància en centímetres tenim diverses opcions. Gràficament, opció no implementable per software, a partir d'una equació aproximada o extrapolant el valor a través d'una taula de valors obtinguda gràficament.

La gràfica mostrada anteriorment conté una línia de tendència potencial per al marge útil de lectures que s'aproxima prou bé a la realitat L'equació 1 mostra la fórmula resultant.

$$D = 30.082 \cdot V^{-0.8254} \qquad \text{(Eq. 1)}$$

On "D" equival a la distància en centímetres i V equival al voltatge de sortida del sensor.

La Taula 8 resumeix les característiques elèctriques del sensor.

Alimentació	4.5 – 5.5 Vdc
Tensió de sortida	-0.3 a Vcc + 0.3 Vdc
Consum	33 mA
Rang de mesura	20 – 150 cm

Taula 8. Especificacions del sensor GP2Y0A02YK

2.8. El dsPIC30F2010

Els dsPIC's pertanyen a la família de PIC's de Microchip d'actualment més altes prestacions. Cada dsPIC controla una meitat del robot, és a dir, una parella de motors, el corresponent encoder i un sensor de distància.

La Taula 9 resumeix les prestacions de cada un d'aquests dsPIC's tot i que la majoria de ports no es fan servir per cap propòsit.

Encapsulat SDIP 28 pins
12 kbytes de memòria de programa
512 bytes de memòria SRAM
1024 bytes de memòria EEPROM
3 Timers de 16 bits
6 canals de PWM (3 parelles)
6 entrades analògiques de 10 bits
1 Entrada per encoder en quadratura
1 Port de comunicacions UART
1 Port de comunicacions SPI
1 Port de comunicacions I^2C
2 Pins alternatius per al port UART
3 pins alternatius per als ports de depuració

Taula 9. Prestacions del dsPIC30F2010

Igual que en les generacions anteriors de PIC's els mòduls descrits anteriorment es poden configurar tots com a ports d'entrada i/o sortida digital segons les necessitats de cada aplicació.

A diferència de les series anteriors de PIC's els dsPIC's treballen sobre una arquitectura de 16 bits de manera que augmenta notablement la capacitat de càlcul així com la velocitat.

A part de portar implementades les operacions de càlcul bàsiques (suma i multiplicació) els dsPIC's tenen nous mòduls de càlcul que implementen funcions complexes com potències i multiplicacions de 16 bits. L'implementació d'aquestes funcions per hardware augmenta múltiples vegades la velocitat de càlcul per aquests casos.

Val destacar que si fos precís aquest dsPIC té patillatge compatible amb altres dsPIC's de la mateixa família amb majors prestacions de memòria.

La Figura 16 detalla la estructura interna del dsPIC30F2010 incloent els ports d'entrada i sortida.

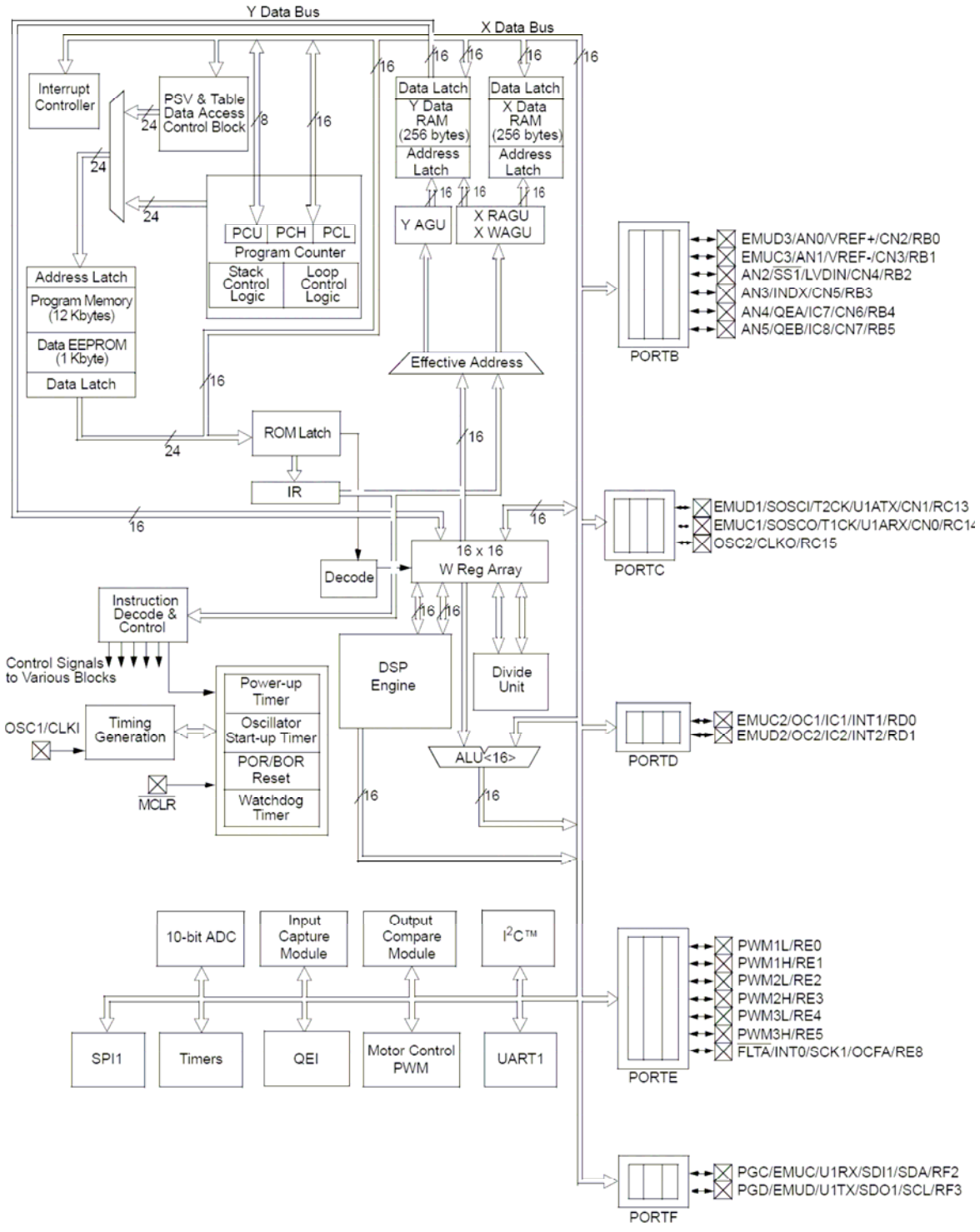

Figura 16. Estructura del dsPIC30F2010

Aquests dsPIC's treballen a partir d'un cristall de 7.3728 MHz escalat per software a una freqüència de 29.4912 MHz.

Finalment la Figura 17 mostra la distribució de tots els ports del dsPIC sobre les 28 potes de l'encapsulat SDIP utilitzat en el robot.

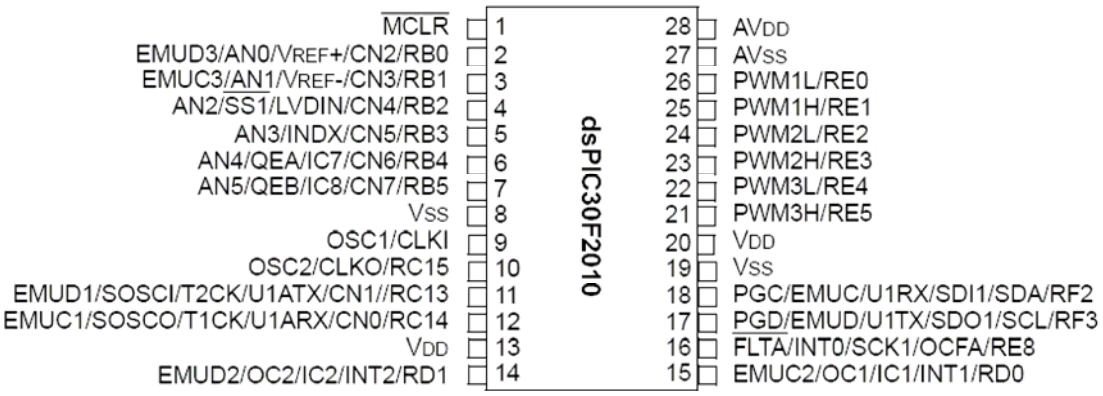

Figura 17. Distribució del dsPIC 30f2010

2.9. El PCF8591 Conversor Analògic-digital

Aquest integrat únicament es fa servir per controlar el nivell de les bateries del robot. Es tracta d'un conversor bidireccional analògic-digital de 8 bits compatible amb I^2C. En la Figura 18 observem un diagrama de blocs explicatiu sobre el funcionament d'aquest conversor.

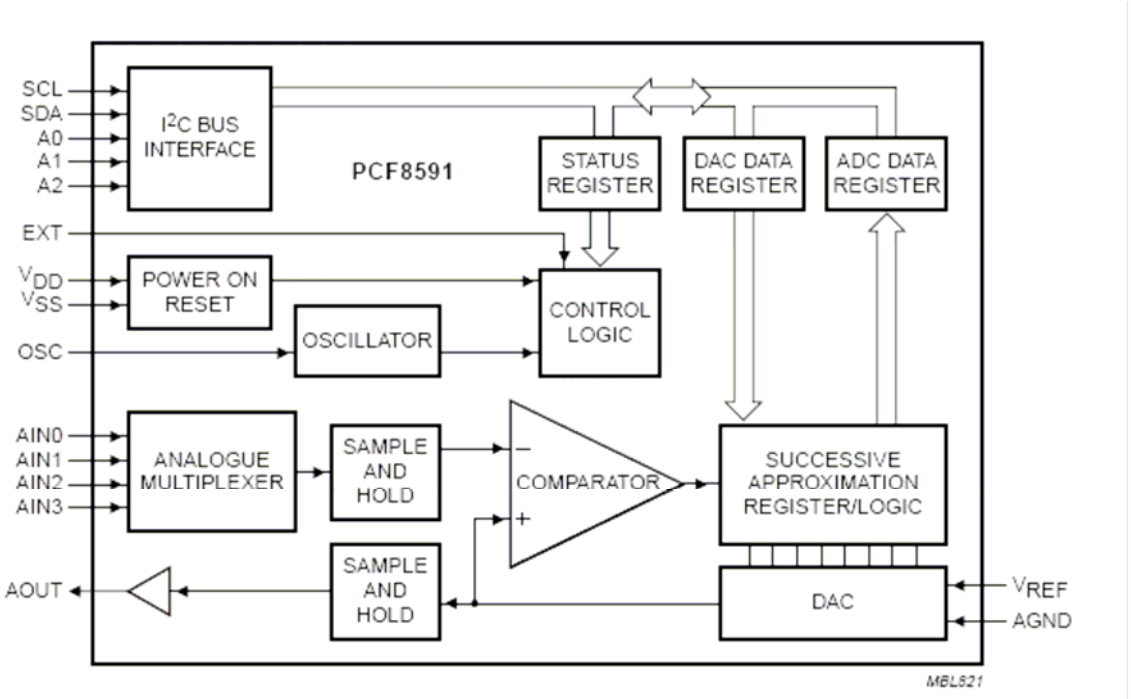

Figura 18. Diagrama de blocs del conversor AD

Tot i que disposa de quatre canals d'entrada i un de sortida en aquest propòsit només es fa servir un sol canal d'entrada. D'aquesta manera es pot monitoritzar el nivell de les bateries des dels nivells de programació superiors. La forma en que es connecta la bateria a l'entrada del conversor es detalla en el plànol 3 que mostra el circuit de la font d'alimentació.

La Figura 19 detalla el patillatge de l'integrat per al cas de l'encapsulat SDIP fet servir en el robot.

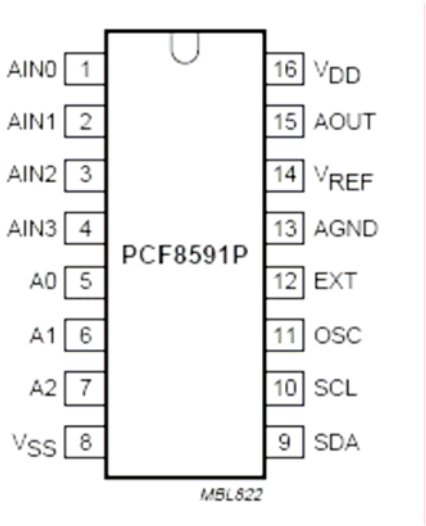

Figura 19. Connexió del conversor AD

La direcció que té aquest dispositiu dintre un bus I^2C depèn de la configuració. Aquesta configuració es realitza per hardware de forma que no serà modificable. En concret es tracta d'una part de direcció fixa i una part programable. Donat que els bits de direcció programables (Figura 20) estan posats a massa (plànol 3) la direcció resultant del dispositiu és 0x48 en hexadecimal o 0x90 / 0x91 si s'afegeix el bit de sentit de comunicació.

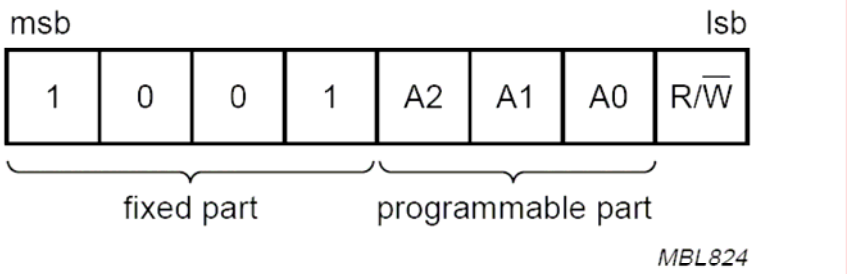

Figura 20. Adreça del conversor AD

Més endavant es detalla amb més precisió el funcionament d'aquest dispositiu, especialment la configuració per al bus I^2C i les dades que cal enviar per tal de configurar el conversor per al nostre propòsit.

2.10. Switch 5 ports 10/100 ethernet

Aquest element enllaça el cub amb altres elements ethernet. Tot i que teòricament es tracta d'un switch de 5 ports a efectes pràctics només és de 4 ports donat que per qüestions d'espai només té quatre connectors (Figura 21).

Figura 21. Switch 5 ports 10/100 ethernet

D'aquests quatre connectors un està ocupat pel cub d'aquesta manera podem connectar-nos al processador mitjançant ethernet. Un altre connector està ocupat per la camera digital. Finalment els dos connectors que resten estan lliures per connectar nous elements.

2.11. La camera digital

Es tracta d'una camera ethernet amb control de pan – tilt (Figura 22). La camera envia les imatges via ethernet cap al switch, a partir d'aquest punt es poden veure en un PC o altre dispositiu connectat via ethernet al switch o connectat via wifi en el cub.

Figura 22. Camera ethernet

Les imatges de la camera són compatibles amb directX o Java amb la qual cosa es poden visualitzar en qualsevol navegador web que disposi d'una d'aquestes aplicacions. La Taula 10 resumeix les característiques de vídeo i elèctriques de la camera.

Resolució del sensor	640 x 480 pixel
Tecnologia del sensor	¼" color CMOS
Sistema de lents	F: 6.0mm, F:1.8
Compressió de la imatge	JPEG
Rati d'imatges	30 fps – QQVGA 25 fps – QVGA 10 fps – VGA
Rati de compressió	5 nivells configurables
Resolucions admeses	160 x 120, 320 x 24, 640 x 480
CPU	RDC R2880
RAM	8 MB
Flash ROM	2 MB
OS	RTOS
Alimentació	5 Vdc
Consum	6.5 Watt
Marge de PAN	-170º ~ +170º
Marge de TILT	+45º ~ -90º

Taula 10. especificacions de la camera.

2.12. La font d'alimentació

Donat que els diferents dispositius que conformen el robot treballen a diferents nivells de tensió el robot incorpora una font d'alimentació commutada. Aquesta font proporciona una tensió estable de 9,6 volts per a la càmera, el cub i el switch i una tensió de 5 volts per als dsPIC's, encoders i sensors.

El circuit de la font es detalla en el plànol 3.

3. COMUNICACIONS

En aquest apartat s'expliquen les comunicacions des del nivell més alt fins al nivell més baix. Aquests protocols depenen de la programació i per tant els programes que corren en els dsPIC's, en el cub i en el PC han de ser compatibles entre ells.

De forma general el punt d'accés wifi del robot es pot configurar per tres modes operatius diferents La Figura 23 descriu els modes operatius de forma gràfica.

Solitari **Infrastructura** **En manada**

Figura 23. Modes operatius del robot

En el mode operatiu en solitari un únic robot actua com un punt d'accés wifi. El sistema de control extern (per exemple un PC) es connecta com a client en aquest robot. Segons les aplicacions implementades també es poden connectar més d'un client simultàniament.

En el mode operatiu en manada permet l'interacció de varis robots. En aquest cas un robot actua com a punt d'accés wifi i la resta de robots i/o sistemes externs es connecten en aquest com a clients.

El mode infrastructura pot estar format per un o varis robots, tots ells actuen com a clients d'una xarxa wifi externa. Aquesta xarxa pot estar formada per un o varis punts d'accés. El mode infrastructura també permet configurar diverses manades de forma que poden interactuar entre elles a través de la xarxa wifi externa.

Per al nostre cas el mode operatiu serà sempre en solitari donat que només disposem d'un robot.

3.1. Comunicació entre un PC o similar i el robot

Ens podem comunicar amb el robot o la camera amb un PC per dues vies diferents, wifi i ethernet. Aquestes comunicacions es caracteritzen per les direccions IP. Aquestes direccions es poden configurar segons les necessitats o el gust de l'usuari, tot i això s'han mantingut les configuracions de fàbrica.

3.1.1. Comunicació via ethernet

Connectant un cable ethernet en un dels dos ports lliures del switch podem tenir accés directe a la camera o al processador (Figura 24). Per tal propòsit la direcció IP del PC ha de ser fixa i dins del rang de direccions que fa servir el robot.

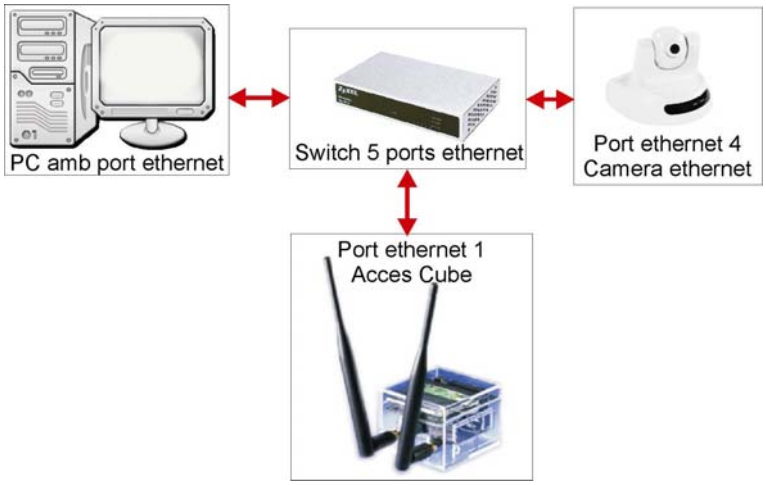

Figura 24. Comunicacions ethernet

La Taula 11 descriu les direccions IP de cada part d'aquesta xarxa, evidentment el marge de direccions possibles per al PC exclou aquelles que ja estan donades a altres dispositius.

Dispositiu	IP	Port
Robot	192.168.0.103	22 i 15000
Camera	192.168.0.20	80
PC	192.168.0.0-255	--

Taula 11. Direccions ethernet

Mentre que la direcció de la càmera correspon a la direcció que porta de fàbrica la direcció del cub es correspon amb el número de sèrie del robot (les 3 últimes xifres).

Val destacar que en els exemples sempre s'exposa un PC com a punt de comunicació aquest no necessita ser estrictament un PC, pot ser qualsevol dispositiu amb punt d'accés de xarxa com ara consoles de jocs, PDA's o mòbils.

3.1.2. Comunicació via WiFi

En el cas de comunicació inalàmbrica accedim al processador de forma directa i a la camera hi accedim de forma indirecta a través del switch i el cub (Figura 25). Igual que en el cas anterior la direcció IP del PC ha de ser estàtica i dins del rang especificat.

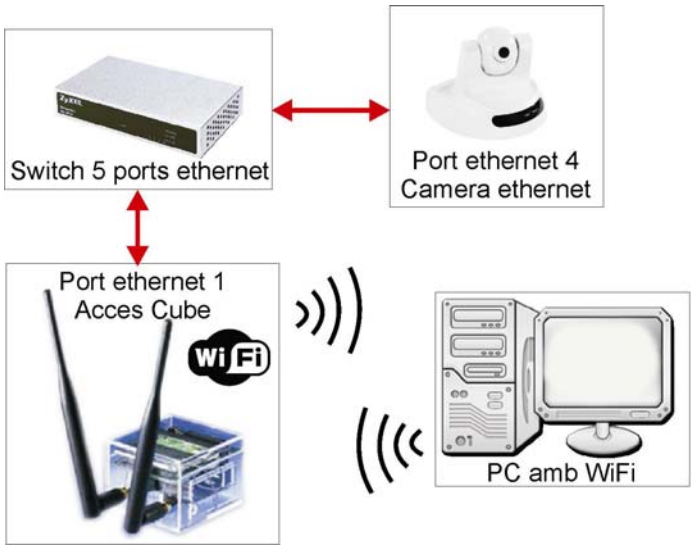

Figura 25. Comunicacions WiFi

La Taula 12 descriu les direccions IP de cada part d'aquesta xarxa, evidentment el marge de direccions possibles per al PC exclou aquelles que ja estan donades a altres dispositius. En aquest cas la direcció IP de la camera es redirecciona a través del cub i per tant es la mateixa diferenciant-se en el port.

Dispositiu	IP	Port
Robot	192.168.1.103	22 i 15000
Camera	192.168.1.103	80
PC	192.168.1.1-255	--

Taula 12. Direccions ethernet

3.1.3. Configuració de les adreces

Les direccions esmentades anteriorment són les direccions que estan configurades en el robot de fàbrica. Així mateix es poden modificar tant en el robot com en la càmera.

Per tal d'entrar en la configuració de la càmera es pot fer a través d'un navegador web. Marcant la direcció de la càmera veurem un menú interactiu de configuració. Per més detalls es pot consultar el manual de la càmera.

Per tal de reconfigurar el cub es necessita entrar en el sistema Linux i editar els arxius pertinents tal com descriu el manual.

3.2. Comunicacions I^2C

En els nivells més baixos del robot trobem el bus de comunicacions I^2C. Tot i que és possible accedir al bus, aquest no està dissenyat per comunicar amb l'usuari sinó per comunicar entre diferents controladors. En el cas del robot aquest bus comunica els dos dsPIC's, el conversor AD i el cub entre ells (Figura 26)

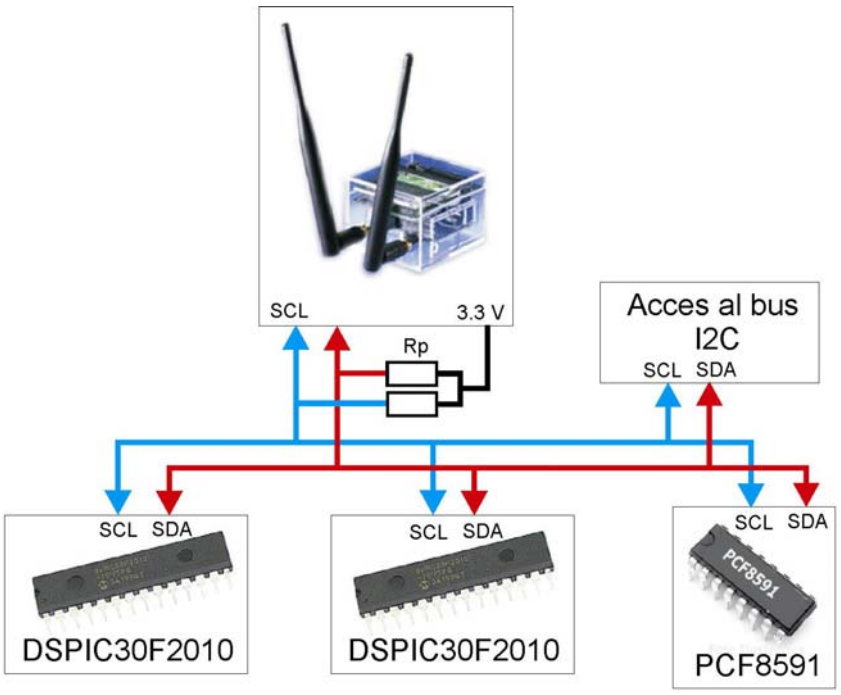

Figura 26. Esquema del bus I^2C

El bus I^2C dissenyat i patentat per Philips semiconductors permet la comunicació entre diferents dispositius mitjançant una línia de dades i una línia de rellotge. A continuació es fa una breu explicació del protocol I^2C.

3.2.1. Hardware

Tal i com es pot observar en la figura el bus es conforma per dues línies que uneixen tots els dispositius del bus. Aquestes dues línies han d'anar referenciades a una tensió positiva a través de dues resistències de "pull-up" (Rp). En el nostre cas el bus es referència a una tensió de 3.3 volts donat que el mòdul I^2C del cub no suporta tensions superiors.

La quantitat de dispositius que es poden connectar en un bus I^2C es limita per 2 paràmetres. Per una banda per un sistema d'adreces de 7 bits poden haver-hi un màxim de 127 dispositius i per altre banda la capacitat total de bus no pot superar els 400pF.

Aquest últim factor dependrà de les característiques de cada dispositiu connectat. Com a norma general es pot considerar que cada dispositiu introdueix una capacitat de 10 pF.

Donat que les línies SDA i SCL es referencien a una tensió positiva en estat de repòs aquestes línies representaran un “1” lògic. Per transmetre dades o senyal de rellotge l’emissor haurà de forçar la corresponent línia a l’estat baix.

3.2.2. Definicions

Per tal de poder entendre el funcionament del bus I^2C primerament cal definir una quants conceptes.

Emissor: Dispositiu que emet la informació. En el bus només hi pot haver un sol emissor en tot moment.

Receptor: Dispositiu connectat en el bus que rep les dades de l’emissor. En un bus normalment només hi pot haver un sol receptor. Existeix un cas especial en el qual un dispositiu en el bus fa la funció d’emissor i tots els altes fan de receptor.

SDA: Línia de dades del bus. Aquesta línia transmet les dades en consonància amb la línia de rellotge des de l’emissor al receptor.

SCL: Línia de rellotge del bus. Serveix per sincronitzar la transmissió. Aquesta senyal sempre la genera el mestre del bus.

Mestre: Dispositiu connectat en el bus que inicia la comunicació contra un esclau. El mestre sempre és el dispositiu que genera la senyal de rellotge. Un mestre tant pot emetre dades a un esclau com rebre dades d’un esclau.

Esclau: Dispositiu connectat en el bus que pot ser cridat per un mestre a través de la seva adreça. Un cop un esclau ha estat cridat aquest tant pot rebre dades com enviar-ne al mestre. En tot cas és el mestre qui decideix en quin sentit han d’anar les dades.

General call: És un cas especial de comunicació en la qual un mestre transmet dades a tots els esclaus a la vegada, en aquest cas els esclaus no poden respondre sense que es generi abans una crida personalitzada.

A partir d’aquestes definicions es dedueix que hi ha dues modalitats de transmissió de dades, transmissió de mestre a esclau, transmissió d’esclau a mestre i crida general (general call)

3.2.3. Sincronització de les dades

Per tal de rebre correctament les dades aquestes es transmeten per la línia SDA sincronitzades amb la línia SCL (Figura 27).

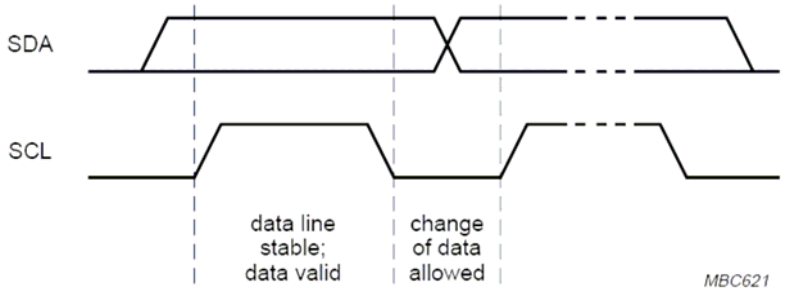

Figura 27. Sincronisme del bus I^2C

L'estat alt de la línia de rellotge indica que la dada en la línia SDA és estable i pot ser llegida. Per altra banda l'estat baix del rellotge indica que hi ha una transició en la línia de dades i no pot ser llegida.

En principi les dades s'envien a través del bus en paquets de 8 bits (1 byte). Igualment hi ha una sèrie d'events especials que no es composen per un paquet de 8 bits.

3.2.4. Condició especial de start i stop

Aquests dos casos són excepcions del descrit anteriorment donat que generen transicions en la línia de dades durant el període alt del rellotge.

Aquestes dues condicions són sempre generades pel mestre del bus. La condició de start indica inici de comunicació, a partir d'aquest punt el mestre començarà a generar la senyal de rellotge i mentrestant cap altre mestre pot enviar dades. El start es defineix per un flanc de baixada en la línia de dades durant el període alt del rellotge. La condició de stop indica final de transmissió i obliga a l'esclau (si es que està transmetent) a interrompre la transmissió. La condició de stop es defineix per un flanc de pujada en la línia de dades durant el període alt del rellotge. La Figura 28 detalla gràficament aquestes dues condicions.

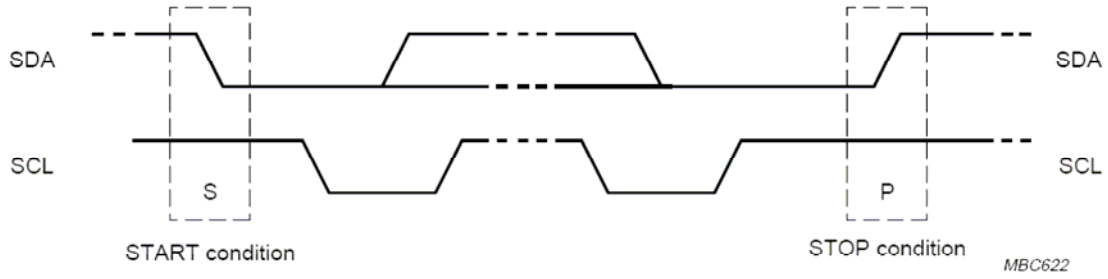

Figura 28. Condició de Start i de Stop

3.2.5. Condició Aknowledge (confirmació)

Durant la transmissió de dades el receptor tant si es mestre com si és esclau ha de confirmar cada byte rebut generant la condició d'Aknowledge (ACK). Aquesta condició especial es genera després del vuitè bit transmès de cada byte. Durant el novè pols de rellotge generat pel mestre del bus el transmissor ha de deixar lliure la línia de dades de manera que el receptor l'haurà de forçar a estat baix per tal de confirmar la recepció. Si aquesta condició no es genera indicarà que no s'han rebut les dades i s'interromprà la comunicació. La Figura 29 mostra gràficament la generació de l'estat ACK

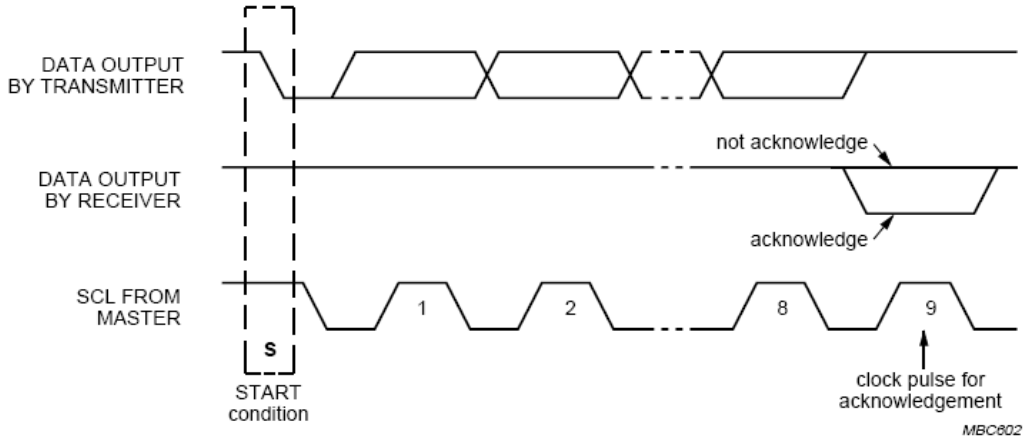

Figura 29. Condició ACK

No generar la senyal ACK, és a dir, Not Aknowledgment (NACK) indicarà al transmissor que ha de parar l'emissió, això podrà ser degut a que el receptor no ha rebut l'ultima dada o a que no vol rebre més dades.

3.2.6. Adreçat

Cada element connectat en un bus pot ser mestre o esclau en un moment determinat, és a dir, tot i que no pot ser les dues coses simultàniament si ho pot ser en moments

diferents. Aquells dispositius que es troben configurats com a esclaus necessiten una adreça. Aquesta adreça ha de ser única en el bus.

Les adreces poden ser de 10 o de 7 bits. En aquest cas només s'exposarà la comunicació per al cas de adreces de 7 bits donat que el sistema de 10 bits no es compatible amb el robot (degut al conversor AD).

Quan un mestre es vol comunicar amb un esclau la primera dada que s'envia (després de la condició de start) és l'adreça. Totes les dades transmeses pel bus es formen en paquets de 8 bits (1 byte). Donat que l'adreça només és de 7 bits ens permet enviar el vuitè bit com un bit de configuració.

Aquest bit (R/W) definirà en quin sentit es transmetran les dades després d'enviar l'adreça. Si R/W = 0 llavors el mestre transmetrà dades cap a l'esclau en canvi si R/W = 1 serà l'esclau qui haurà d'enviar dades al mestre. La Figura 30 mostra així la composició del byte d'adreça.

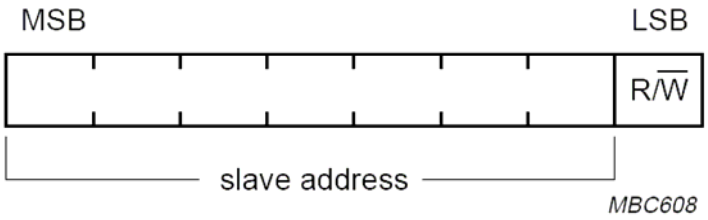

Figura 30. Composició de l'adreça

3.2.7. Seqüència "master send"

La Figura 31 representa una seqüència d'emissió de mestre a esclau en el bus. De color gris es marquen les accions generades pel mestre mentre que el color blanc marca les accions generades per l'esclau. En aquest cas el mestre transmet dos bytes a l'esclau.

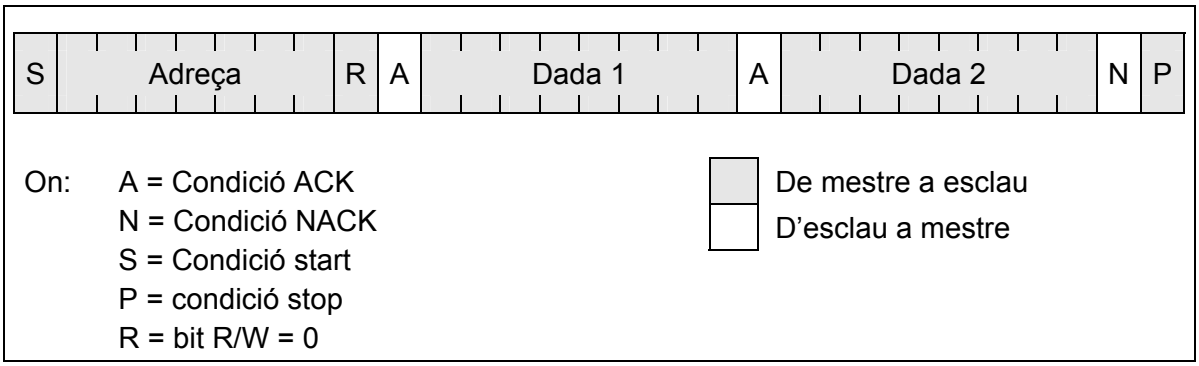

Figura 31. Seqüència master send

3.2.8. Seqüència “master recieve”

La Figura 32 representa amb la mateixa tipologia que l’anterior la seqüència a seguir quan un esclau envia dades a petició del mestre. En aquest cas concret el mestre crida l’esclau i aquest li envia dos bytes

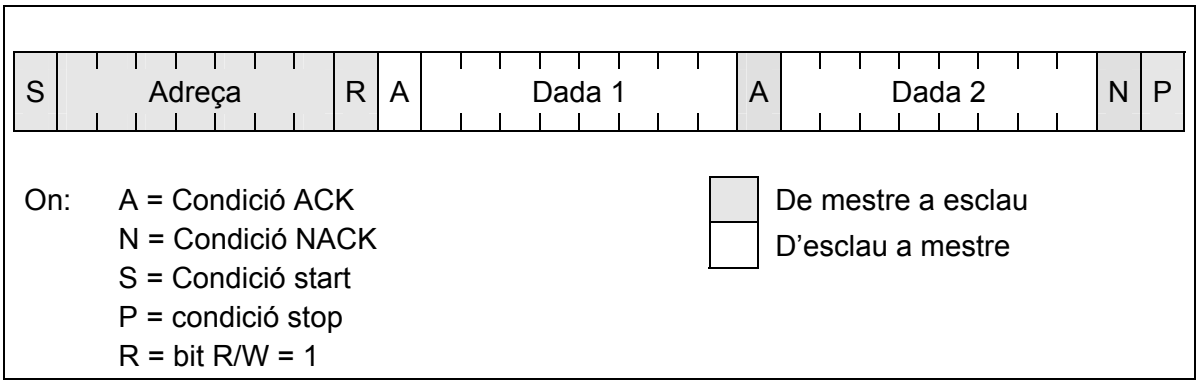

Figura 32. Seqüència master recieve

3.2.9. Bus multimestre

Com ja hem comentat un bus pot tenir més d’un mestre de forma que pot passar que dos mestres intentin accedir simultàniament al bus. Aquesta situació es gestiona mitjançant un sistema aleatori. Si dos mestres intenten accedir a l’hora al bus aquests aniran transmetent les seves dades fins que un dels dos “perd” les dades. Es considera que un dels dos perd quan aquest ha enviat per la línia de dades un “1” lògic però llegeix un “0” degut a que l’altre ha enviat aquest “0”durant qualsevol fase de la transmissió. Quan un dels mestres perd aquest haurà de interrompre immediatament la comunicació i reintentar-ho en un altre moment. La Figura 33 il·lustra gràficament aquest sistema arbitrari.

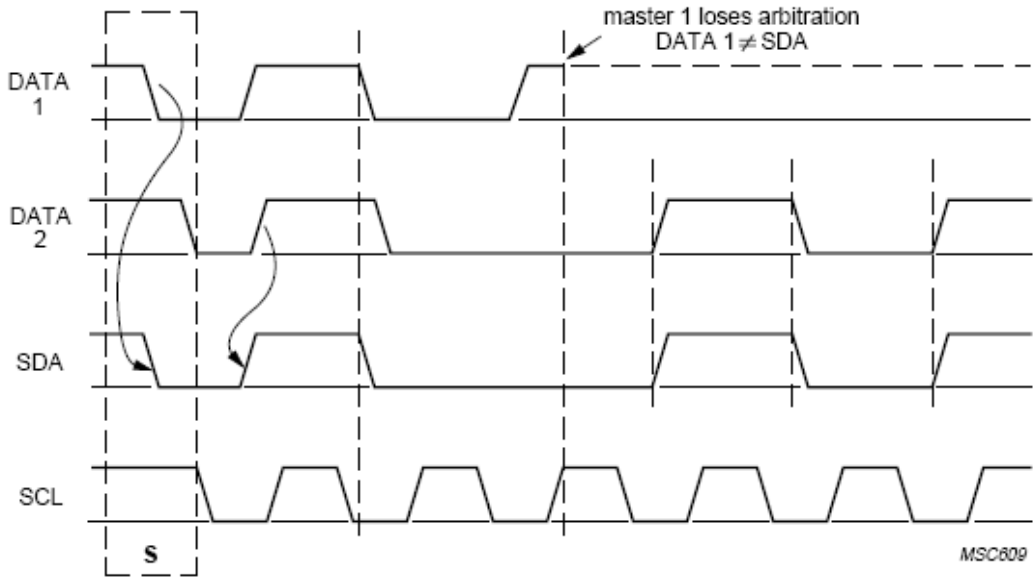

Figura 33. Sistema arbitrari en busos multimestre

3.2.10. Cas especial “general call”

Si un mestre inicia una seqüència de comunicació i envia la direcció “0” aquesta serà interpretada com una crida general. Les dades transmeses a continuació seran llegides per tots els esclaus del bus (sempre i quan suportin aquest sistema). Aquesta seqüència especial només existeix en un sentit, és a dir només el mestre pot enviar dades als esclaus mentre aquests no podran respondre.

4. NIVELLS DE CONTROL DEL ROBOT

A continuació es descriuen els tres nivells de control del robot des del punt de vista de no modificar la programació dels nivells inferiors es presenten els avantatges i els inconvenients. A part es descriuran les eines necessàries per la programació en cada nivell posant especial atenció en els nivells inferiors.

Per tal de controlar la posició del robot necessitem un programa que en funció de les lectures dels encoders ens determini la posició del robot, és a dir, determinar la distància recorreguda. Per tant els nivells de programació s'analitzaran des d'aquest punt de vista.

Aquest programa pot estar situat en tres plataformes diferents, la Figura 34 representa els tres nivells de programació gràficament.

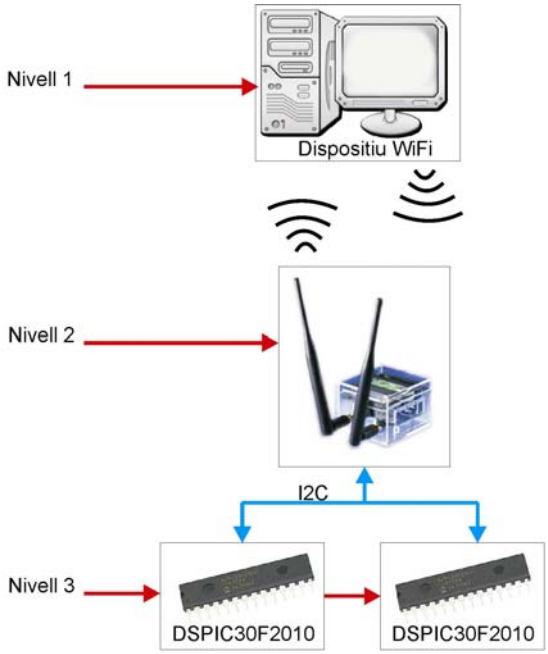

Figura 34. Nivells de control

A partir d'aquest esquema és fàcil deduir que modificar la programació en un determinat nivell implica modificar també la programació dels nivells superiors.

4.1. Control des d'un PC

Si volem controlar la posició del robot sense alterar la programació del cub o la programació dels dsPIC's depenem dels paràmetres que ens proporcionen els programes que corren de forma estàndard en aquests dispositius. Com ja hem comentat

anteriorment es posa com a exemple la programació en un PC tot i que pot tractar-se de qualsevol dispositiu programable amb connexió WiFi.

La Figura 35 detalla les consignes que podem enviar des del exterior al robot per tal de controlar-lo.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ctl_vel_E	Sentit_E	Con_vel_E					

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ctl_vel_D	Sentit_D	Con_vel_D					

Con_vel_X Consigna de velocitat dels motors
Si Ctl_vel_X = 0: valor entre 0 i 60
Si Ctl_vel_X = 1: valor entre 0 i 40

Ctl_vel_X Control de velocitat PID
0: Desactivat
1: Activat

Sentit_X Sentit de moviment del robot
0: Enrera
1: Endavant

Figura 35. Consignes acceptades des de l'exterior del robot

Podem observar que tot i que disposa d'un control de velocitat PID aquest rep una consigna proporcional sense unitats. D'aquesta manera el control de velocitat es relatiu donat que podem fer que el robot vagi a velocitat constant però no a una velocitat coneguda.

La Figura 36 detalla el contingut de la resposta que ens proporciona el cub quan li enviem una consigna.

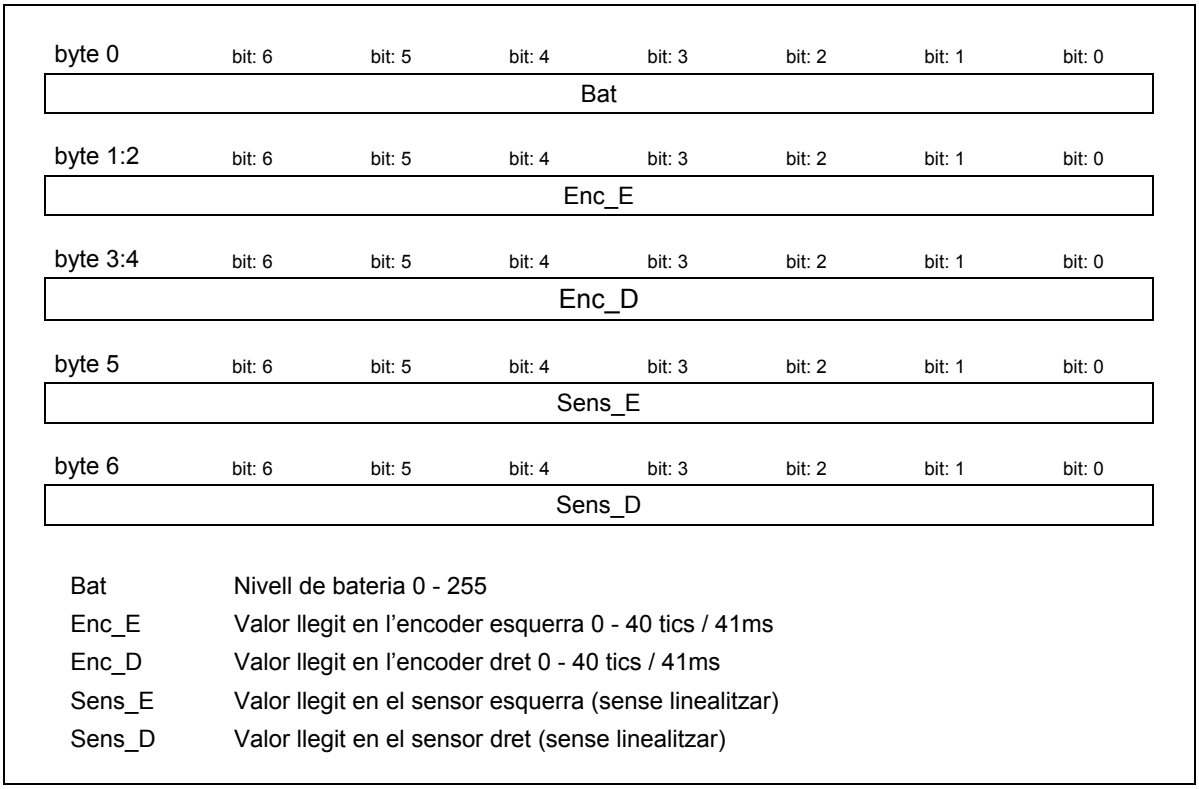

Figura 36. Valors retornats a l'exterior del robot

En quant a aquestes dades cal tenir en compte de que difereixen de lo descrit pel fabricant. Per una banda el fabricant descriu que el robot retorna valors de quatre encoders però donat que no disposa d'aquests encoders les dades s'envien de forma repetida. Per altra banda el fabricant diu que els valors dels sensors de distància es troben entre 0 i 150 cm. I en realitat no estan linealitzats, es tracta de la lectura del conversor AD del dsPIC sense tractar.

Programar exclusivament fora del robot presenta tota una sèrie d'inconvenients que fan pràcticament impossible fer un control de posició precís. El principal inconvenient el trobem en el retard que pot arribar a tenir la comunicació ethernet o wifi, en especial l'última. Donat que la distància recorreguda depèn del temps transcorregut un retard en la comunicació dóna lloc a errors de càlcul.

Per altra banda les dades que ens subministren els programes per defecte del cub i dels dsPIC's no són les idònies per realitzar càlculs d'odometria. Concretament els valors llegits en els encoders no són valors absoluts sinó tics comptats en un determinat període de temps, això ens dóna indirectament un valor de velocitat i no d'espai recorregut. Tot i que es poden realitzar càlculs d'odometria a partir de valors de velocitat evidentment no és el mètode més precís.

En conclusió podem dir que fins i tot modificant els programes dels nivells inferiors aquest no és un bon sistema per fer control de posició per odometria donat el retard incontrolable de la comunicació wifi. D'aquesta manera es reserva aquest nivell de programació per la presa de decisions de nivell més alt.

Les eines per tal de programar en aquest nivell no es detallaran donat que podem fer servir gairebé qualsevol eina en qualsevol llenguatge de programació.

4.2. Control des del Mesh Cube

En aquest nivell de programació desapareix l'inconvenient del retard de comunicació que presentava l'ethernet o wifi. Igualment persisteix l'inconvenient de que les dades subministrades pels programes dels dsPIC's no són adients per fer càlculs d'odometria.

Per tal de poder crear programes en aquest nivell necessitem entrar en el sistema operatiu del robot. Com ja s´ha comentat anteriorment es tracta d'un sistema Linux especialment adaptat per al tipus de processador. Existeixen diferents alternatives i aplicacions que ens permeten veure i modificar el sistema d'arxius del sistema Linux des de l'exterior. El fabricant subministra dos senzills programes que ens permeten realitzar aquesta tasca des d'un sistema Windows. Ambdues aplicacions, Putty i WinSCP es troben descrites en l'annex B. Aquest mateix annex descriu dues alternatives per tal d'obtenir una eina capaç de compilar codi que després es pugui executar en el robot.

Un cop s'ha entrat en el sistema d'arxius del robot es necessita modificar la instrucció que inicia el programa de comunicacions cada cop que s'inicia el sistema. Es pot substituir aquesta instrucció per una de nova que inicií el nostre nou programa o simplement eliminar-la i iniciar les aplicacions manualment.

Les dades de les quals es disposa en aquest nivell de programació són les mateixes que des del PC, donat que el programa que corre originalment en el cub no fa cap altre funció que de pont. Aquest programa (simple_server_ADC) descrit en l'annex A simplement recopila la informació enviada pels dos dsPIC's i el conversor AD a través del bus I^2C i la reenvia a través de la xarxa wifi o LAN.

L'annex A recull el codi d'uns senzills programes que es poden executar en el robot anomenats "recte", "gir" i bateria. Aquests programes no precisen cap aplicació en el nivell superior, s'executen en el cub i mostren els resultats per consola.

El programa bateria ens retornarà el valor en volts amb una precisió d'un decimal del nivell de bateria llegit pel conversor analògic-digital. La primera lectura d'aquest després de l'engegada del robot serà sempre errònia, això es deu al retard normal que té el conversor per iniciar el seu oscil·lador.

El programa "recte" requereix com a paràmetre d'entrada una consigna en centímetres i farà desplaçar el robot en línia recta fins assolir la consigna. Les consignes poden donar-se tant amb signe positiu com negatiu. La Figura 37 i la Figura 38 representen els resultats obtinguts a partir de diverses proves realitzades.

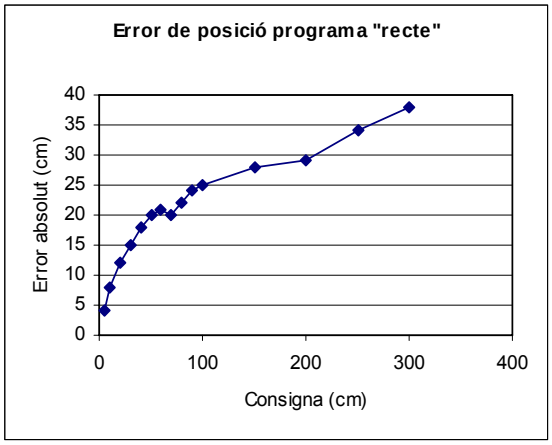

Figura 37. Error absolut programa "recte"

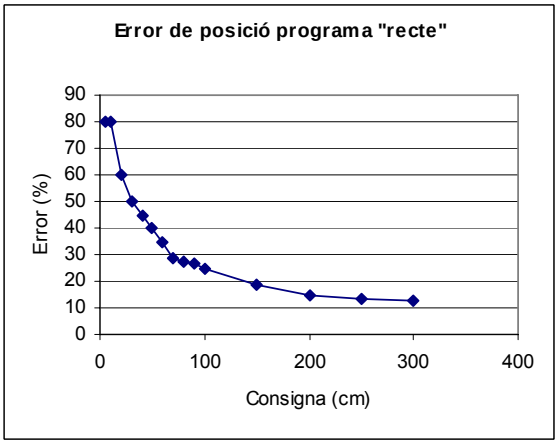

Figura 38. Error relatiu programa "recte"

Es pot observar que sobretot per consignes petites els resultats no són satisfactoris. L'error es deu a diversos motius, per una banda realitzar càlculs d'odometria amb valors de velocitat no és la forma ideal sobretot podent disposar dels valor dels encoders. Per altra banda hi ha un cert error fix que es deu a la inèrcia del robot, donat que davant d'una consigna de velocitat nul·la el robot deixa girar les rodes lliurement aquest es desplaça per inèrcia un cop assolida la consigna. Per consignes molt petites l'error és més elevat donat que les dades de velocitat del robot es mostregen cada 41 milisegons la qual cosa és un temps relativament llarg comparat amb el temps que triga en assolir la consigna.

El programa "gir" requereix com a paràmetre d'entrada una consigna en graus que farà girar el robot sobre el seu propi eix. Les consignes es poden donar amb signe positiu o negatiu fent girar el robot en sentit horari o antihorari respectivament. La Figura 39 i la Figura 40 representen els resultat obtinguts a partir de diverses proves amb aquest programa.

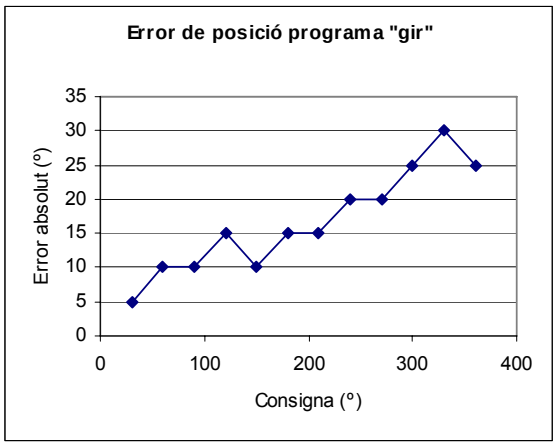

Figura 39. Error absolut programa “gir”

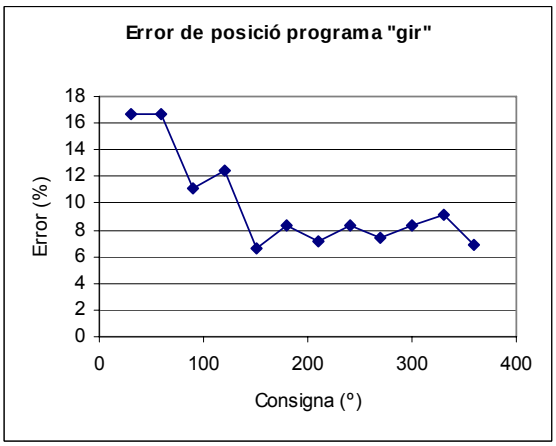

Figura 40. Error relatiu programa “gir”

En aquest cas els resultats tampoc són satisfactoris. L’error es deu als mateixos motius que en el cas anterior tot i que cal afegir que en aquest cas segons la superfície sobre la qual es desplaça el robot fa patinar les rodes i per tant augmentar l’error produint-se fins i tot un error de posició del centre del robot, el qual no hauría de presentar desplaçament.

Totes dues proves descrites anteriorment s’han realitzat amb una consigna de velocitat de 30 sobre 60 (duty cicle 50%) i sense el control PID que du de fàbrica. Els errors augmenten conforme augmenta la velocitat. Cap d’aquestes proves s’han pogut realitzar utilitzant el control PID donat que aquest dona una resposta de sobrepic extremadament elevada donant errors de fins al 300%.

A partir dels resultats obtinguts queda clar que no es pot realitzar un control de posició per odometria sense modificar la programació dels dsPIC’s. Si en aquest nivell de programació es disposés de dades sobre les lectures directes dels encoders segurament es podrien millorar notablement els resultats. Tot i que un cert error és inevitable part d’aquest error es podria calcular de forma que no s’acumulés sobre el següent desplaçament.

De totes formes un cop arribat al punt d’haver de reprogramar els dsPIC’s es planteja fer els càlculs de posició directament en el nivell més baix.

4.3. Control des dels dsPIC’s

Aquest és el nivell més baix en el qual es pot programar el Wifibot. Un cop arribat en aquest punt es poden esgotar totes les capacitats del robot donat que implica redissenyar tota la programació del robot.

Arribat a la conclusió de que és necessari programar en aquest nivell es planteja la possibilitat de redissenyar les comunicacions de forma que es pugui fer el control de posició en un nivell superior o de realitzar aquest control directament en aquest nivell reservant els nivells superiors per altres tasques.

Tot i que el control de posició per odometria des dels dsPIC's és viable cal remarcar que no serà mai perfecte. El control de posició per odometria per definició només funciona acceptablement en el cas de que el terreny sigui regular de forma que el robot no patina en cap moment. Donat que no sempre tenim les condicions ideals la odometria no es pot considerar com un sistema de localització fiable quan no obté cap altre suport.

Existeixen altres mètodes de localització com per exemple a través de càmeres o de la detecció de punts coneguts que poden ser complements de la odometria. Aquests altres mètodes sovint són capaços de determinar amb molta precisió la posició però són mètodes lents de forma que no són capaços de refrescar la posició amb prou rapidesa.

Així la nova programació es dissenyarà de tal forma que el control de posició per odometria es pugui complementar amb altres sistemes que es podran implementar en els nivells de programació superiors i interactuar amb la odometria en els dsPIC's.

La Figura 41 mostra de forma gràfica els resultats que s'obtindrien combinant la odometria amb un sistema de localització extern. Quan el robot es posa en marxa per arribar a una posició destí concreta aquest calcula la seva posició per odometria cada cert temps. (per exemple cada 50 ms). A mesura que transcorre el temps augmenta la incertesa sobre la posició real del robot degut a la imprecisió de la odometria. El sistema de localització extern calcularà la posició del robot cada cert temps (molt superior a la freqüència de la odometria) i l'enviarà als dsPIC's de forma que la incertesa es reduirà de cop.

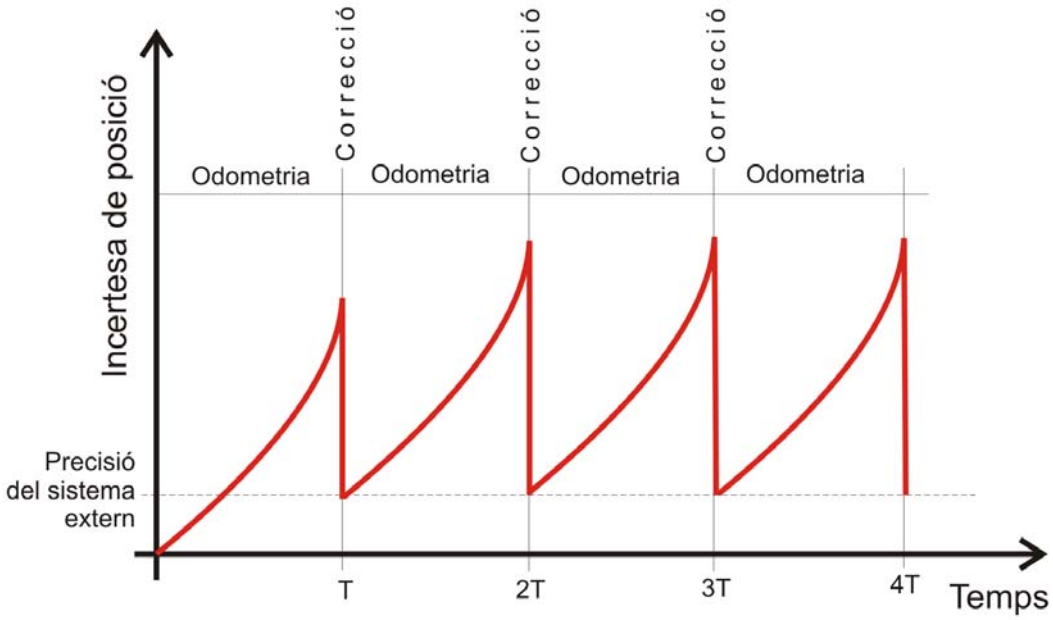

Figura 41. Evolució de la incertesa de posició.

En la gràfica es representa el temps de mostreig de la odometria proper a infinit mentre que el temps de mostreig del sistema extern equival a “T”. Donat que el sistema extern també tindrà una cert nivell d’exactitud que segurament no serà mai del 100% es manté sempre una mínima incertesa sobre la posició.

5. DISSENY DEL CIRCUIT DELS DSPIC'S

Observant en detall el circuit dels dsPIC's del robot es pot veure que tot i que el dsPIC és un dispositiu d'unes prestacions elevades aquestes no s'aprofiten plenament.

Donat que s'ha arribat a la conclusió de que és necessari modificar la programació dels dsPIC's es planteja la possibilitat d'optimitzar prèviament el circuit.

5.1. Justificació del redisseny

Estudiant el circuit original del robot es pot veure que els dsPIC's tenen prou capacitat i prestacions com per que el mode de funcionament es pugui realitzar amb un sol dsPIC. Donat que només es precisen dos mòduls PWM, dues entrades per als sensors infrarojos i dues entrades per als encoders.

Aquest fet planteja dos possibles redissenys del circuit. Per una banda es podria simplificar el circuit amb un sol dsPIC simplificant d'aquesta manera també la programació o es pot mantenir l'estructura de dos dsPIC's i augmentar les prestacions del robot. Finalment s'ha optat per l'opció d'augmentar les prestacions del robot.

A continuació es descriu breument el mode de funcionament original del robot per tal de poder contrastar el avantatges que presenta redissenyar el circuit.

5.1.1. Estructura original del circuit.

En la Figura 42 podem observar un esquema general del circuit dels dsPIC's original del robot. Tot i que el circuit conté dos drivers de motors (LM298) capaços de controlar dos motors cada un aquests es troben connectats en paral·lel desaprofitant la meitat de la capacitat de tota la circuiteria. Aquesta estructura presenta l'inconvenient de no poder controlar la velocitat per separat de les quatre rodes del robot, és a dir, fer un control "4x4".

Els sensors de distància que incorpora el robot es troben tots dos encarats cap a davant, d'aquesta manera es pot evitar que el robot xoqui frontalment amb obstacles però no podem controlar els obstacles que poden haver anant enrera.

Així mateix si detecta un obstacle per mitjà dels sensors i es vol esquivar-lo de manera eficient no es pot donat que en el moment de desviar el robot perd de vista l'obstacle i no sap en quin moment pot tornar a la trajectòria desitjada.

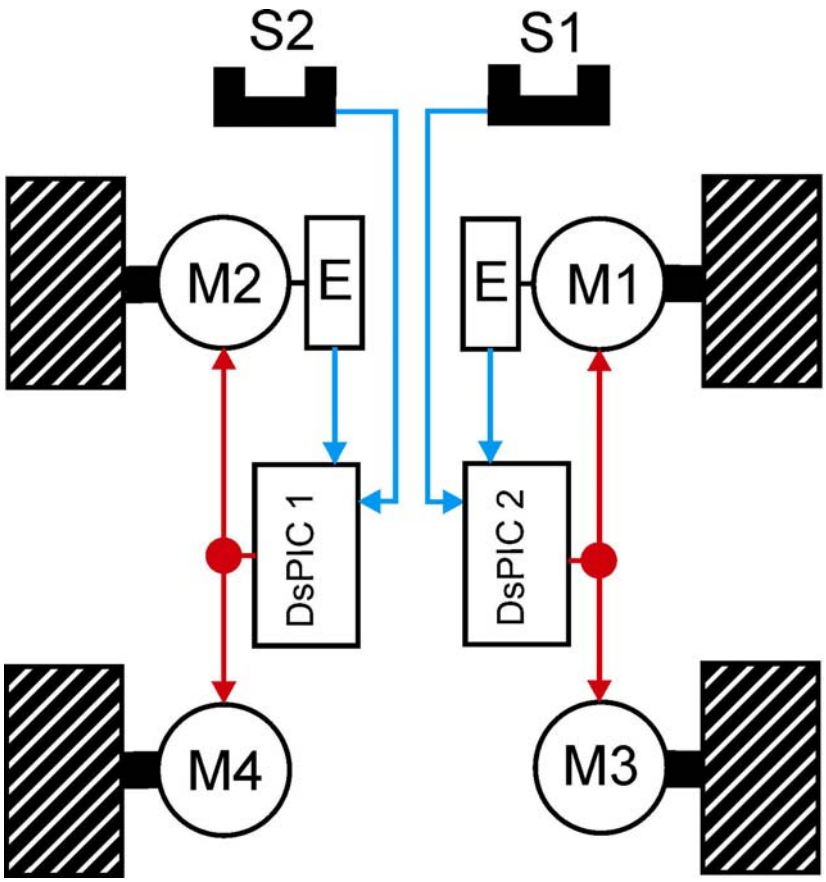

Figura 42. Circuit original

Per altra banda la separació del circuit dividint el robot en la meitat esquerra i la meitat dreta faria impossible controlar el moviment del robot en cas que un dels dos dsPIC's fallés en el seu funcionament.

5.1.2. Estructura del circuit nou

Amb el redisseny del circuit es pretén solucionar els inconvenients presentats anteriorment sense modificar essencialment l'estructura del robot.

El nou circuit dels dsPIC's del robot igualment es forma per dos dsPIC's iguals que els anteriors. La principal diferència es troba en que en aquest cas cada dsPIC controla la part de davant i la part de darrera del robot respectivament. Aquesta nova configuració permet que el robot segueixi operatiu fins i tot si un dels dos dsPIC's no funciona. En tal cas simplement una meitat del robot arrossegaria l'altre.

Els motors estan connectats a sortides PWM independents en cada dsPIC de manera que es pot regular de forma independent la velocitat de les quatre rodes. Aquesta qualitat pot ser especialment útil en terrenys irregulars on es pugui perdre la tracció d'alguna de les rodes.

La Figura 43 mostra de nou l'esquema bàsic del nou circuit. Podem observar que a part de modificar l'estructura s'han afegit nombrosos sensors.

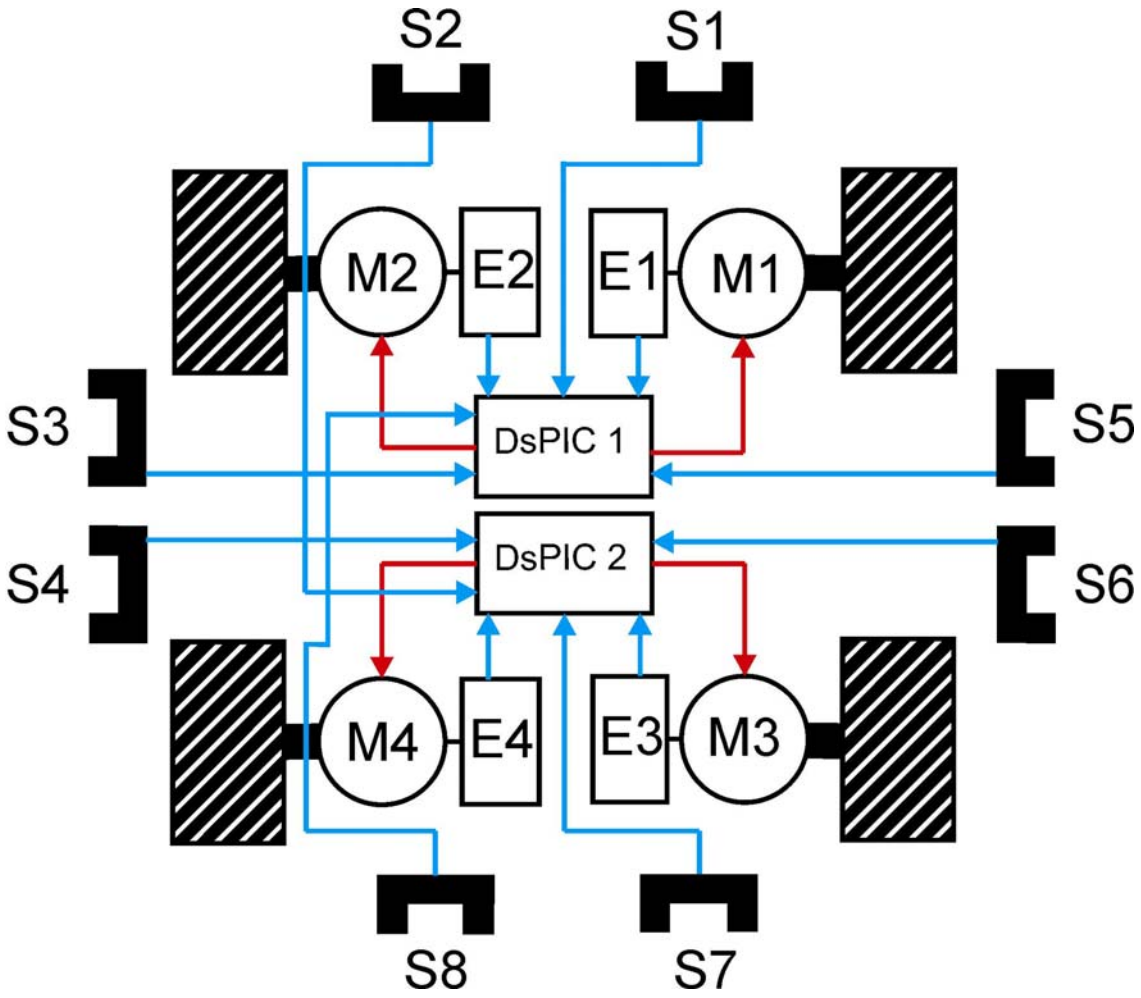

Figura 43. Circuit nou

Per una banda s'han afegit encoders en les rodes de darrera per tal de poder controlar i actuar quan una de les rodes patina o està bloquejada. També s'han afegit nous sensors de distància en el lateral i en la part posterior del robot. Aquests sensors no només permetran detectar obstacles situats a tot el voltant del robot sinó que serviran per poder sortejar els obstacles cosa que anteriorment no era possible.

Els dos sensors de davant i els dos de darrera estan connectats de forma creuada en els dsPIC's, això constitueix un alt nivell de seguretat donat que en cas de fallada d'un dels dos dsPIC's el robot seguirà sent capaç de detectar obstacles en gairebé tot el perímetre.

Tot i l'augment de les prestacions i dels ports ocupats en cada dsPIC encara queden tres ports d'entrada/sortida (un possible PWM) lliures per futures aplicacions o ampliacions. Per aquests ports es preveurà un connector que inclourà alimentació de forma que es podran connectar nous sensors o actuadors fàcilment.

5.2. Contrast

La Taula 13 resumeix les avantatges més importants que presenta el nou circuit proposat per als dsPIC's així com els inconvenients, tot i que lleus. A partir d'aquest contrast es decideix portar a terme les modificacions abans de començar la programació dels dsPIC's per al control de posició per odometria.

Avantatges	Inconvenients
Funcionament en cas de fallada d'un dels dsPIC's. Augment de la robustesa.	Ja no funcionaran els programes subministrats pel fabricant
Control de velocitat en les quatre rodes.	Augmenta la complexitat del programa
Control de tracció en les quatre rodes	
Possibilitat d'esquivar obstacles sense perdre'ls de vista	

Taula 13. Contrast d'avantatges e inconvenients

6. ENTORN DE PROGRAMACIÓ

Per tal de crear les noves aplicacions per als tres nivells de control es treballarà amb tres entorns de programació diferents. Cada entorn ens permet crear aplicacions que posteriorment podran funcionar en el respectiu nivell.

6.1. Programació dels dsPIC's

Per tal de carregar nous programes en els dsPIC's existeixen diverses alternatives. En primer lloc es pot fer servir un programador específic. Aquesta forma de programar s'ha descartat donat que requereix extreure el dsPIC del robot cada cop que es vol programar, la qual cosa no resultaria pràctica durant el procés de disseny.

Com a segona opció es planteja fer servir un programa "bootloader" que ens permeti carregar els programes a través del port sèrie (UART) del dsPIC tal i com ja s'ha fet en altres ocasions en el departament amb els robots futbolistes. Tot i que aquesta solució seria viable finalment s'ha optat per una solució més versàtil.

L'entorn de programació MPLAB de Microchip permet fer servir diversos dispositius per programar i/o depurar els PIC's i dsPIC's. Un d'aquests és l'ICD2 (In Circuit Debugger), seguidament es descriu de forma detallada el seu funcionament i l'annex B conté un resumit manual sobre com fer-lo servir. Aquest dispositiu permetrà programar i depurar els dsPIC's del robot directament des de l'entorn MPLAB i sense necessitat d'extreure'ls en cap moment.

Es tracta d'un senzill circuit controlat per un PIC 16F876 descrit en el plànol 4. Aquest dispositiu permet programar qualsevol PIC o dsPIC de la casa Microchip actualment en el mercat i donat que es pot actualitzar segurament serà compatible amb nous models del futur.

El circuit dissenyat presenta gairebé les mateixes prestacions que el producte ofertat per la casa Microchip. Tot i que presenta alguna petita limitació les possibles aplicacions son igualment àmplies. L'ús del circuit dissenyat presenta principalment una avantatja econòmica, sobretot tenint en compte que es podrà aplicar en nombrosos projectes del departament.

Un cop dissenyat el circuit, la part més costosa del procés, se'n poden produir rèpliques a un cost molt inferior al producte acabat que ofereix el mercat. Donat que es composa de

pocs i senzills components el cost material és relativament baix. La Figura 44 mostra l'aspecte del prototip fabricat

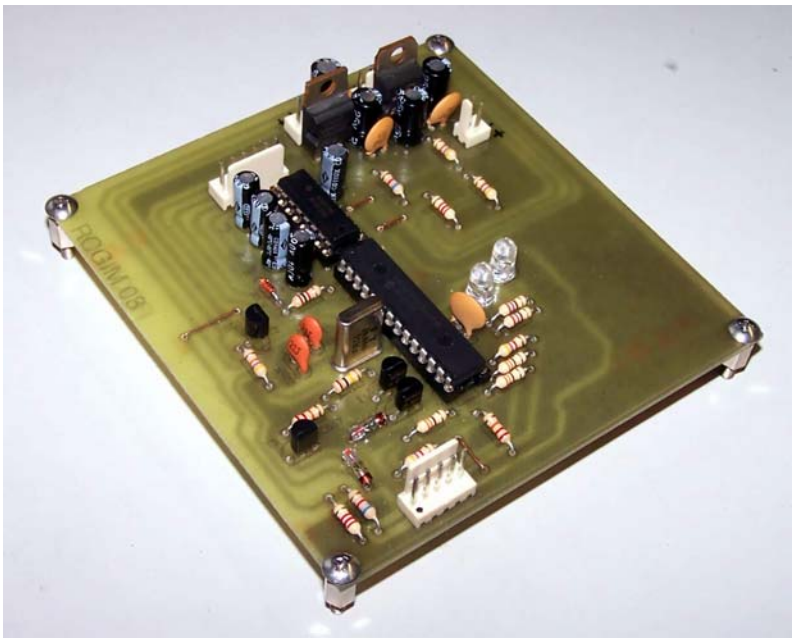

Figura 44. Prototip de l'ICD2

6.1.1. Principi de funcionament de l'ICD2

El PIC 16F876 que controla el ICD2 conté un programa bootloader que permet carregar-li un "sistema operatiu" diferent segons el model de PIC o dsPIC que es vol programar. Aquest bootloader el podem trobar en la mateixa instal·lació de l'entorn MPLAB (...\ruta_instal·lació\Microchip\MPLAB IDE\ICD2\BL010101.hex). Un cop haguem carregat aquest bootloader en el PIC aquest es reconfigurara de forma automàtica a través del port sèrie cada cop que es connecti l'entorn de programació MPLAB IDE. La Figura 45 mostra de forma esquemàtica aquest sistema.

Figura 45. Principi de funcionament ICD2

Cal remarcar que aquest dispositiu necessita un port sèrie natiu en el PC de forma que no funciona correctament amb conversors USB-RS232.

A part de permetre programar el dsPIC aquest dispositiu permet depurar els programes carregats. D'aquesta forma es poden executar els programes pas per pas des de l'entorn

MPLAB i a l'hora observar l'evolució de les variables i dels registres de configuració del dsPIC.

6.1.2. Ports ocupats pel ICD2

Per tal de programar aquest dispositiu fa servir dos ports del dsPIC (PGD i PGC). Aquests dos ports són únics en el dsPIC i només es troben disponibles en una parella de potes. Per altra banda per depurar es fan servir una altra parella de ports (EMUC i EMUD). En el cas del nostre dsPIC aquests ports tenen tres parelles de pins alternatives.

Donat que en el nostre cas aquests ports per programar ja es fan servir per un altre propòsit (bus I^2C) s'haurà d'incorporar un commutador per tal de poder desconnectar el bus I^2C i connectar el programador. En canvi una de les parelles de pins alternatius pel port de depurar està lliure de forma que els podrem connectar directament.

6.1.3. Interfície de programació

Donat que el robot té dos dsPIC's a part de necessitar un sistema per commutar entre programador i bus I^2C necessita un sistema que permeti seleccionar quin dsPIC es vol programar o depurar. El plànol 5 mostra el circuit d'aquesta interfície que ens permet seleccionar amb un selector si es vol programar un o altre dsPIC o connectar el bus I^2C. Per altra banda té un selector per commutar el port de depuració entre un o altre dsPIC. La Figura 46 mostra de forma esquemàtica la funció d'aquesta interfície.

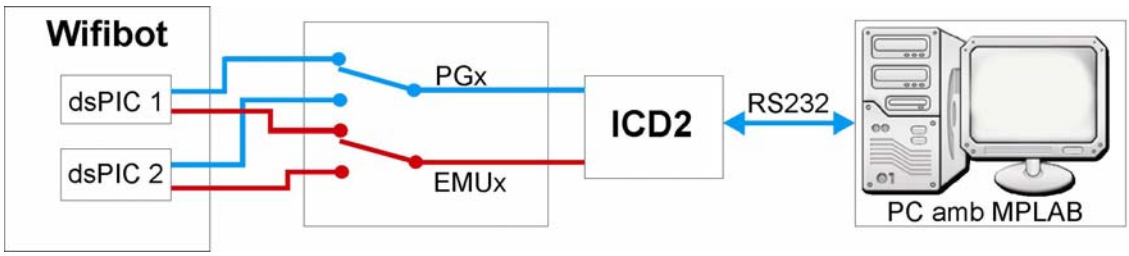

Figura 46. Interfície de programació.

Tot i que es possible programar i depurar pel mateix port per la nostra aplicació s'ha escollit fer servir per depurar un del ports alternatius per tal de poder depurar mentre es troba connectat el bus I^2C.

L'annex B descriu de forma detallada el mode de funcionament del ICD2 en conjunt amb l'entorn MPLAB i el robot.

6.2. Programació del Mesh cube

Com ja s'ha comentat anteriorment la programació del cub requereix treballar en un entorn Linux. De les diverses alternatives que es presenten per això es treballarà des d'una màquina amb Linux (distribució Gentoo). L'annex B descriu com obtenir el compilador per aquest entorn i com crear i compilar les aplicacions.

En un sistema Linux la compilació d'un codi es regeix per les directrius senyalades en l'anomenat "Makefile". En aquest arxiu es defineixen les directrius de compilació així com altres comandes. Per al cas de les aplicacions creades per al robot el "Makefile" conté la definició d'una instrucció que ens permet compilar el codi i enviar-lo automàticament al robot en un sol pas. La Taula 14 descriu les comandes definides per tots els "Makefiles" creats per les aplicacions del robot.

Comanda	Descripció
Make nom_codi.c	Únicament compila el codi creat
Make clean	Borra els arxius intermitjos que genera la compilació
Make send_wifi	Envia el codi compilat (ha de existir prèviament) al robot per wifi
Make send_LAN	Envia el codi compilat (ha de existir prèviament) al robot per ethernet LAN
Make all_wifi	Compila el codi, envia l'executable al robot per wifi i borra els arxius intermitjos de la compilació
Make all_LAN	Compila el codi, envia l'executable al robot per ethernet LAN i borra els arxius intermitjos de la compilació

Taula 14. Comandes definides per compilar

L'ús d'aquests arxius de configuració per a la compilació no només presenta l'avantatge de poder definir diferents accions amb un nom de comanda fàcil de recordar. Quan el codi de l'aplicació és llarg i es composa per diversos documents de codi la compilació pot allargar-se considerablement. Linux gestiona la compilació a través dels "Makefiles" de forma que només s'executaran aquelles accions que suposin una diferència en l'aplicació creada. Molt sovint modificacions en el codi no requereixen que aquest es recompili completament sinó únicament la part modificada.

6.3. Programació externa

Per últim queda definir l'entorn per crear aplicacions externes al robot. En aquest cas ja no es pot definir de forma tan clara quin entorn és millor o més adient. Per al present projecte s'ha escollit l'entorn Lab View.

L'entorn Lab View permet crear aplicacions amb interfície gràfica de forma senzilla i ràpida. Per al tipus d'aplicació que s'ha creat per aquest projecte ha resultat ser l'entorn de programació més pràctic.

Per a la creació d'altres aplicacions en el futur com podrien ser aplicacions capaces de pendre decisions possiblement resulti més pràctic fer servir altres aplicacions o entorns.

Com ja s'ha comentat anteriorment les aplicacions per controlar el robot des de l'exterior no necessàriament necessiten estar ubicades en un PC. L'annex A presenta un senzill programa que permet controlar les funcions bàsiques del robot des d'una consola de joc Nintendo DS o Nintendo DS Lite.

Aquest tipus de consoles de joc presenten una alternativa atractiva per controlar el robot donat que es tracta de dispositius portables amb comunicacions wifi incorporades.

Així mateix la Nintendo DS no és l'únic dispositiu que ens dóna aquestes possibilitats. El propi fabricant del robot ofereix via internet programes capaços de controlar el robot des d'una PDA i des d'una consola PSP (Play Station Portable). Òbviament aquestes aplicacions no funcionen sense ser modificades amb el robot Wifibot modificat.

7. PROGRAMA DSPIC'S

Dins la programació dels dsPIC's s'ha de distingir clarament el programa que regeix el dsPIC de les dues rodes davanteres i el dsPIC que regeix les rodes de darrera. Donada la modificació del circuit i l'implementació de noves funcions aquests programes ja no son iguals tal i com es presentava en la programació del robot original.

7.1. Consignes i configuracions a les que responen els dsPIC's

En aquest apartat es descriu de forma general el tipus de consignes i configuracions a les quals responen els programes del dsPIC's. El format específic d'aquestes configuracions i consignes es descriuen en el subapartat de protocol de comunicació I^2C d'aquest tema.

La varietat de configuracions i consignes a les quals respon cada dsPIC s'ha pensat de forma que el control del robot final resulti el més versàtil possible i esgoti totes les capacitats d'aquest nivell de control.

7.1.1. Configuracions

Com a configuracions s'entén la forma d'actuar del robot davant determinades situacions i el tipus de consignes a les quals respondrà.

Configuració dels PWM's: El mode de treball dels PWM es pot configurar de dues formes diferents amb les opcions de gir lliure. Activant l'opció del gir lliure la senyal de PWM només s'aplica a un dels terminals dels motors mentre l'altre es manté a massa d'aquesta forma davant una consigna de velocitat nul·la les rodes giraran de forma lliure. Aquesta configuració presenta l'avantatge de consumir menys energia però per contra el robot podria desplaçar-se involuntàriament sobre un terreny amb pendent. La Figura 47 mostra un exemple del cicle de treball del PWM per aquesta configuració per diferents velocitats, s'entén que "T" equival al període del PWM.

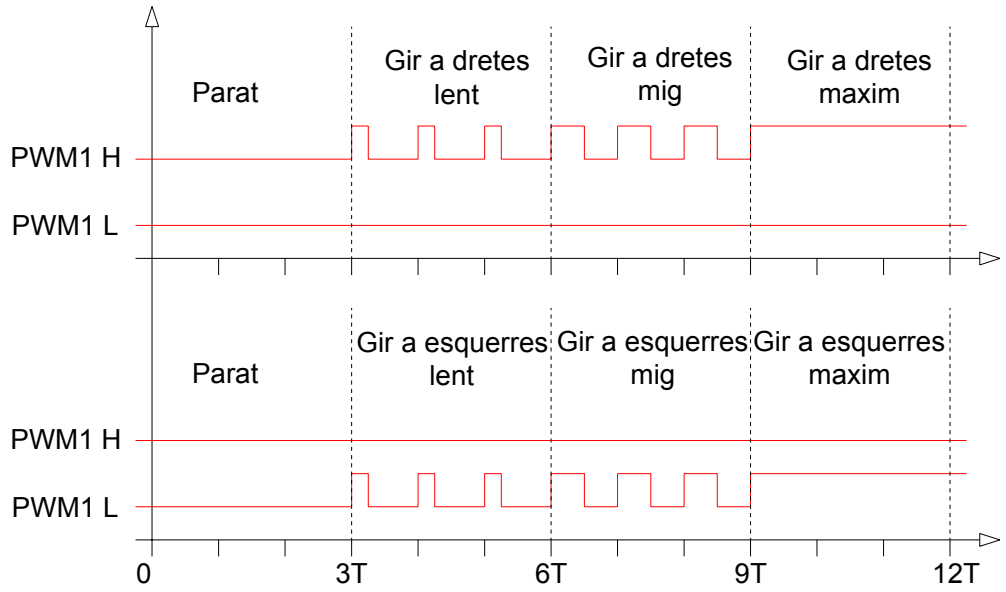

Figura 47. Cicles de PWM amb l'opció de gir lliure activada

Per altra banda si desactivem la opció de gir lliure davant una consigna de velocitat nul·la s'aplicarà en cada terminal del motor una senyal de PWM inversa, així amb un cicle de treball del 50% els motors estaran parats. Aquesta configuració permet bloquejar fins un cert punt les rodes evitant desplaçaments involuntaris. La Figura 48 mostra l'evolució dels cicles de treball i els corresponents sentits de gir resultants.

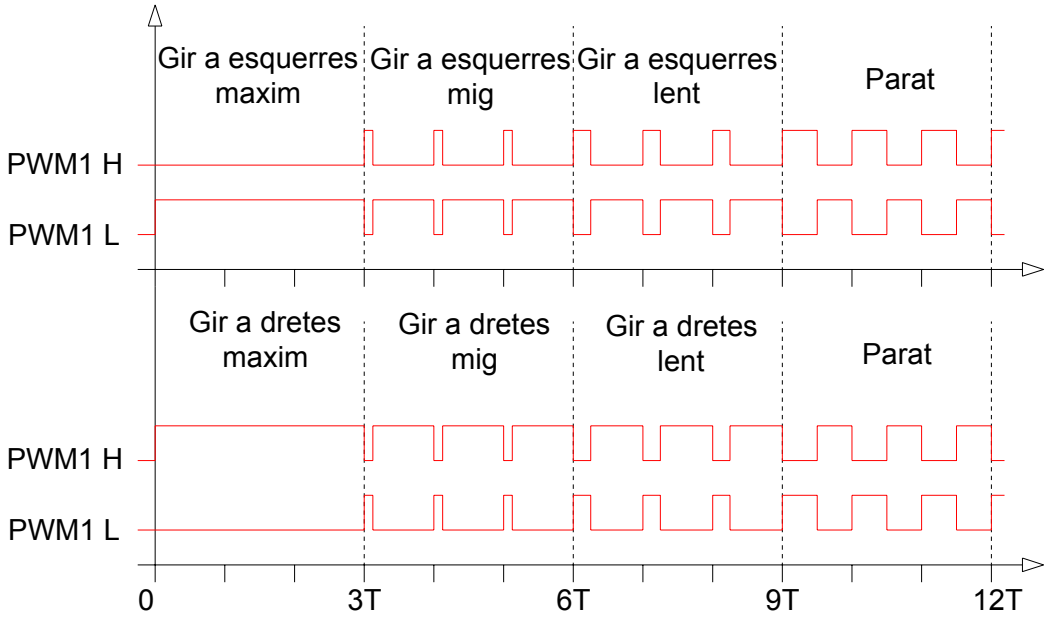

Figura 48. Cicles de PWM amb l'opció de gir lliure desactivada

Control d'obstacles: Mitjançant els 8 sensors de distància el robot pot detectar obstacles que es troben al seu voltant. Si s'activa el control d'obstacles el robot es pararà de forma automàtica davant el perill de generar-se una col·lisió. Un cop arribat en aquest punt

l'usuari o el sistema de control de nivell superior haurà de desactivar aquesta opció per continuar la marxa i sortejar l'obstacle.

Control de velocitat: Les consignes de velocitat poden ser interpretades pel robot de dues formes diferents. Si es desactiva el control de velocitat les consignes s'aplicaran de forma directa al cicle de treball del PWM. D'altra forma, si s'activa el control de velocitat les consignes seran interpretades com consignes de velocitat en cm/s.

Control de posició: Si es treballa en aquest mode el robot es desplaçarà automàticament a la posició indicada per la consigna de posició. Aquesta consigna s'haurà de donar referenciada a la posició inicial del robot sobre els eixos X i Y. Opcionalment es podrà donar consigna per l'angle final.

7.1.2. Consignes que es poden enviar al robot

Les diverses consignes que es poden enviar al robot s'interpretaran d'una o altra forma segons la configuració, igualment determinades dades simplement no es tenen en compte segons la configuració.

Consignes de velocitat: Es pot enviar una consigna de forma independent per cada una de les rodes. Si es treballa en mode de control de posició només es té en compte la primera consigna que s'aplicarà a totes les rodes.

Consignes de posició: Aquesta consigna s'ha d'enviar de forma que no es superin els límits màxims. Aquesta consigna sempre s'ha de referenciar a l'origen de coordenades extern al robot.

Correcció de posició: Si es disposa d'algun sistema extern que determina la posició real del robot aquesta es podrà enviar al robot per tal que substitueixi els valors calculats per l'odometria.

Ports auxiliars: Els tres ports de cada dsPIC que queden lliures es poden configurar segons les necessitats com a entrades o com a sortides des de l'exterior. En cas de configurar-se com a sortides se'n pot determinar també l'estat.

7.1.3. Dades que ens retorna el robot

Un cop s'han enviat les consignes i configuracions al robot aquest retornarà una sèrie de dades que donen informació sobre l'estat del robot.

Lectures dels encoders: Segons la configuració les lectures dels encoders s'enviaran en valors absoluts, és a dir, en tics comptats des de l'última consulta o en valors de velocitat (centímetres per segon)

Posició del robot: Es tracta de la posició sobre el pla XY on es troba el robot segons els càlculs d'odometria. A més es dóna l'angle de rotació respecte l'eix X. La Figura 49 mostra com s'han d'entendre aquestes dades.

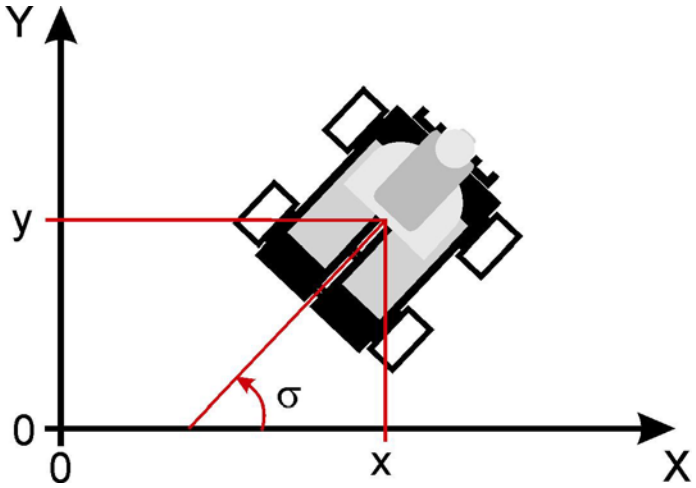

Figura 49. Posició del robot

Lectures dels sensors: Equival a la distància mesurada de cada un dels vuit sensors de distància expressada en centímetres. A tenir en compte que les lectures excessivament baixes poden ser ambigües donada la corba de resposta dels sensors.

Estat dels ports auxiliars: Retorna l'actual configuració d'aquests ports i si escau l'estat en que es troben.

Errors: Cada dsPIC retorna un byte que conté de forma codificada l'estat de funcionament del robot. Si s'ha produït alguna falla aquesta es comunica mitjançant un codi únic. Segons la gravetat de l'error el robot seguirà operatiu o es pararà de forma prudencial fins que es corregeixi l'error.

7.2. Programa del dsPIC de darrera

Dels programes dels dos dsPIC's aquest és el més senzill donat que els càlculs d'odometria i control de velocitat es realitzen en el dsPIC de davant. D'aquesta manera el dsPIC de darrera obeeix segons les configuracions a les ordres rebudes des del dsPIC de davant o les rebudes des del cub.

La Figura 50 mostra un esquema generalitzat de l'execució del programa. Tot i que podem diferenciar diverses parts dins del programa mes o menys independents l'execució del fil principal del programa es regeix per les comunicacions I^2C.

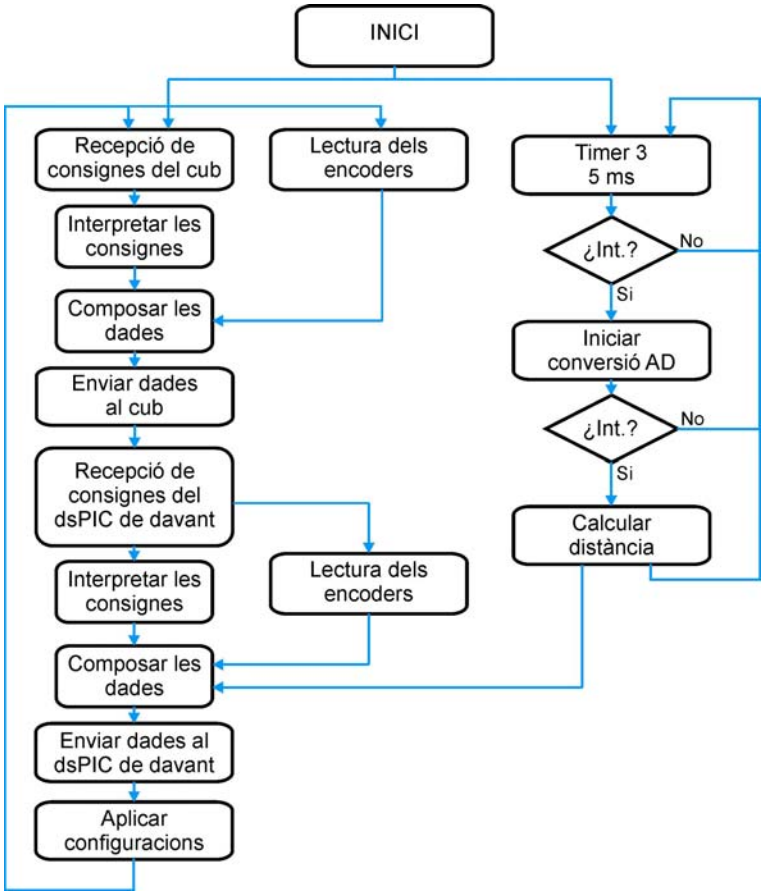

Figura 50. Esquema del programa del dsPIC de darrera.

Donat que en les comunicacions hi intervenen tots dos dsPIC's aquestes es descriuen al final d'aquest apartat.

7.2.1. Lectures dels conversors AD

Tot i que les lectures es poden fer de forma continuada en aquest cas es regeixen per les interrupcions generades pel timer 3. Els 5 ms de període d'aquest timer donen al dsPIC suficient temps entre les lectures per tal de fer els càlculs que converteixen les lectures de volts a centímetres. Aquests càlculs inclouen la linealització de la lectura.

La linealització de les lectures es realitza per aproximació a una equació exponencial. Aquest mètode impedeix poder fer lectures consecutives donat que aquest tipus de càlculs requereixen uns 2 ms de temps. La Figura 51 mostra el gràfic i l'equació aproximada de la distància en funció de la lectura feta servir en els càlculs.

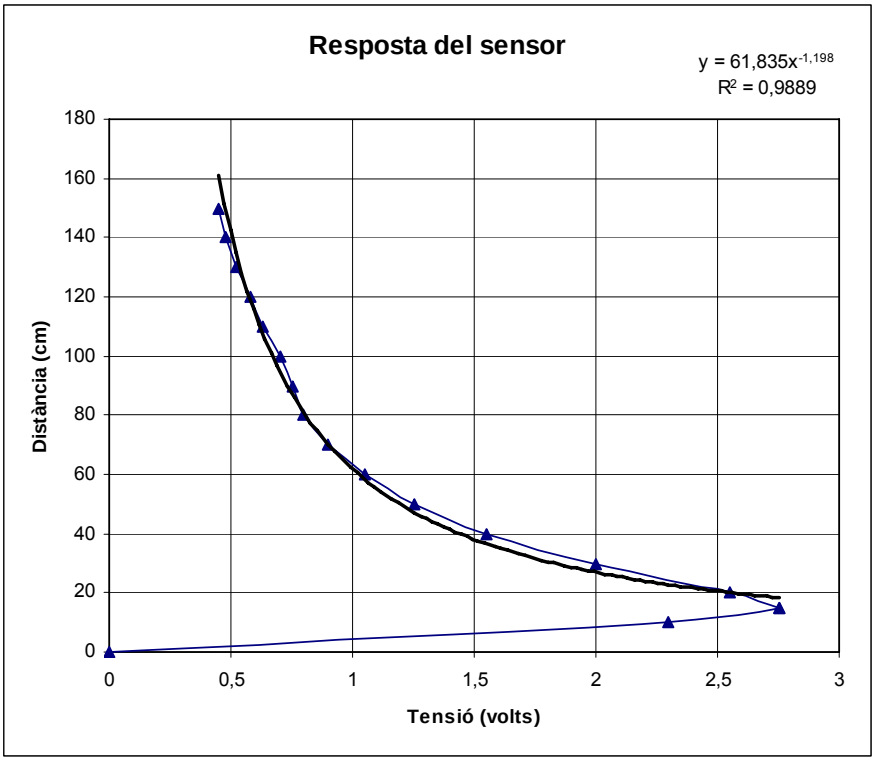

Figura 51. Equació de linealització

El conversor AD del dsPIC està configurat de forma que abans no generi una interrupció aquest realitza quatre conversions. D'aquesta manera en cada interrupció es fa la mitja de 4 lectures proporcionant un resultat més estable i atenuant els sorolls i errors de lectura.

Les lectures realitzades serveixen per una panda per ser enviades al cub i per altra banda permeten controlar l'existència d'obstacles en el camí del robot. Si s'activa aquesta opció el robot es pararà de forma automàtica per tal d'evitar xocar amb obstacles. Més endavant es podrà fer que el robot eviti l'obstacle seguint una trajectòria.

7.2.2. Lectures dels encoders.

Les lectures dels encoders es realitzen un cop iniciades les comunicacions. El resultat de la lectura també té diverses aplicacions. Per una banda s'envien al cub on es podran aprofitar per futures aplicacions. Per altra banda aquestes lectures s'envien al dsPIC de davant on serviran per als càlculs de control de posició i per als càlculs de control de velocitat PID.

Les unitats d'aquestes lectures es mantenen en valors absoluts de polsos llegits des de l'última lectura. Treballar amb aquesta unitat (tic) permet guanyar precisió donat que no es perden decimals durant els càlculs de conversió. En tot cas mitjançant el bit de

configuració dels encoders es pot escollir si les lectures s'enviaran al cub de forma directa o si es transformen en valors de velocitat abans de ser enviades.

7.3. Programa del dsPIC de davant.

Aquest programa evidentment realitza les mateixes operacions de lectura dels conversors AD i dels encoders que el del dsPIC de darrera. A part d'aquestes funcions en aquest cas s'incorporen els càlculs d'odometria, càlcul de trajectòria i control de velocitat PID.

La forma de llegir els valors dels sensors de distància i dels encoders es pot entendre igual que en el descrit en l'apartat anterior de forma que no es tornarà a incidir.

La propera figura mostra de forma esquemàtica l'execució del programa per al dsPIC de davant.

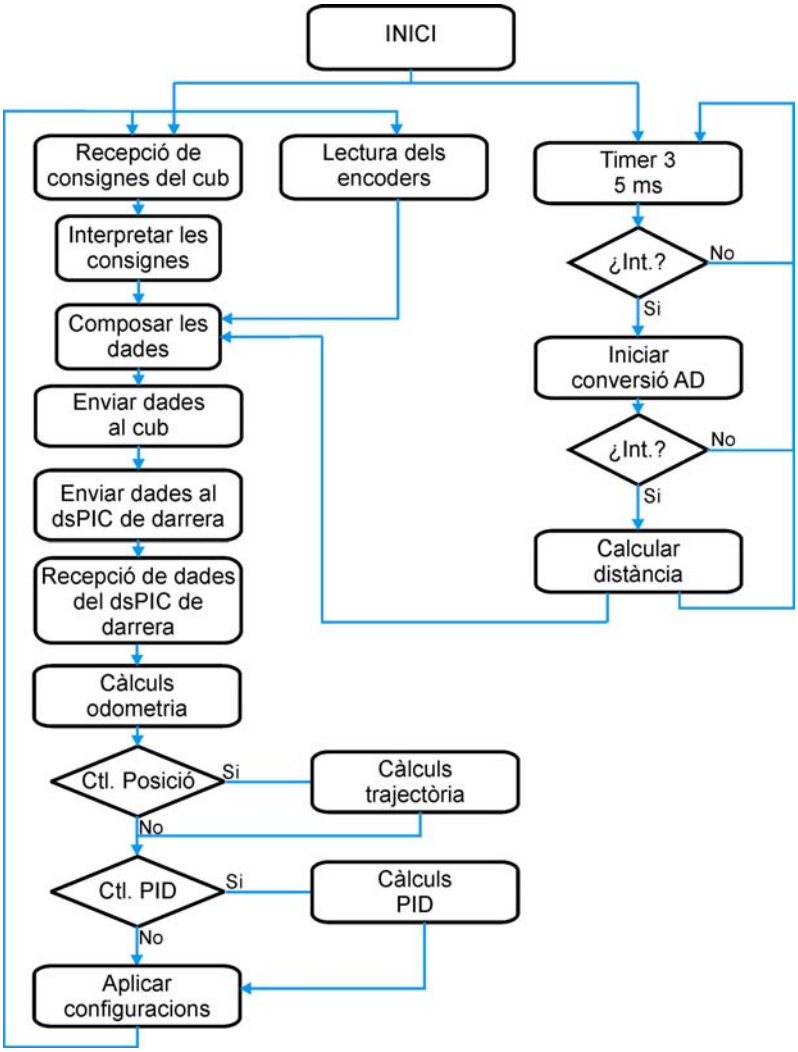

Figura 52. Esquema del programa del dsPIC de davant

7.3.1. Càlculs d'odometria

Els càlculs d'odometria permeten determinar la posició en cada instant del robot a partir de les lectures dels encoders. Aquests càlculs es descriuen de forma detallada en l'annex C. De forma general es pot dir que la posició del robot en cada instant es calcula a partir de la posició en que es trobava en l'instant anterior més l'increment de posició.

Tot i que el programa envia cap a l'exterior les dades de posició en unitats normalitzades, és a dir, posició en l'eix X i l'eix Y en metres i orientació en graus, internament no es treballa amb aquestes unitats. Per tal d'estalviar potència de càlcul i sobretot en càlcul amb fraccionaris, internament es pren com a unitat bàsica el "tic" dels encoders. Aquesta unitat té l'avantatge de no generar decimals de manera que es minimitzen els errors generats per arrodoniments.

Donat que el càlcul de increment de posició es fa a intervals molt curts els errors per arrodoniments s'acumulen de forma que poden ser molt importants al cap de poc temps.

7.3.2. Control de velocitat PID

El bit de configuració del control de velocitat permet donar al robot consignes de velocitat en centímetres per segon.

Els càlculs del PID els realitza el dsPIC que controla la part davantera del robot. Si s'activa el control el dsPIC de darrera no tindrà en compte les consignes de velocitat rebudes des del cub sinó que atendrà a les consignes rebudes des del dsPIC de davant. La Figura 53 mostra el concepte d'aquest control PID.

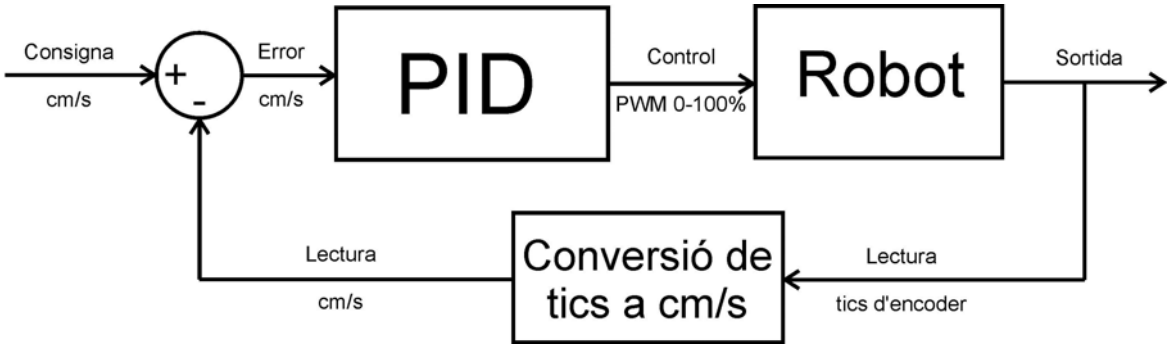

Figura 53. Control de velocitat PID

Per tal de sintonitzar el PID s'ha llegit la resposta del robot davant una consigna de graó. La resposta s'ha capturat amb l'entorn de control creat amb Lab View, aquest permet fer

una captura de les dades llegides pels encoders i emmagatzemar-les juntament amb un vector de temps en un fitxer.

Mitjançant un script de Matlab (alfaro.m, Daniel Caballero Parga) s'han obtingut uns valors inicials per als factors del controlador PID. Aquest Script calcula un model aproximat a partir de la resposta llegida i en dissenya un controlador. En el gràfic s'observen les dades fetes servir per l'aproximació.

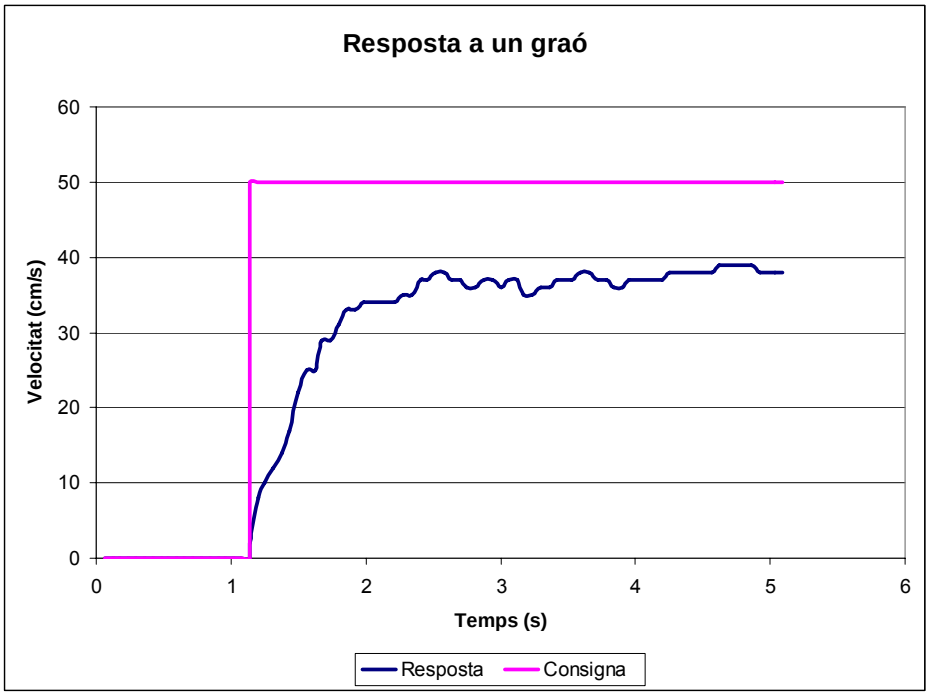

Figura 54. Resposta a un graó

La Taula 15 mostra els resultats inicials per al controlador PID obtinguts amb Matlab.

K_p	0,3097
K_i	1,6180
K_d	0,0746

Taula 15. Valors inicials per PID

A partir d'aquests primers valors per al PID s'han reajustat experimentalment fins a obtenir una resposta adient. S'ha considerat una resposta adient aquella que donada una consigna l'assoleix el més ràpid possible sense provocar un sobrepic excessiu i sense fer patinar el robot per una acceleració excessiva. S'ha estipulat que el sobrepic no ha de superar el 10% per evitar grans oscilacions. Experimentalment s'ha comprovat que per una consigna de graó de 50 cm/s (com en la gràfica anterior) el temps d'establiment ha de ser de com a mínim un segon per evitar que sobre una superfície llisa el robot patini.

La Taula 16 mostra els factors ajustats per al PID de forma que es compleixen les expectatives.

K_p	0,3
K_i	3
K_d	0,9

Taula 16. Valors finals per al PID

La resposta obtinguda amb el controlador dissenyat es pot observar en la Figura 55, com a referència s'ha inclòs en el gràfic la resposta sense controlador. Com es pot observar es cumpleixen els objectius d'un sobrepic màxim del 10% i temps d'establiment superior a un segon.

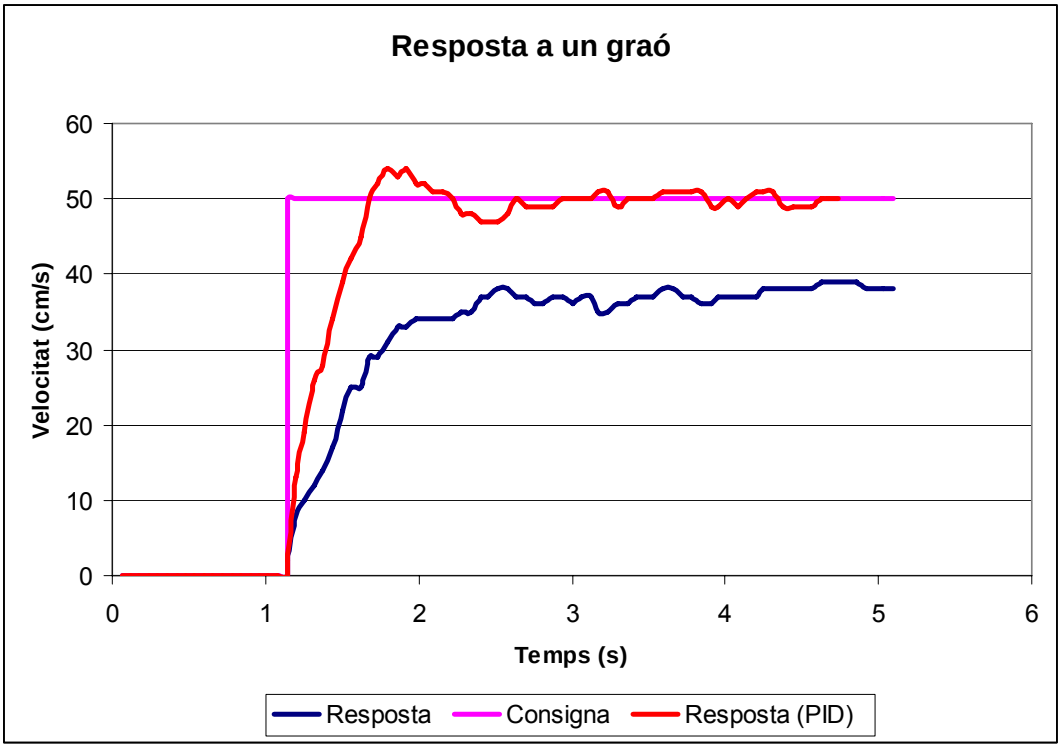

Figura 55. Resposta del controlador PID

7.4. Protocol de comunicació I^2C

La nova arquitectura del Wifibot a baix nivell crea la necessitat de modificar l'estructura del bus I^2C original. L'estructura original es basava en que només hi ha un mestre (el processador del cub). Aquesta estructura té l'avantatge que mai no es generen col·lisions de dades en el bus però te l'inconvenient que dos esclaus del bus no poden comunicar-se entre ells. La nova estructura contempla que hi pugui haver més d'un mestre en el bus.

Tot i que el protocol estàndard I^2C preveu la possibilitat de que es generin col·lisions de dades en un bus multimestre, i per tant preveu la forma de gestionar-les, en el nostre cas es pretén evitar que es pugui donar la possibilitat. Donat que volem una comunicació continua i regular entre els components del bus les col·lisions de dades s'han d'evitar per tal de no perdre dades a causa de retards en la comunicació.

La nova estructura es basa en un ordre de comunicacions seqüencial entre els elements de manera que mai no intentaran escriure o llegir del bus dos elements a la vegada. La Taula 17 descriu quins elements són mestre i quins són esclau així com l'ordre de les comunicacions i la quantitat de bytes transmesos.

Pas	Mestre	Acció	Esclau	Acció	bytes
1	Mesh cube	Escriptura mestre	Conversor AD	Lectura esclau	2
2	Mesh cube	Lectura mestre	Conversor AD	Escriptura esclau	1
3	Mesh cube	Escriptura mestre	dsPIC darrera	Lectura esclau	4
4	Mesh cube	Lectura mestre	dsPIC darrera	Escriptura esclau	8
5	Mesh cube	Escriptura mestre	dsPIC davant	Lectura esclau	14
6	Mesh cube	Lectura mestre	dsPIC davant	Escriptura esclau	13
7	dsPIC davant	Escriptura mestre	dsPIC darrera	Lectura esclau	3
8	dsPIC davant	Lectura mestre	dsPIC darrera	Escriptura esclau	3

Taula 17. Comunicacions I^2C

Aquest sistema evita que es generin col·lisions en el bus sempre i quan entre comunicació i comunicació el programa del cub deixi transcórrer suficient temps com per que els dsPIC's es comuniquin entre ells.

La seqüència de comunicació descrita en la Taula 17 s'inicia amb la primera lectura en el conversor AD, i tots els passos següents s'inicien en el moment en que es finalitzi l'anterior. La Figura 56 mostra de forma esquemàtica les comunicacions.

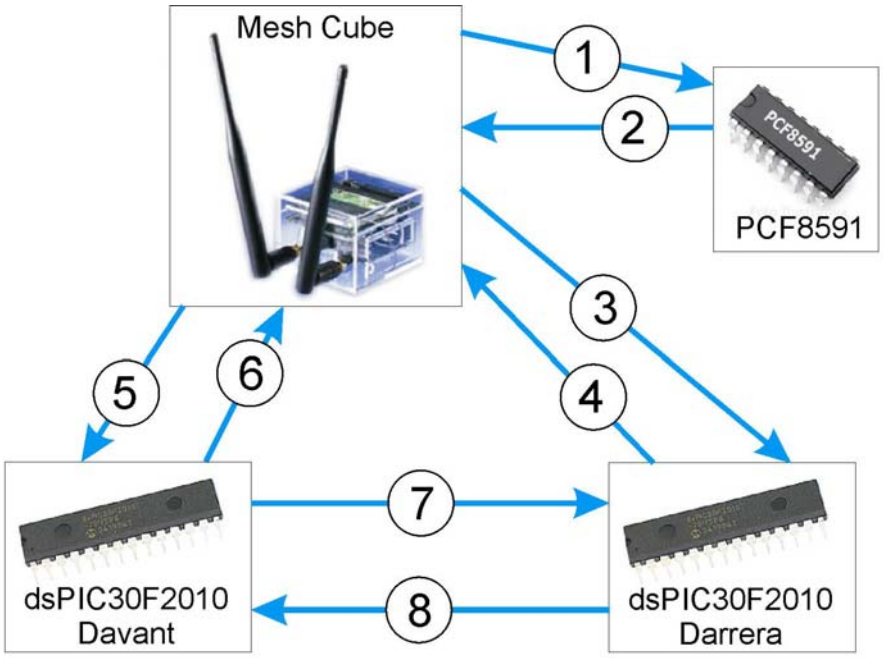

Figura 56. Esquema de comunicacions I^2C

D'aquesta manera la freqüència amb la que es repeteix la comunicació la controla el programa del cub. Aquesta haurà de ser de com a mínim cada 150ms i com a màxim cada 100ms, si es superen els 150ms els dsPIC's emetràn un error interpretant que s'ha tallat la comunicació en canvi un temps inferior als 100ms no dóna temps suficient per acabar el cicle de comunicacions. D'aquesta manera es pot detectar un error en les comunicacions i actuar de forma adient per tal d'evitar que el robot segueixi executant l'ultima comanda rebuda sense control.

A continuació es descriuran les dades que es transmeten en cada un dels passos de comunicació.

7.4.1. Pas 1. Configuració del conversor AD

El Mesh Cube actuant com a mestre enviarà un byte de configuració al conversor AD. Aquest byte sempre serà el mateix donat que per la nostra aplicació no es requereixen diferents configuracions. La Figura 57 mostra el significat de cada bit enviat i ens mostra com queda la configuració per al nostre cas.

bit: 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
0	EN_out	mode:1	mode:0	0	auto inc.	canal:1	canal:0

bit: 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
0	0	0	0	0	0	0	0

Figura 57. Configuració del conversor A/D

7.4.2. Pas 2. Lectura del Conversor AD

El Mesh Cube actuant com a mestre llegirà el valor del conversor AD per tal d'obtenir l'estat de les bateries. En aquest cas només es rebrà un sol byte (Figura 58). Aquest valor caldrà tractar-lo per tal de convertir-lo a un valor en volts.

bit: 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Lectura conversor AD (0-255)							

Figura 58. Lectura del conversor AD

En la Figura 59 es veu que la lectura del nivell de bateria es realitza a través d'un divisor de tensió donat que l'entrada del conversor no pot ser superior a 5 volts.

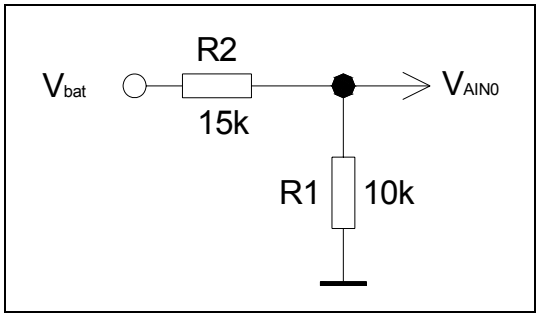

Figura 59. Divisor de tensió per llegir l'estat de les bateries

D'aquí s'obté l'equació 2 que ens permet convertir el valor del conversor AD en volts de la bateria.

$$V_{bat} = V_{AIN0} \bullet 2{,}5 \qquad \text{(Eq. 2)}$$

7.4.3. Pas 3. Enviar consignes al dsPIC de darrera

Les dades enviades al dsPIC de darrera des del Mesh Cube depenen del primer byte enviat que definirà la configuració. La Figura 60 mostra el significat de cada bit del byte de configuració.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	M_enc	Gir_ll	Err_r	Ctl_obs	Ctl_pos	Ctl_vel	Ident.

Ident	Identificació del remitent
	0: Mesh Cube
	1: dsPic davant
Ctl_vel	Control de velocitat PID
	0: Desactivat
	1: Activat
Ctl_pos	Control de posició
	0: Desactivat
	1: Activat
Ctl_obs	Control d'obstacles
	0: Desactivat
	1: Activat
Gir_ll	Mode de treball del PWM
	0: Gir lliure desactivat
	1: Gir lliure activat
Err_r	Reset dels errors
	0: Sense efecte
	1: Borrarà els errors en cas de haver estat solventats
M_enc	Mode de retornar les lectures dels encoders
	0: Valors absoluts
	1: Valors de velocitat (cm/s)

Figura 60. Byte de configuració dsPIC de darrera.

El bit d'identificació del remitent permet al dsPIC saber si les dades rebudes són consignes del Mesh Cube o si són consignes del dsPIC de davant.

El bit de control de velocitat permet activar el control de velocitat per PID que s'aplicarà de forma independent per cada roda. El bit de control de posició defineix si s'enviaran consignes de posició. Aquestes consignes s'enviaran al dsPIC de davant.

El bit de control d'obstacles permet evitar que el robot xoqui amb obstacles detectats pels sensors infrarojos. Un cop es detecti un obstacle el robot es pararà i s'haurà de desactivar aquest bit des del nivell de control superior per tal de sortejar l'obstacle. El robot interpreta que es troba un obstacle quan el detecta amb els dos sensors de davant

o de darrera segons el sentit de la marxa. Un obstacle detectat pels quatre sensors laterals no activarà aquesta funció donat que es suposa que no interfereixen en la trajectòria.

Per defecte davant una consigna de velocitat nul·la els motors es frenaran mitjançant un cicle de PWM del 50%. Això permet una frenada més ràpida i evitar que en un terreny amb pendent el robot rodi pel seu propi pes. Donat que aquest sistema consumeix considerablement les bateries es pot forçar el gir lliure de les rodes mitjançant el bit de gir lliure.

El bit de configuració de les lectures dels encoders defineix en quin format es retornaran les lectures dels encoders, aquestes es poden retornar en valors absoluts respecte l'última lectura o es poden retornar en valors de velocitat.

Per últim el bit Err_r resetejarà els codis de error en cas de que la causa de l'error estigui solucionada.

Seguidament s'enviaran dos bytes amb les consignes de velocitat per cada costat i el sentit de gir. La Figura 61 mostra la configuració d'aquests bytes. En cas de haver enviat la configuració activant el bit de control de posició o de velocitat aquests dos bytes no es tindran en compte (tot i que cal enviar-los) i el dsPIC de darrera rebrà les consignes des del dsPIC de davant.

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_E	Con_E						

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_D	Con_D						

Con_X Consigna de velocitat per al motor esquerra
Si C_vel = 0
0 – 100% respecte a la velocitat màxima
Si C_vel = 1
0 – 128 cm/s

Sentit_X Sentit de gir dels motors
0: Enrera
1: Endavant

Figura 61. Consignes de velocitat

El significat de les consignes de velocitat depèn de si el control PID es troba activat o desactivat. En cas d'estar activat les consignes s'envien en cm/s i en cas contrari s'enviaran en tant per cent, aquest percentatge es traduirà directament sobre el "duty

cycle" dels PWM's que controlen cada un dels motors. Els sentits de gir es referiran sobre el sentit de moviment del robot i no sobre el sentit de gir absolut dels motors.

Amb l'últim byte es pot actuar sobre els ports del dsPIC lliures. La Figura 62 mostra el significat de cada un dels bits per a la configuració.

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

TRISE_X Sentit de les dades del port RE_X
0: Sortida
1: Entrada
RE_X Valor del port de sortida
Si TRISE_X = 0:
0: Sortida desactivada ('0' lògic)
1: Sortida activada ('1' lògic)
Si TRISE_X = 1:
x: No tenen efecte

Figura 62. Configuració dels ports auxiliars

7.4.4. Pas 4. Recepció de dades del dsPIC de darrera

El primer byte rebut serà el byte d'estat que ens informarà sobre qualsevol error que s'hagi produït. La Figura 63 ens mostra la composició d'aquest byte.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_des	Err_cons	Err_com	Cod_err				

Err_des Ens indica que algun valor de posició hi ha hagut desbordament
0: No hi ha desbordament
1: Hi ha desbordament. El robot està parat
Err_cons Alguna de les consignes s'ha enviat fora de rang
0: Consignes correctes
1: Consignes errònies. El robot està parat
Err_com Error de comunicació amb l'altre dsPIC
0: La connexió es correcte
1: Hi ha hagut fallada de comunicació. El robot funciona igualment
Cod_err Codi d'errors

Figura 63. Codis d'error

La Taula 18 descriu el significat de cada un dels codis de error. Donat que es poden generar més d'un error a la vegada el codi només reflectirà l'últim error detectat. Aquests

missatges es mantindran fins que s'hagin solucionat i s'hagi enviat el bit per resetejar els errors.

Codi	Significat
00	No hi ha errors
01	Desbordament de la posició en l'eix X. La posició calculada del robot surt dels rangs màxims
02	Desbordament de la posició en l'eix Y. La posició calculada del robot surt dels rangs màxims
03	Error en la consigna de l'eix X. La consigna es troba fora del rang permès
04	Error en la consigna de l'eix Y. La consigna es troba fora del rang permès
05	Sense implementar
06	Desbordament dels encoders. Donat un retràs de comunicació s'ha desbordat la lectura dels encoders de forma que la dada rebuda no és vàlida
07	Error de comunicació amb el cub. No s'ha completat correctament la recepció de dades
08	Error de comunicació amb el cub. No s'ha completat correctament la transmissió de dades
09	Error de comunicació amb el dsPIC. No s'ha completat correctament la recepció de dades
10	Error de comunicació amb el dsPIC. No s'ha completat correctament la transmissió de dades
11	Obstacle detectat. El dsPIC de davant ha detectat un obstacle i ha fet parar el robot
12	Obstacle detectat. El dsPIC de darrera ha detectat un obstacle i ha fet parar el robot
13	Error de comunicació. S'ha superat el temps d'espera en la comunicació amb el cub.
14	Error al enviar la lectura de velocitat. La velocitat supera els 127 cm/s. Aquest cas només es pot donar quan va en buit.
15	Error de consigna de velocitat. Conforme a la configuració la consigna de velocitat s'ha enviat fora de rang

Taula 18. Significat dels codis d'error

Seguidament es transmeten la resta de bytes amb les informacions i lectures descrites en la Figura 64.

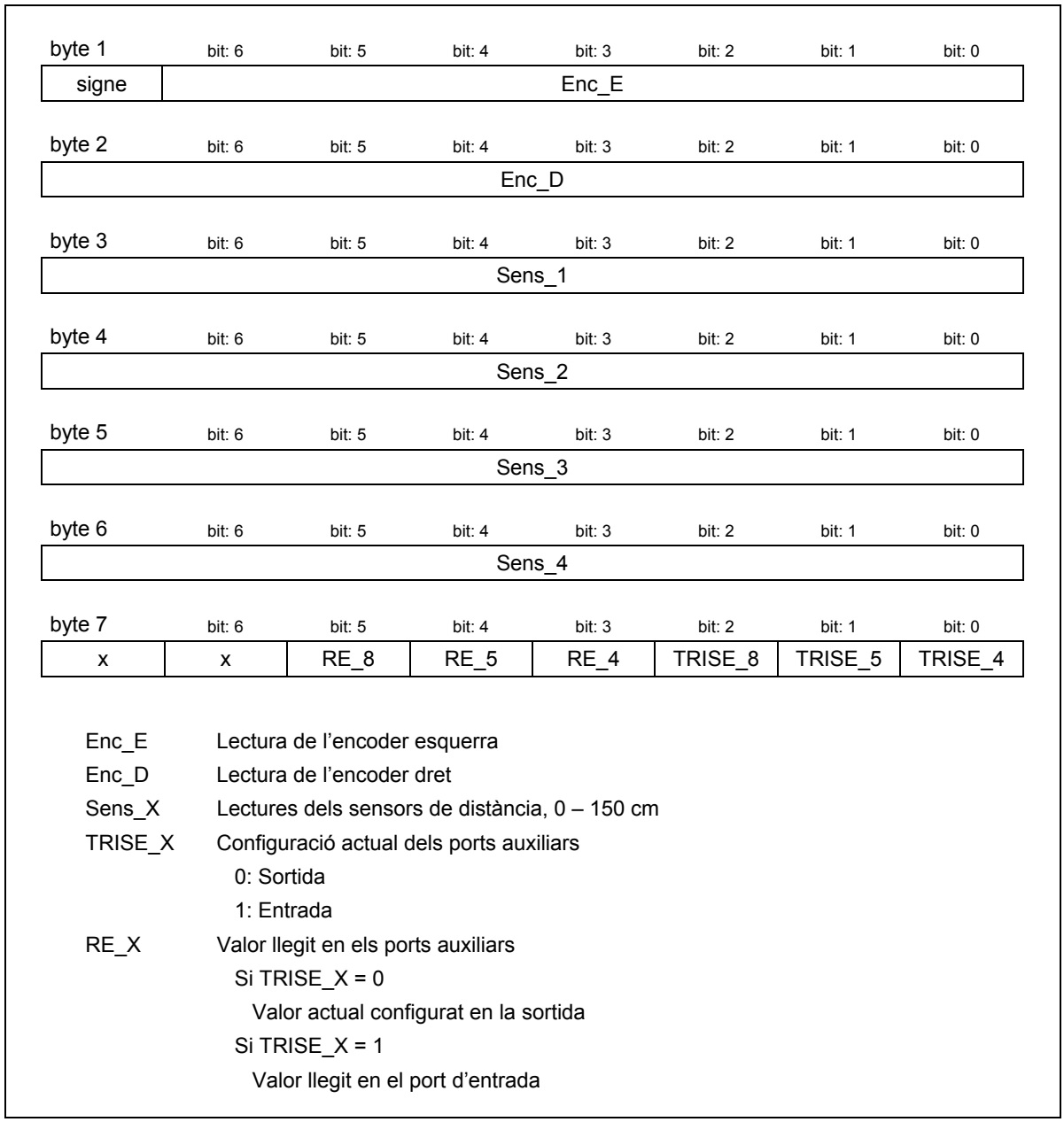

Figura 64. Dades retornades pel dsPIC de darrera

Segons la configuració enviada al dsPIC les lectures dels encoders s'enviaran en valors absoluts (tics) o en velocitat (cm/s). Si s'envien en valors absoluts aquests es troben escalats de forma que per cada volta de la roda s'envien 600 polsos. Donat que l'encoder esquerra es troba connectat a una entrada especial del dsPIC en aquest cas s'envia un bit amb el signe. Si el sentit de gir es cap a endavant aquest bit valdrà "1".

7.4.5. Pas 5. Enviar consignes al dsPIC de davant

Les dades enviades al dsPIC de davant són molt similars al cas del dsPIC de darrera amb alguna dada afegida. La Figura 65 mostra el significat de cada bit del byte de configuració.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Corr_pos	Ctl_Tf	Gir_ll	Err_r	Ctl_obs	Ctl_pos	Ctl_vel	M_enc

Ctl_vel	Control de velocitat PID
	0: Desactivat
	1: Activat
Ctl_pos	Control de posició
	0: Desactivat
	1: Activat
Ctl_obs	Control d'obstacles
	0: Desactivat
	1: Activat
Corr_pos	Correcció de posició actual
	0: Els bytes 8:12 no tenen efecte
	1: Els bytes 6:8 si s'enviaran
Gir_ll	Mode de treball del PWM
	0: Gir lliure desactivat
	1: Gir lliure activat
Err_r	Reset dels errors
	0: Sense efecte
	1: Borrarà els errors en cas de haver estat solventats
M_enc	Mode de retornar les lectures dels encoders
	0: Valors absoluts
	1: Valors de velocitat (cm/s)
Ctl_Tf	Control de l'angle final
	0: Desactivat
	1: Activat

Figura 65. Byte de configuració del dsPIC de davant

El significat dels bits Ctl_vel, Ctl_pos, Ctl_obs, Gir_ll, M_enc i Err_r és exactament el mateix que en el cas del dsPIC de darrera. El bit Corr_pos permet enviar al dsPIC una correcció de la posició actual en cas de que es disposi d'algun sistema de localització extern.

El bit de control d'angle final defineix si en mode de control de posició es controlarà l'angle en que quedarà situat el robot un cop hagi assolit la consigna de posició. Si aquest bit es desactiva l'angle en que queda el robot serà aleatori.

Seguidament s'enviaran dos bytes amb les consignes de velocitat per cada costat i el sentit de gir. La Figura 66 mostra la configuració d'aquests bytes.

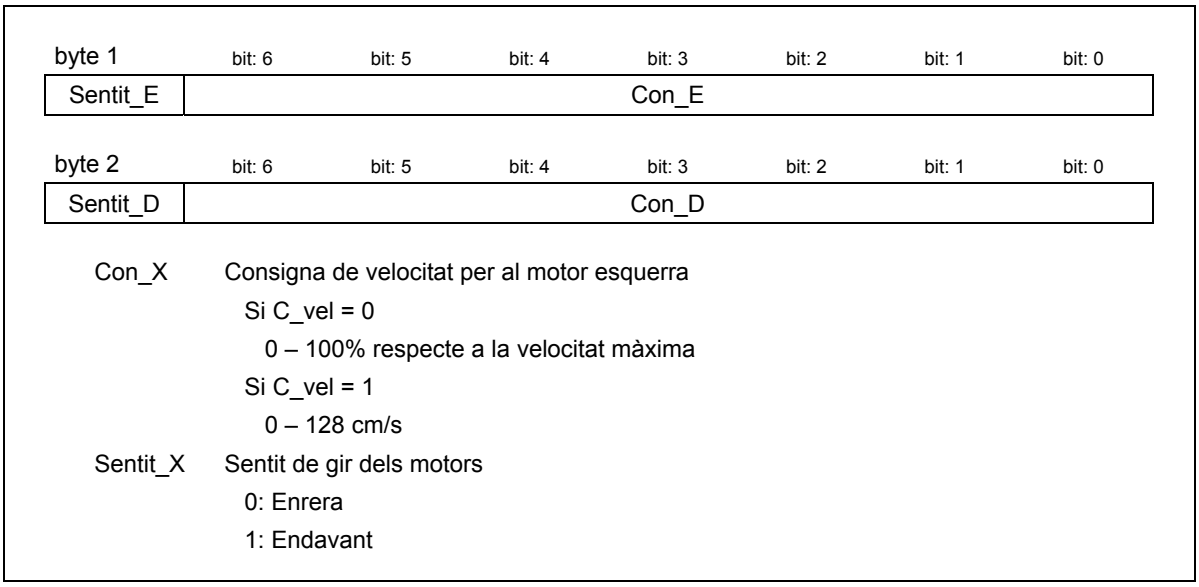

Figura 66. Consignes de velocitat

La interpretació de les consignes de velocitat serà igual que per al cas del dsPIC de darrera.

Els bytes 3:7 enviats als dsPIC's corresponen a les consignes de posició. Si el bit de control de posició s'envia com a "0" aquestes dades no tindran efecte. La Figura 67 mostra el contingut d'aquests bytes i el seu significat. Si alguna d'aquestes consignes s'envien fora dels rangs descrits el robot es pararà prudencialment fins a rebre consignes vàlides

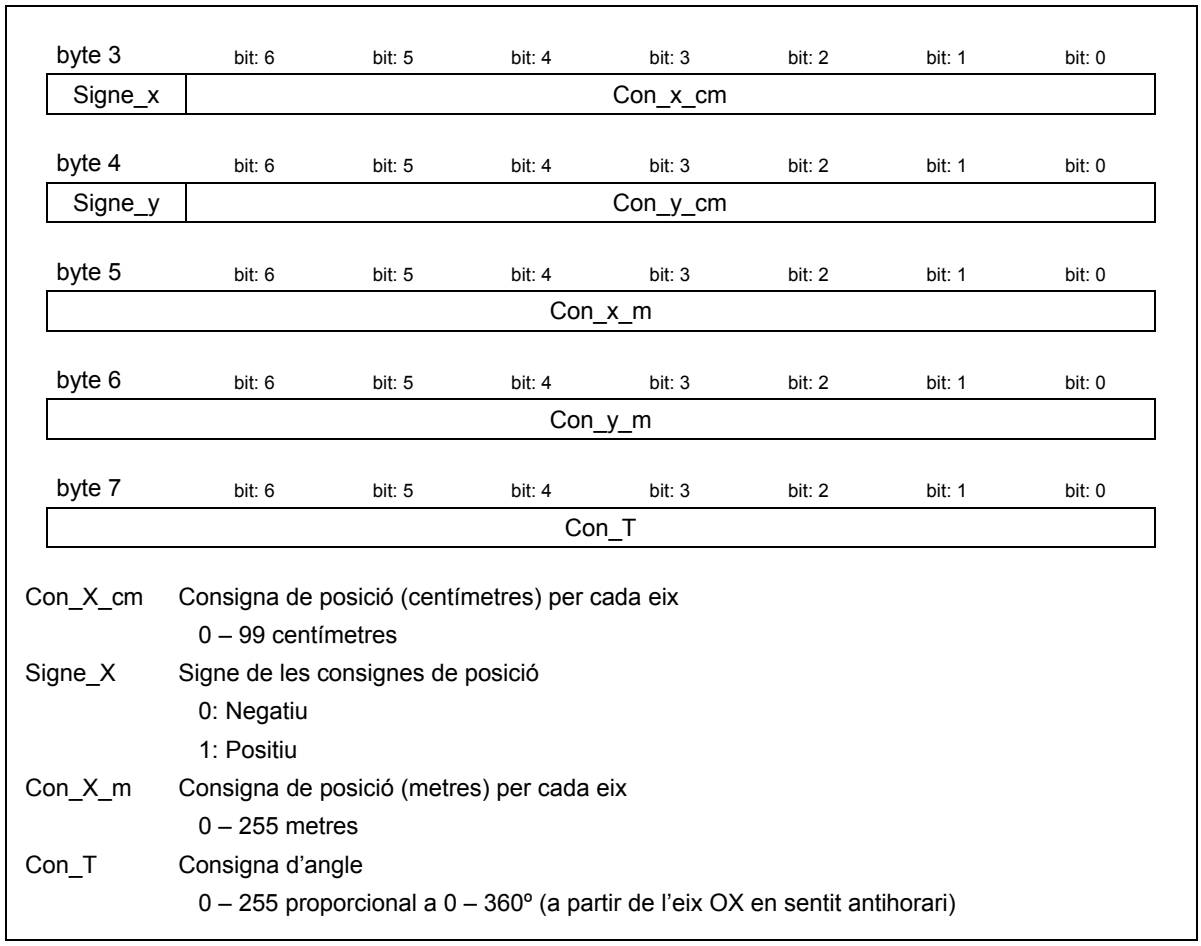

Figura 67. Consigna de posició per al dsPIC de davant

Les consignes de posició sobre els eixos X i Y s'envien en dues parts, la primera part conté la fracció de cm i la segona part conté els metres.

En cas que el bit de correcció de posició s'hagi enviat com un "1" els bytes 8:12 contindran un valor actualitzat de la posició real del robot. Aquests valors substituiran als valors calculats per la odometria del dsPIC. La Figura 68 ens mostra el contingut i significat d'aquests bytes.

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Pos_x_cm						

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Pos _y_cm						

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _x_m							

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _y_m							

byte 12	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _T							

Pos_X_cm Posició (centímetres) per cada eix
0 – 99 centímetres
Signe_X Signe de les posicions enviades
0: Negatiu
1: Positiu
Pos _X_m Posició (metres) per cada eix
0 – 255 metres
Pos _T Posició angular del robot
0 – 255 proporcional a 0 – 360º (a partir de l'eix OX en sentit antihorari)

Figura 68. Correcció de posició

Igual que en el cas anterior les posicions sobre l'eix X i Y s'envien en dues parts, la fracció de centímetres i els metres.

Per últim s'enviaran les configuracions per als ports auxiliars del dsPIC de la mateixa forma que per al dsPIC de darrera. La Figura 69 mostra el significat de cada bit d'aquest byte.

byte 13	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

TRISE_X Sentit de les dades del port RE_X
0: Sortida
1: Entrada
RE_X Valor del port de sortida
Si TRISE_X = 0:
0: Sortida desactivada ('0' lògic)
1: Sortida activada ('1' lògic)
Si TRISE_X = 1:
x: No tenen efecte

Figura 69. Ports auxiliars del dsPIC de davant

7.4.6. Pas 6. Recepció de dades del dsPIC de davant

Les dades rebudes pel dsPIC de davant seran de la mateixa tipologia que per al dsPIC de darrera amb l'excepció que aquest també ens retorna la posició calculada. Tal i com es mostra a la Figura 70 primer s'enviarà un byte amb els possibles errors. Els codis d'error son els mateixos que els descrits en la Taula 18

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_des	Err_cons	Err_com	Cod_err				

Err_des Ens indica que algun valor de posició hi ha hagut desbordament
0: No hi ha desbordament
1: Hi ha desbordament. El robot està parat

Err_cons Alguna de les consignes s'ha enviat fora de rang
0: Consignes correctes
1: Consignes errònies. El robot està parat

Err_com Error de comunicació amb l'altre dsPIC
0: La connexió es correcte
1: Hi ha hagut fallada de comunicació. El robot funciona igualment

Cod_err Codi de errors

Figura 70. Codis d'error del dsPIC de davant

Seguidament el dsPIC respondrà amb la resta de bytes que contenen les informacions de posició i lectures dels sensors. La Figura 71 i la Figura 72 mostren totes les dades transmeses.

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_E							

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
signe	Enc_D						

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_1							

byte 4	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_2							

byte 5	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_3							

byte 6	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_4							

byte 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Pos_x_cm						

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Pos _y_cm						

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _x_m							

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _y_m							

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _T							

byte 12	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

Figura 71. Dades rebudes del dsPIC de davant

Enc_E	Lectura de l'encoder esquerra
Enc_D	Lectura de l'encoder dret
Sens_X	Lectures dels sensors de distància, 0 – 150 cm
Pos_X_cm	Posició (centímetres) per cada eix
	0 – 99 centímetres
Signe_X	Signe de les posicions enviades
	0: Negatiu
	1: Positiu
Pos _X_m	Posició (metres) per cada eix
	0 – 255 metres
Pos _T	Posició angular del robot
	0 – 255 proporcional a 0 – 360º (a partir de l'eix OX en sentit antihorari)
TRISE_X	Configuració actual dels ports auxiliars
	0: Sortida
	1: Entrada
RE_X	Valor llegit en els ports auxiliars
	Si TRISE_X = 0
	Valor actual configurat en la sortida
	Si TRISE_X = 1
	Valor llegit en el port d'entrada

Figura 72. Significat de les dades rebudes del dsPIC de davant

7.4.7. Pas 7. Enviar consignes des del dsPIC de davant al de darrera.

Les dades que el dsPIC de davant transmeti al de darrera depenen de la configuració que s'ha enviat a cada un dels dsPIC's. Si el control de posició i el control d'obstacles estan desactivats no s'enviaran dades útils. D'altra manera la Figura 73 mostra les dades que es transmetran.

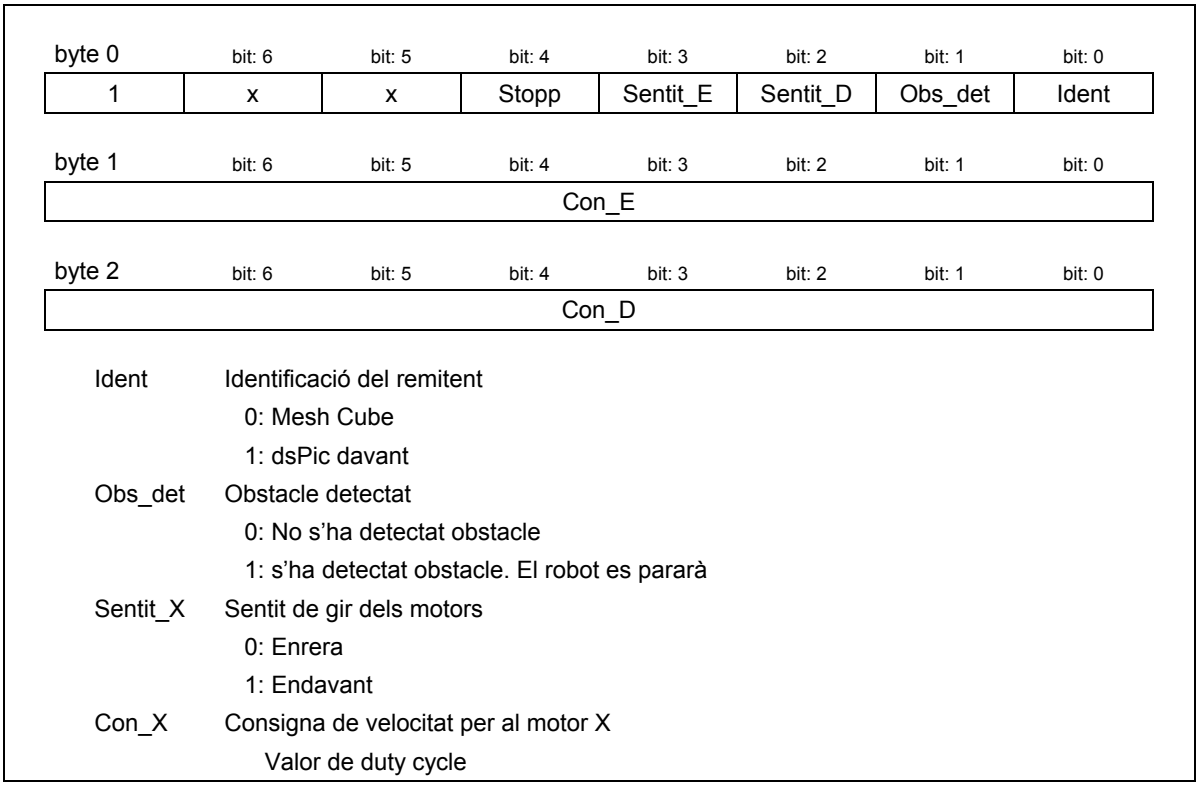

Figura 73. Dades enviades del dsPIC de davant al de darrera.

7.4.8. Pas 8. Recepció de dades del dsPIC de darrera al de davant

Les dades que el dsPIC de darrera envia al de davant serviran per tal de realitzar el control de posició i la detecció d'obstacles. La Figura 74 mostra els bytes que s'enviaran segons la configuració establerta.

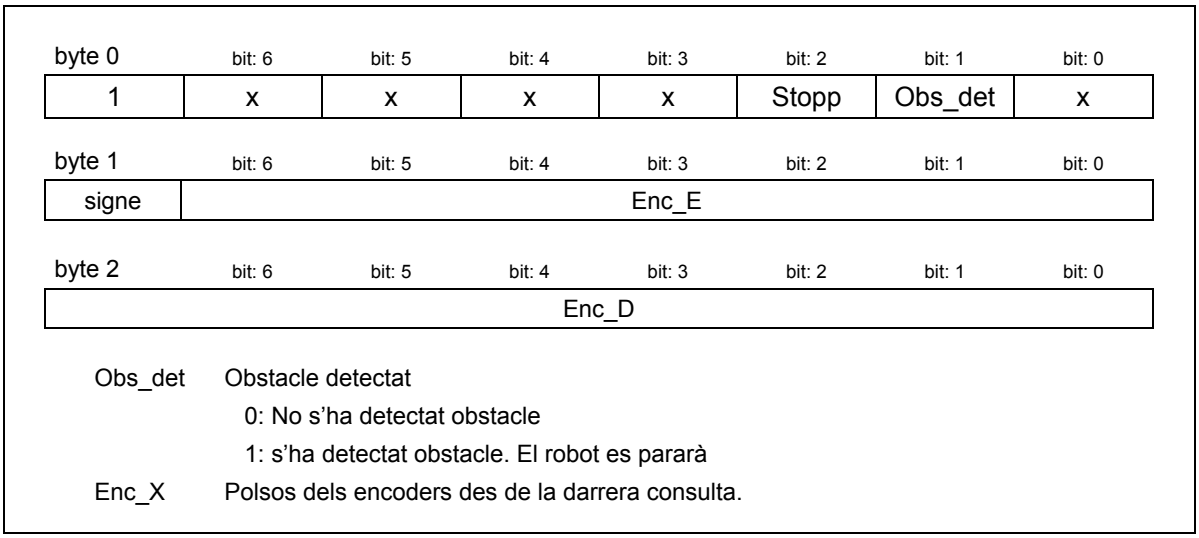

Figura 74. dades retornades pel dsPIC de darrera al de davant

8. PONT DE COMUNICACIÓ ENTRE I²C I WIFI

Des del Cub podem accedir directament al bus I²C i comunicar amb els diversos dispositius seguint el protocol descrit en l'apartat anterior. En aquest nivell de programació es poden implementar funcions de major nivell com ara la presa de decisions. Ara bé, degut a la possible necessitat de la intervenció humana és precís poder també programar en un nivell superior extern al robot com ara un PC. En aquest apartat es descriu el funcionament d'un programa que actua com a pont entre el bus I²C i el WiFi. La funcionalitat d'aquest programa es la mateixa que el "Simple_Server" descrit en l'annex B, afegint les noves prestacions del robot. Aquest programa descarta l'opció de prendre decisions en aquest nivell i simplement reagrupa les dades rebudes pel bus I²C per enviar-les per wifi i distribueix les ordres rebudes per wifi cap als diferents dispositius del bus I²C. La Figura 75 il·lustra aquest sistema de pont de comunicacions. En els successius apartats ens referirem en aquest programa com el "Programa Pont".

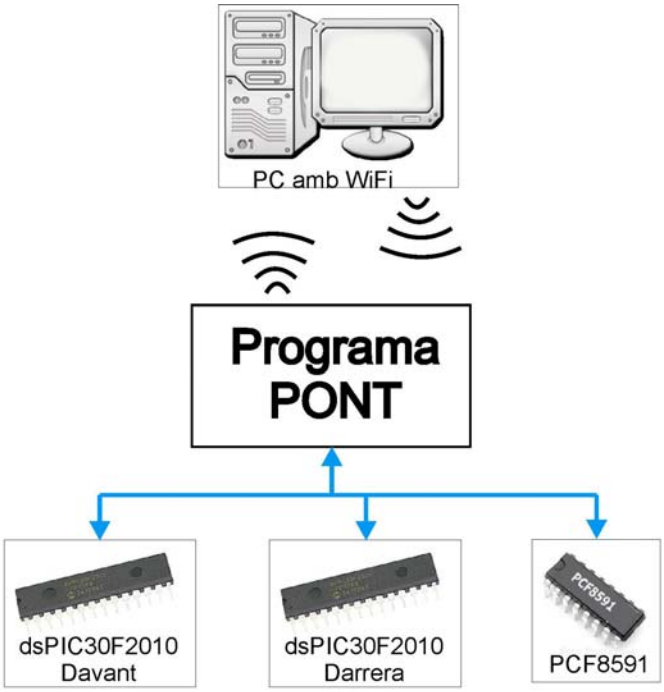

Figura 75. Programa pont

8.1. El programa Pont

Com ja s'ha descrit aquest programa gestiona les comunicacions del bus I²C tal com es descriu en l'apartat anterior i les retransmet via wifi. Per tal d'evitar trànsit de dades innecessàries i redundàncies les dades es recomponen en un sol paquet.

8.1.1. Dades que rep el programa pont des de l'exterior

A continuació es descriuen les dades que rebrà el programa pont a través de wifi. Primerament s'enviaran dos bytes que corresponen a la configuració i el mode de funcionament del control dels motors. La Figura 76 mostra el significat dels diversos bits. Per a tots els valors les lletres "D" i "E" fan referència a dreta i esquerra respectivament mentre que les lletres "F" i "R" es refereixen a davant i darrera.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	Gir_ll_R	Gir_ll_F	x	x	Ctl_obs	Ctl_pos	Ctl_vel

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	x	M_enc	Ctl_Tf	Err_r_R	Err_r_F	Corr_pos

Ctl_vel	Control de velocitat PID
	0: Desactivat
	1: Activat
Ctl_pos	Control de posició
	0: Desactivat
	1: Activat
Ctl_obs	Control d'obstacles
	0: Desactivat
	1: Activat
Corr_pos	Correcció de posició actual
	0: S'envia una correcció de posició
	1: No s'envia una correcció de posició
Gir_ll_X	Mode de funcionament del PWM
	0: Gir lliure desactivat
	1: Gir lliure activat
Err_r_X	Reseteja els valors d'error dels dsPIC's
	0: No pren cap acció
	1: Reseteja l'error
M_enc	Mode de lectura dels encoders
	0: Lectures absolutes
	1: Lectures en valors de velocitat
Ctl_Tf	Control de l'angle final
	0. Desactivat
	1: Activat

Figura 76. Configuracions des del exterior

Els propers quatre bytes que s'han d'enviar corresponen a les consignes de velocitat per cada un dels quatre motors. La Figura 77 detalla la composició d'aquests bytes i el rang de valors admesos.

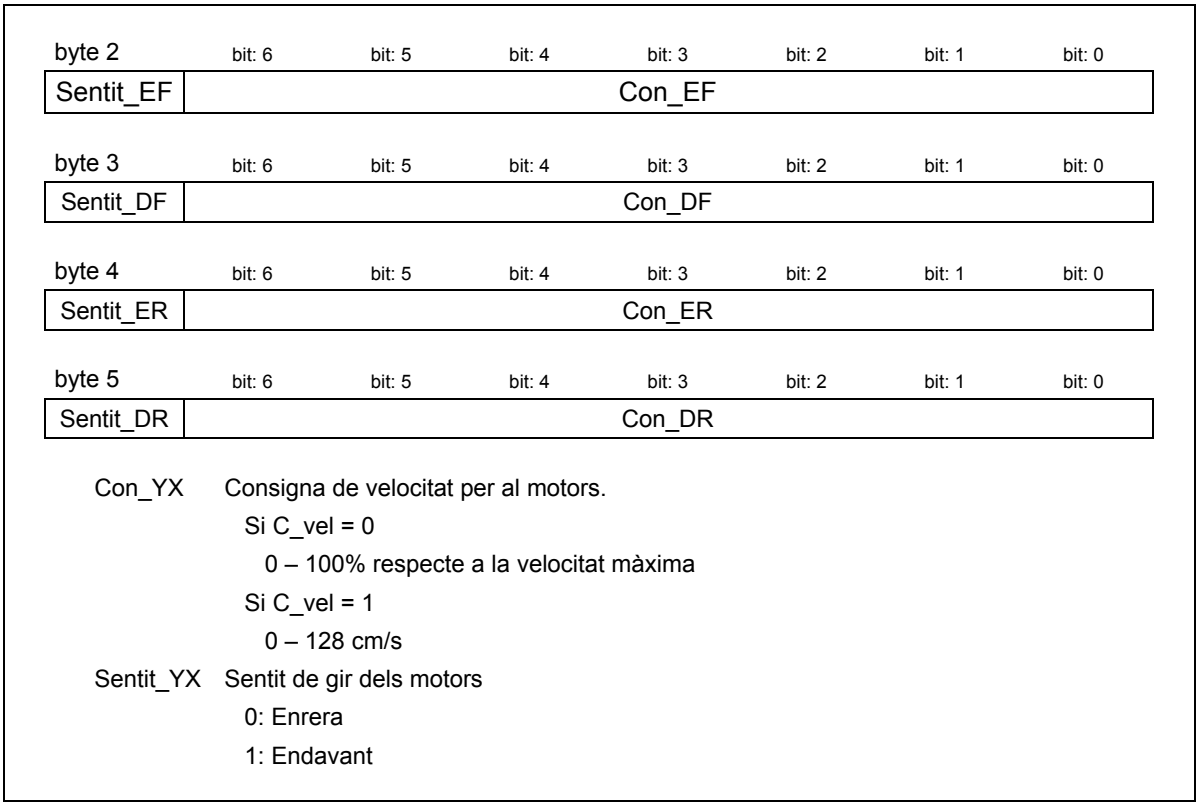

Figura 77. Consignes des de l'exterior

Els bytes 6:10 detallats en la Figura 78 han de contenir la consigna de posició per al robot. En cas que el bit de control de posició no es trobi actiu aquestes dades no es tindran en compte en quant al mode de funcionament.

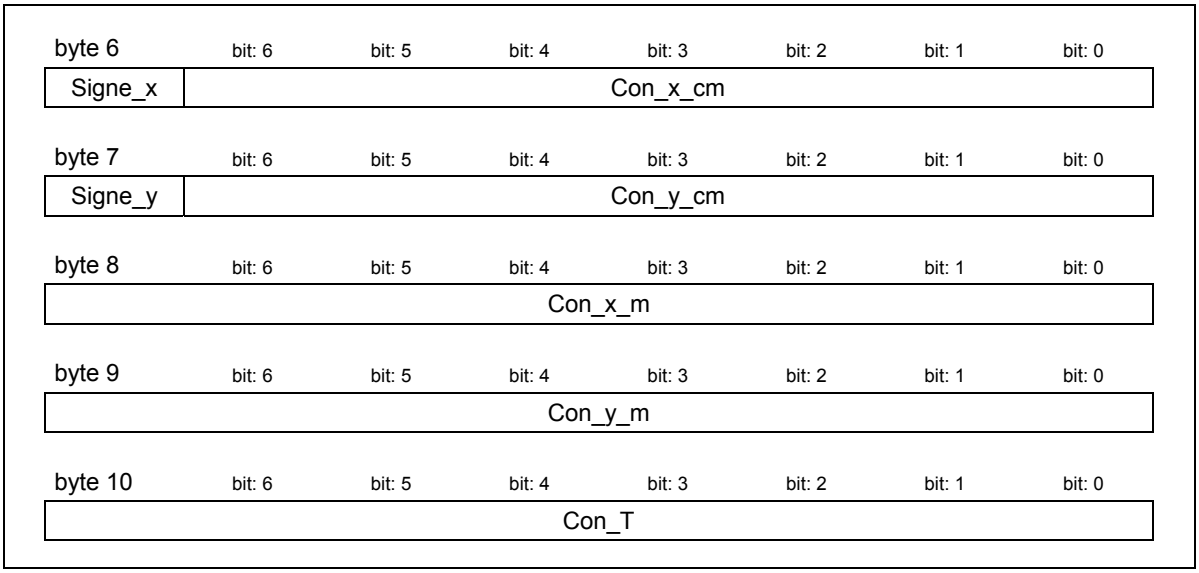

Figura 78. Consignes de posició des de l'exterior

En cas de disposar d'un mètode de localització extern per al robot es pot combinar aquest sistema amb la odometria interna del robot. Així amb els últims bytes podem enviar una correcció de posició per al robot. Si el bit "Corr_pos" s'envia com a actiu aquestes dades

es substituiran pels valors calculats pel robot, d'altra manera no es tindran en compte. La Figura 79 mostra la composició d'aquests últims 5 bytes.

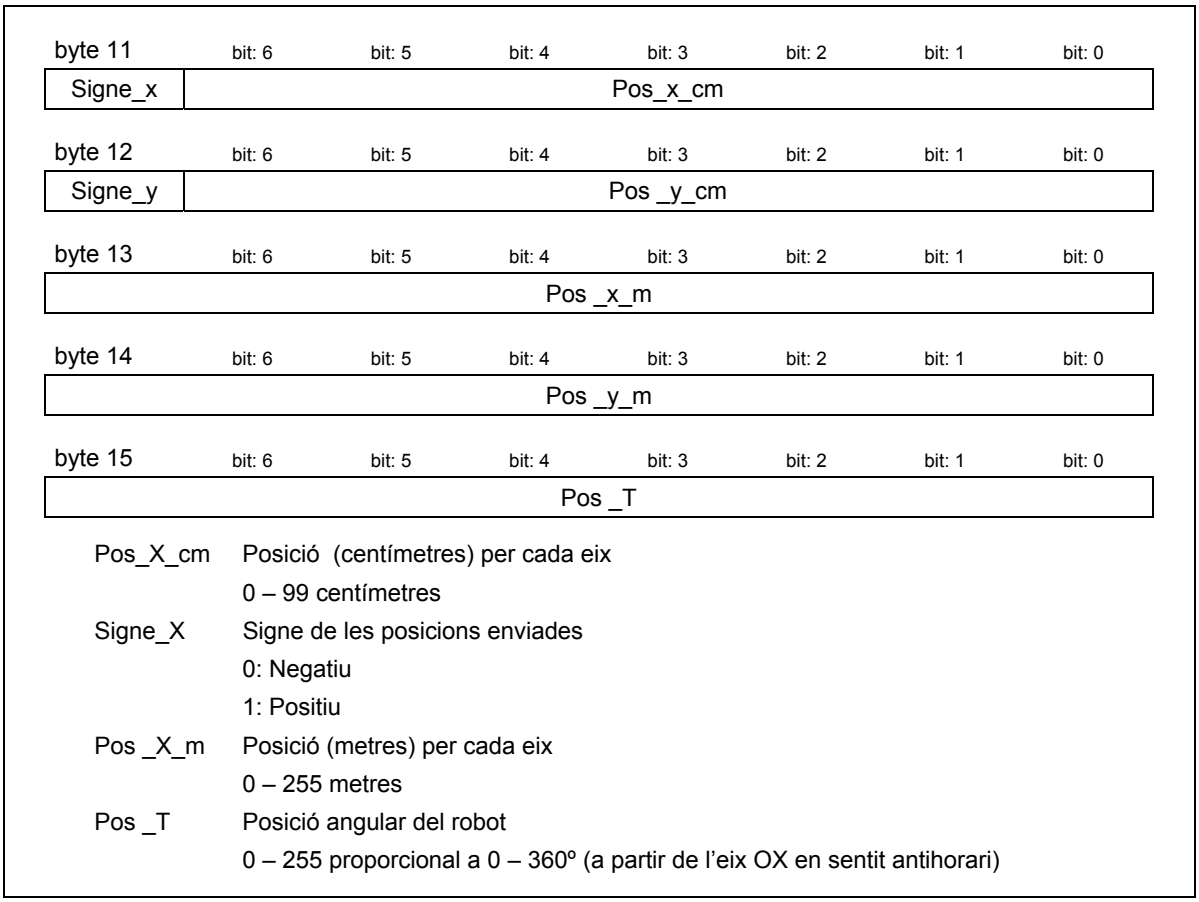

Figura 79. Correccions de posició des de l'exterior

Els bytes 16 i 17 contenen la configuració per als ports auxiliars dels dos dsPIC's de manera que es poden configurar com a entrades i sortides digitals i en cas de ser sortides fixar l'estat. La Figura 80 mostra la composició dels dos bytes i el significat dels diversos bits.

byte 16	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
X	x	RE_8F	RE_5F	RE_4F	TRISE_8F	TRISE_5F	TRISE_4F

byte 17	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8R	RE_5R	RE_4R	TRISE_8R	TRISE_5R	TRISE_4R

TRISE_XX Configuració dels ports auxiliars
0: Sortida
1: Entrada

RE_XX Valor llegit en els ports auxiliars
Si TRISE_X = 0
Valor actual configurat en la sortida
Si TRISE_X = 1
Valor llegit en el port d'entrada

Figura 80. Configuració dels ports auxiliars des de l'exterior

8.1.2. Dades retornades pel programa pont

Un cop s'han enviat totes les dades al programa pont aquest les distribueix al corresponent dsPIC per aplicar les configuracions i consignes. Seguidament el programa llegirà en cada dsPIC els valors retornats i els agruparà per transmetre'ls a l'exterior.

A continuació es descriurà el contingut de dades retornat pel programa pont. Les primeres dades retornades detallades en la Figura 81 donen l'estat de la bateria i els possibles codis d'error retornats per cada un dels dsPIC's.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Lectura conversor AD (0-255)							

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_desF	Err_consF	Err_comF	Cod_errF				

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_desR	Err_consR	Err_comR	Cod_errR				

Err_des Ens indica que algun valor de posició hi ha hagut desbordament
0: No hi ha desbordament
1: Hi ha desbordament. El robot està parat

Err_cons Alguna de les consignes s'ha enviat fora de rang
0: Consignes correctes
1: Consignes errònies. El robot està parat

Err_com Error de comunicació amb l'altre dsPIC
0: La connexió es correcte
1: Hi ha hagut fallada de comunicació. El robot funciona igualment

Cod_err Codi de errors

Figura 81. Estats retornats pel programa pont

El significat dels codis d'error ja s'ha descrit en l'apartat 7.4.3. Les següents dades transmeses resumides en la Figura 82 corresponen a les lectures dels encoders i dels sensors de distància infrarojos.

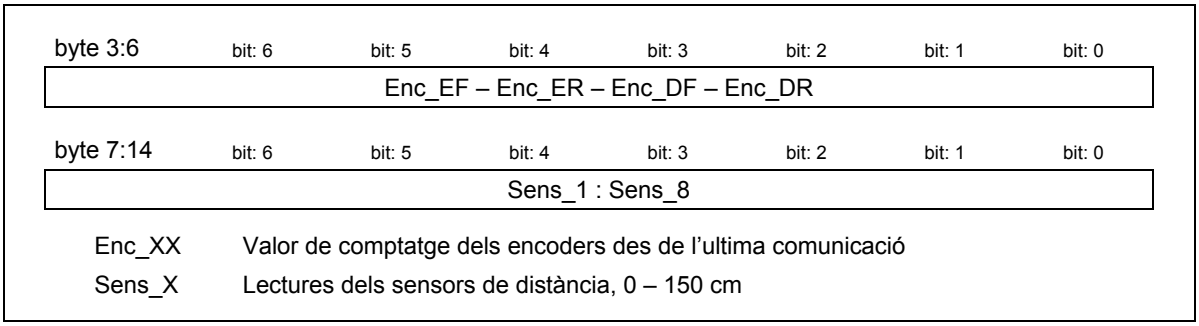

Figura 82. Valor dels encoders i sensors retornats pel programa pont

Seguidament el programa pont retorna la posició del robot segons els càlculs d'odometria efectuats en el dsPIC de davant. La Figura 83 detalla el format d'aquestes dades.

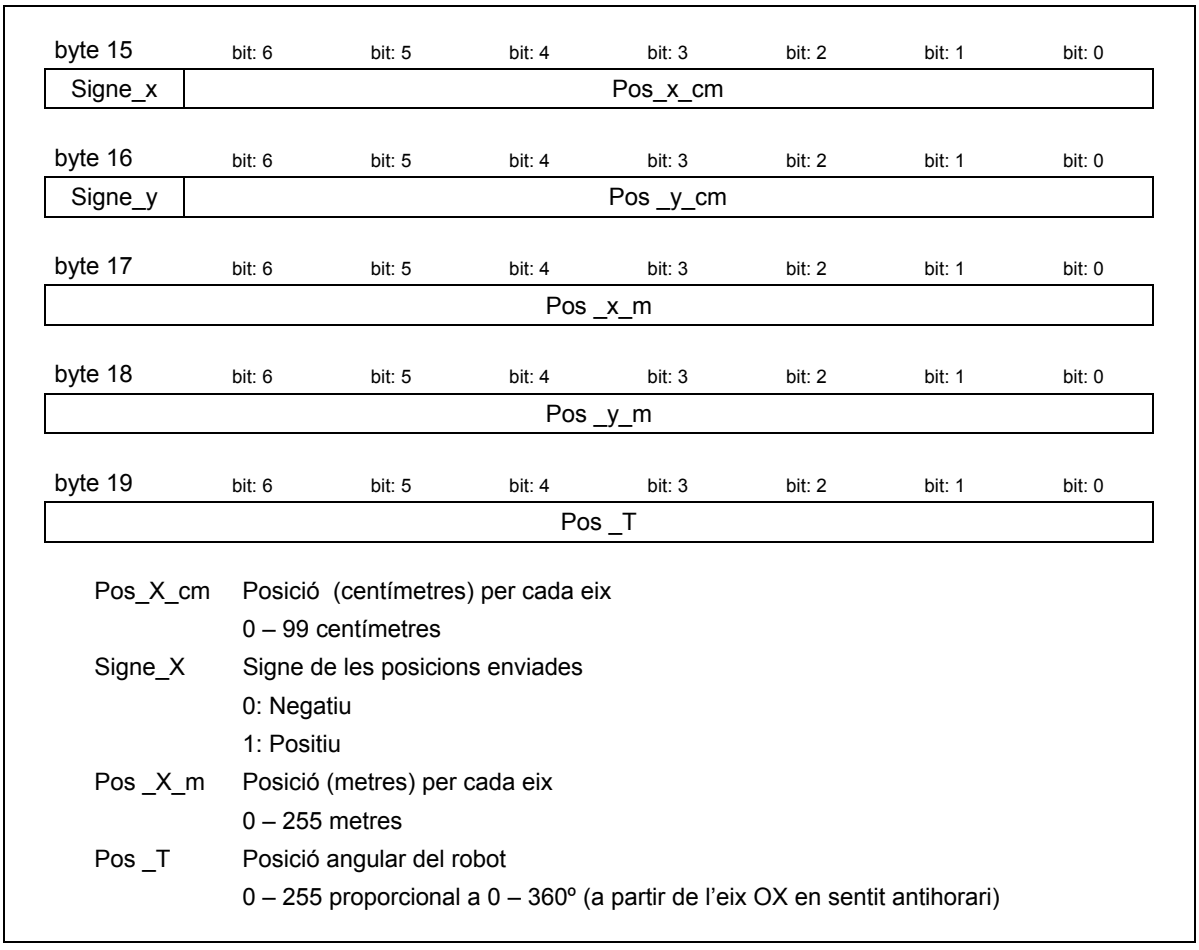

Figura 83. Valors de posició del robot

Finalment l'últim byte detallat en la Figura 84 ens mostrarà l'estat dels ports auxiliars de cada un dels dsPIC's.

byte 20	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8R	RE_5R	RE_4R	RE_8F	RE_5F	RE_4F

RE_XX Valor llegit en els ports auxiliars
Si TRISE_X = 0
Valor actual configurat en la sortida
Si TRISE_X = 1
Valor llegit en el port d'entrada

Figura 84. Ports auxiliars

9. CONSOLA DE CONTROL PER L'USUARI

Les aplicacions descrites fins aquest punt no permeten l'intervenció del usuari. Els programes per als dsPIC's proporcionen un nivell bàsic de control i comunicació. Es pot dir que les possibilitats en aquest nivell de control estan pràcticament esgotades. Tot i que es podran fer potser algunes petites millores ja no es podran afegir noves prestacions d'importància.

El programa pont ubicat en el cub actua com a unió entre el nivell més baix, els dsPIC's, i el nivell més alt. En aquest punt hi ha capacitat i potencial per implementar noves funcions.

En aquests dos nivells no es poden implementar aplicacions que requereixin l'intervenció de l'usuari, per tant aquesta necessitat s'implementa en un sistema exterior.

En aquest apartat es descriuen dues aplicacions dissenyades per fer d'interfície entre el robot i l'usuari. Aquesta interfície es pot implementar sobre qualsevol plataforma programable amb interfície de comunicacions wifi.

Per al present projecte es presenten aplicacions per dues plataformes diferents. Per una banda s'ha dissenyat una aplicació creada en l'entorn de Lab View amb la qual es pot interactuar amb el robot des d'un PC. Per altra banda s'ha creat una aplicació amb l'entorn devKitPro el qual es pot executar en una consola de jocs Nintendo DS.

9.1. Interfície per PC

Mitjançant l'entorn de programació Lab View s'ha dissenyat una aplicació que permet controlar i provar totes les funcions implementades en els dsPIC's. Donat que es tracta d'un entorn de programació gràfic no es detalla el codi del programa. En aquest apartat es farà una breu descripció de l'aplicació i de les seves funcions.

9.1.1. Aspecte global

La Figura 85 mostra l'aspecte global de l'aplicació dissenyada. La finestra es divideix en diferents subapartats segons la funció.

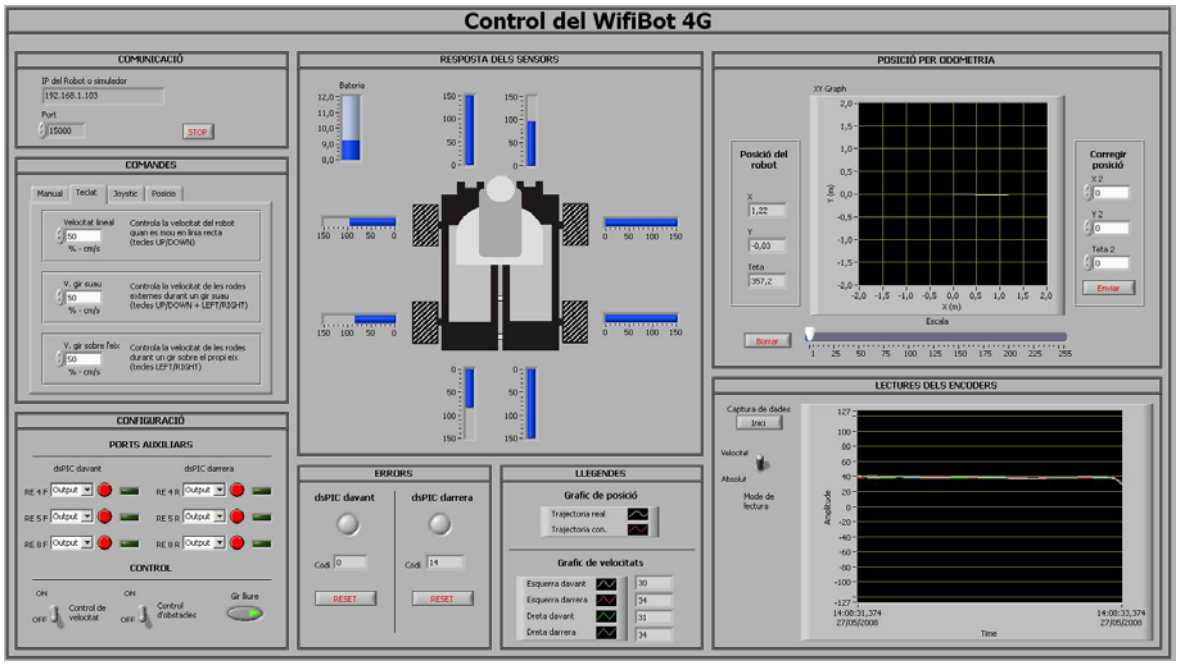

Figura 85. Aspecte global de l'aplicació de control

9.1.2. Comunicació

Aquest apartat permet configurar l'adreça IP del robot i el port a través del qual es connectarà. El botó Stop permet interrompre la comunicació.

En la Figura 86 veiem l'aspecte de la finestra. Els valors per defecte ja son correctes per poder comunicar amb el robot.

Figura 86. Configuració de la comunicació

9.1.3. Resposta dels sensors

Aquest apartat permet visualitzar de forma gràfica les lectures dels sensors de distància i l'estat de la bateria. La Figura 87 mostra l'aspecte de la finestra

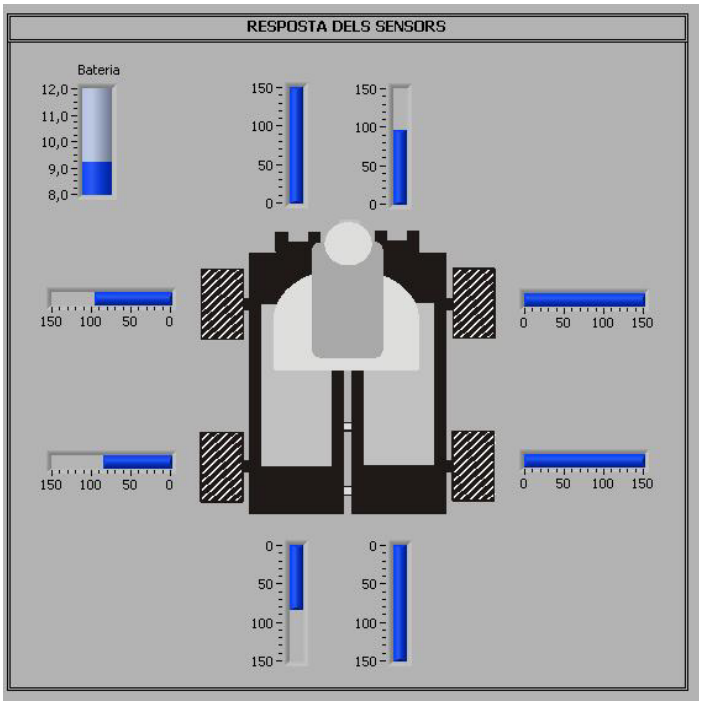

Figura 87. Lectures dels sensors

9.1.4. Configuració

En aquest apartat es pot accedir a les configuracions bàsiques del robot. Es pot seleccionar el sentit de funcionament dels ports auxiliars, és a dir, es poden configurar com a sortides o com a entrades. En cas de configurar-se com a sortides es pot seleccionar l'estat de cada un.

En la Figura 88 es mostra l'apartat, es veu com a part es pot seleccionar si es vol el mode de funcionament amb control d'obstacles o si es vol activar el control de velocitat per PID.

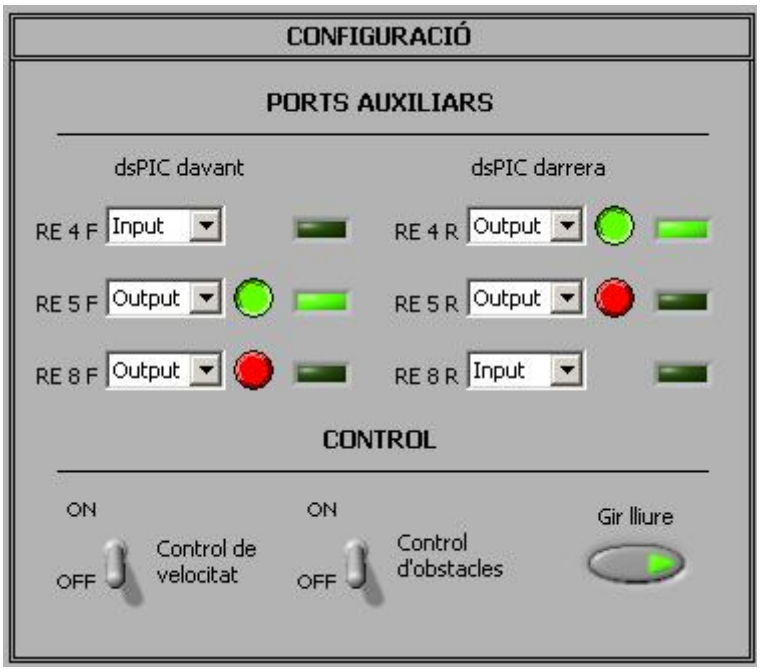

Figura 88. Configuració

9.1.5. Quadre de comandes

Dins del quadre de comandes es troben diferents pestanyes. Cada pestanya permet enviar comandes al robot mitjançant un mètode diferent.

Comandament manual (Figura 89.a): Permet donar una consigna de velocitat individual a cada roda del robot. El botó de Stop força a "0" totes les consignes.

Comandament per teclat (Figura 89.b): Permet controlar el moviment del robot a través de les quatre tecles direccionals del teclat. Hi ha tres caselles per seleccionar les velocitats. La primera correspon a la velocitat del moviment endavant o endarrera, la segona correspon a la velocitat durant un gir suau (rodes interiors parades) i l'ultima correspon a les velocitats per un gir sobre el propi eix.

Comandament per joystick (Figura 89.c): Permet controlar el robot a través d'un Joystick connectat al PC pel port USB.

Comandament per consignes de posició (Figura 89.d): Permet entrar consignes de posició, en el moment que es valida la consigna el robot començarà a desplaçar-se cap a la consigna. Si s'activa el control d'angle final es pot donar també una consigna per l'angle final.

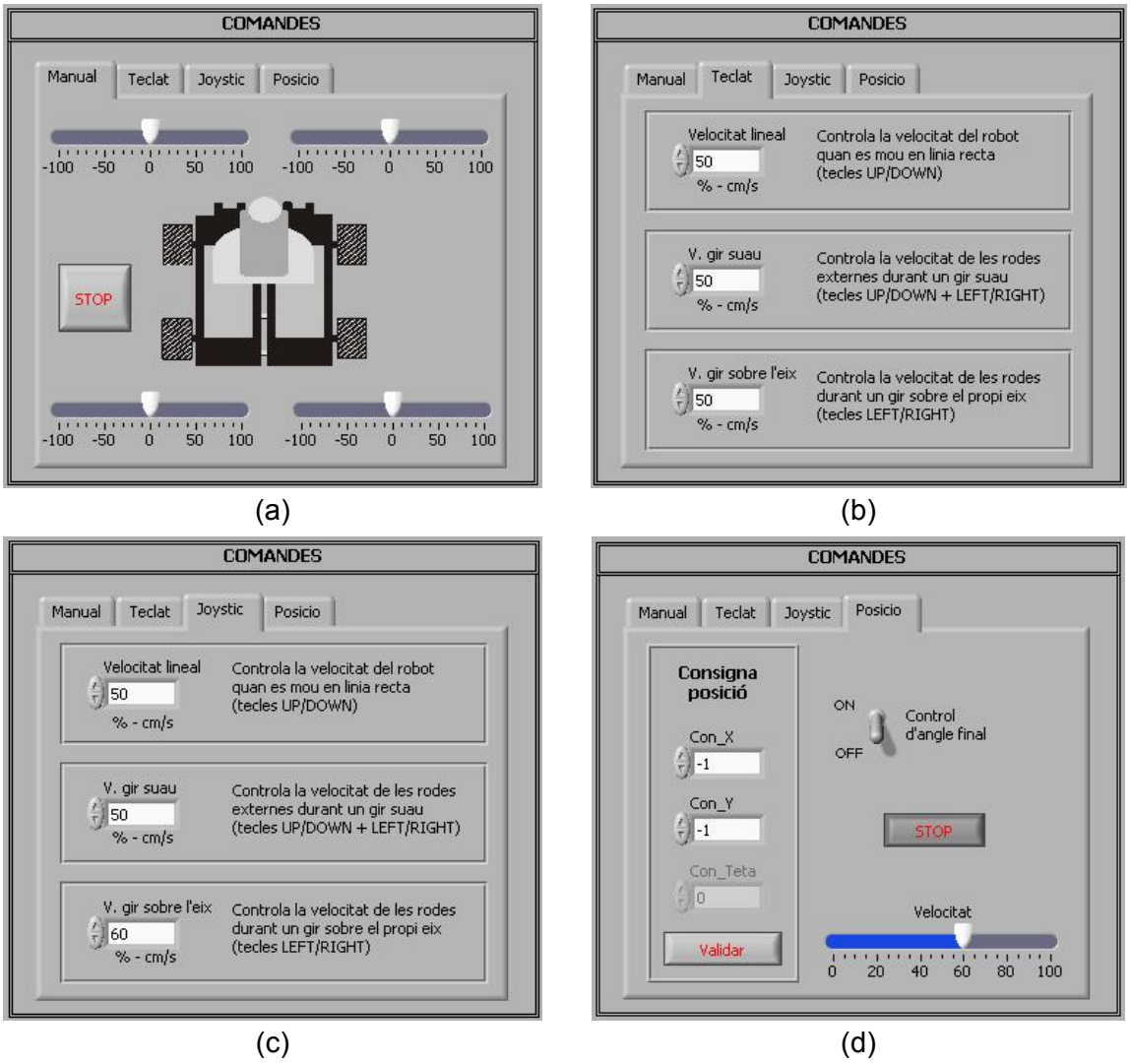

Figura 89. Quadres de comandes

9.1.6. Gràfic de posició

Aquest apartat permet observar la posició del robot tant de forma gràfica com de forma numèrica. Els controls de la banda dreta permeten corregir si es vol la posició calculada pel robot. En la Figura 90 s'observa com el gràfic presenta dues línies, la vermella correspon a les consignes de posició que s'han enviat al robot i la blanca correspon al recorregut real del robot. La barra de desplaçament inferior permet modificar els rangs dels eixos.

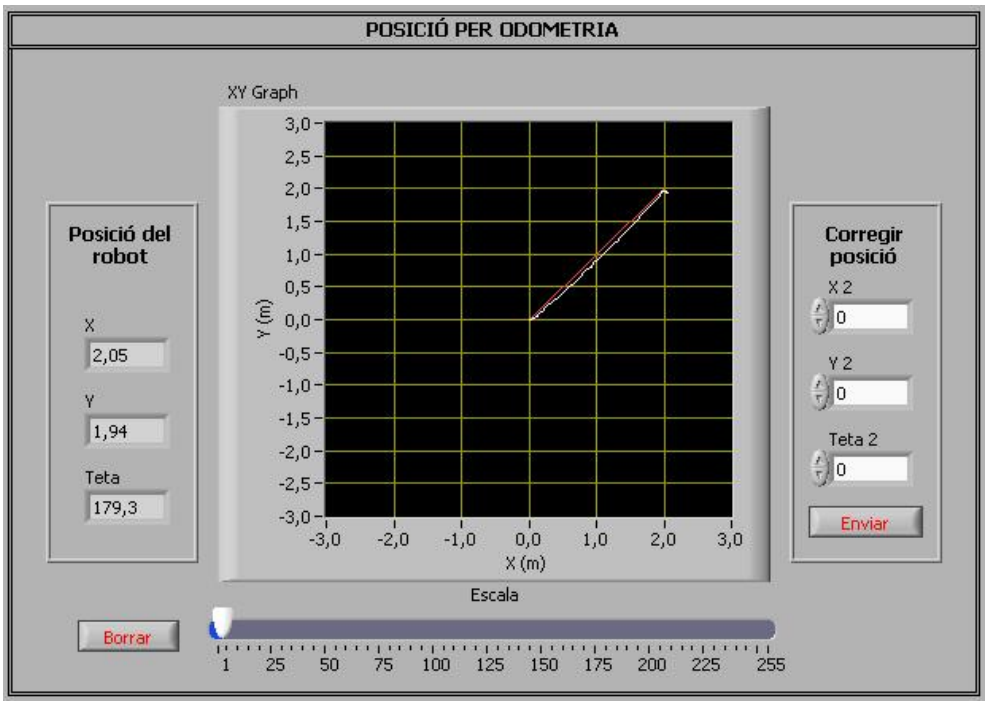

Figura 90. Posició del robot

9.1.7. Lectures dels encoders

Aquest apartat mostra en un gràfic (Figura 91) les lectures dels encoders. Mitjançant el selector es configura si les lectures corresponen a velocitats o a valors absoluts.

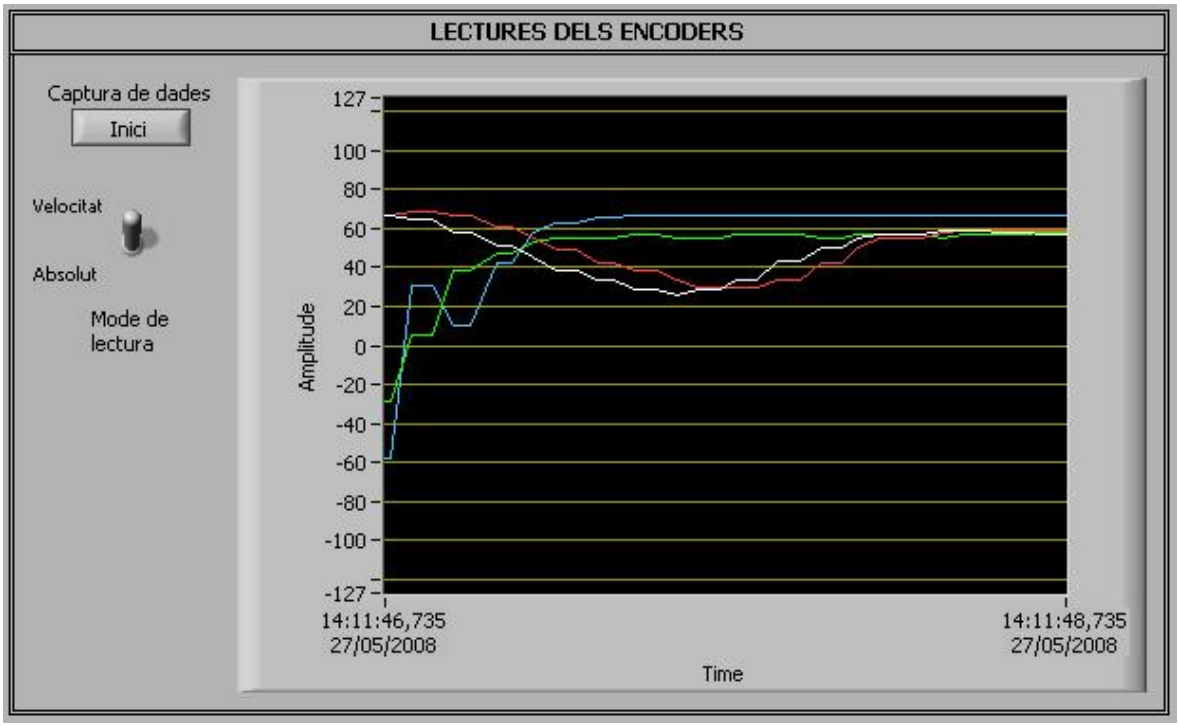

Figura 91. Lectures dels encoders

9.1.8. Errors

Aquest apartat mostra si s'ha produït algun error. Si l'error ha provocat que el robot es pari per la gravetat de l'error es comunicarà amb un missatge emergent, d'altra manera es mostrarà el codi de l'error si aquest no impedeix el funcionament. En la Figura 92 s'observen els dos botons de reset que faran reiniciar el robot si es que l'error s'ha corregit.

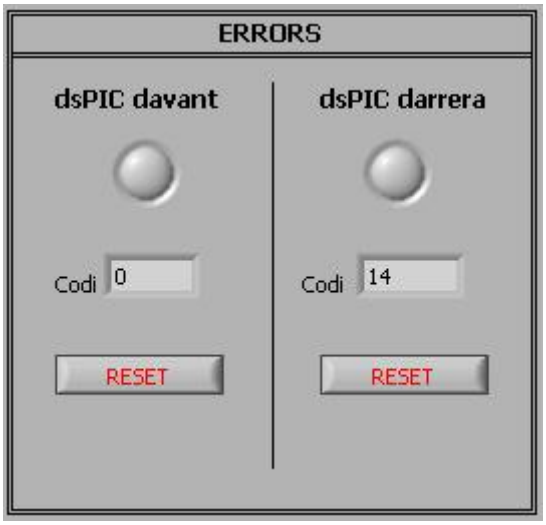

Figura 92. Quadre de errors

9.2. Interfície per Nintendo DS

Per tal de mostrar que es pot controlar el robot mitjançant qualsevol dispositiu amb wifi programable en aquest apartat es presenta una senzilla aplicació capaç de córrer en una consola de jocs Nintendo DS.

L'aplicació s'ha creat amb l'entorn de programació devKitPro, en l'annex A es detalla el codi de l'aplicació.

A diferència de l'aplicació presentada en l'apartat anterior aquesta només permet controlar les funcions bàsiques del robot.

L'entorn gràfic de l'aplicació es divideix en les dues pantalles de la consola. En la pantalla superior es mostra una espècie de consola que recull les accions preses com ara la modificació de paràmetres o la connexió i desconnexió del robot. La pantalla inferior presenta en tot moment un entorn gràfic que permet modificar les configuracions i els paràmetres de la connexió.

El primer menú que apareixerà en la pantalla tàctil inferior permet modificar la configuració TCP/IP de la conola (Figura 93). Les configuracions que apareixen per defecte ja permeten comunicar amb el Wifibot, tot i això es poden modificar tots els paràmetres pensant en futures aplicacions amb múltiples robots.

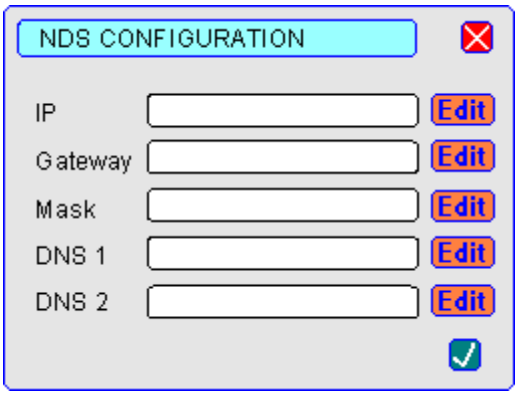

Figura 93. Configuració de la consola

Si es vol modificar algun dels paràmetres es pot premer el botó d'editar al costat del paràmetre, d'aquesta manera apareixerà un teclat (Figura 94) sobre la pantalla tàctil que permet escriure els nous valors.

Figura 94. Finestra d'edició

Quan s'accepta la configuració apareix una finestra amb una llista de xarxes wifi que rep la consola, d'entre aquestes s'ha de seleccionar la xarxa del robot i acceptar.

La següent finestra permet especificar l'adreça IP i el port del robot per efectuar la connexió (Figura 95), els valors que apareixen per defecte son correctes per al robot actual.

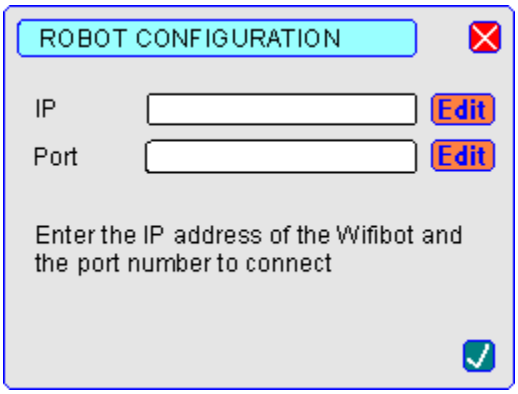

Figura 95. Configuració de la connexió

Finalment l'ultima finestra mostra el control del robot. Un cop establerta la comunicació es podrà controlar el robot amb les tecles ←, ↑, → i ↓. En la finestra es poden especificar les diverses velocitats per al moviment en línia recta, el gir i el gir sobre el propi eix (figura).

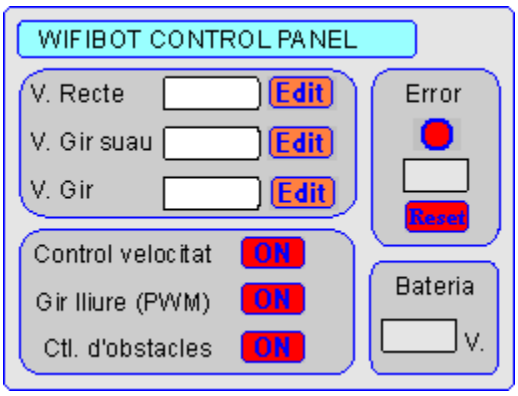

Figura 96. Finestra de control

En aquesta mateixa finestra es pot observar si s'ha produït algun error i l'estat en tot moment de la bateria.

PROVES I RESULTATS DE LA ODOMETRIA

En aquest apartat s'exposen algunes proves realitzades amb les aplicacions dissenyades a fi de mostrar gràficament que el control de posició compleix amb les expectatives proposades.

Les proves s'han realitzat donant diverses consignes de posició amb l'aplicació de control per PC i observant el recorregut calculat respecte al recorregut real fet pel robot.

Donat que el control de posició considera una consigna per assolida quan la distància al punt final és inferior a 10 centímetres, els errors expressats en les proves es referencien respecte a la posició on s'hauria de trobar el robot segons els càlculs d'odometria i no respecte la consigna.

9.3. Prova 1

Aquesta prova s'ha realitzat amb el control d'angle final activat i el control de velocitat desactivat. Els motors es controlen amb el PWM en mode de funcionament sense gir lliure.

La Figura 97 mostra l'evolució del moviment del robot respecte la consigna per la prova realitzada.

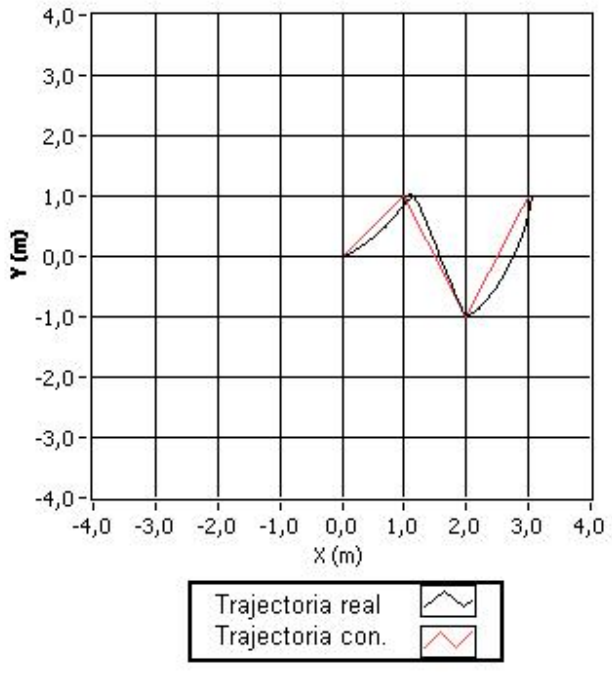

Figura 97. Resultats de la prova 1

La Taula 19 detalla el seguit de consignes que s'han donat al robot i els corresponents errors de posició que s'han assolit.

Consigna x (m)	Consigna y (m)	Consigna σ (º)	Error posició (m)	Error σ (º)
1	1	90	0.06	≈ 0
2	-1	180	0.08	≈ 0
3	1	270	0.07	≈ 5

Taula 19. Consignes de la prova 1

9.4. Prova 2

Aquesta prova s'ha realitzat en les mateixes condicions que en el cas anterior però en aquest cas no s'ha activat la consigna d'angle final.

La Figura 98 mostra l'evolució de la posició del robot segons els càlculs d'odometria.

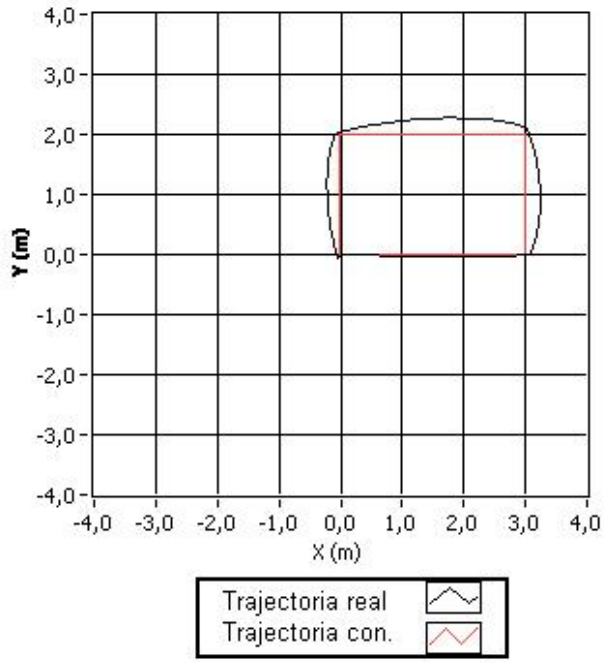

Figura 98. Resultats de la prova 2

La Taula 20 torna a detallar el seguit de consignes donades i els errors mesurats.

Consigna x (m)	Consigna y (m)	Error posició (m)
3	0	0.04
3	2	0.03
0	2	0.04
0	0	0.02

Taula 20. Consignes de la prova 2

Es pot observar com en aquest cas els errors son menors degut a no fer servir el control d'angle final. Així es recomana no fer servir aquesta opció a no ser que sigui necessari. L'augment de l'error amb el control d'angle final es deu a que aquest s'assoleix fent girar el robot sobre el seu propi eix, aquesta acció fa patinar fàcilment el robot portant així a errors.

9.5. Prova 3

Aquesta prova s'ha realitzat igualment sense el control d'angle final i sense control de velocitat, en aquest cas però el PWM treballa amb el mode de gir lliure activat.

La Figura 99 torna a mostrar els resultats obtinguts per les diferents consignes.

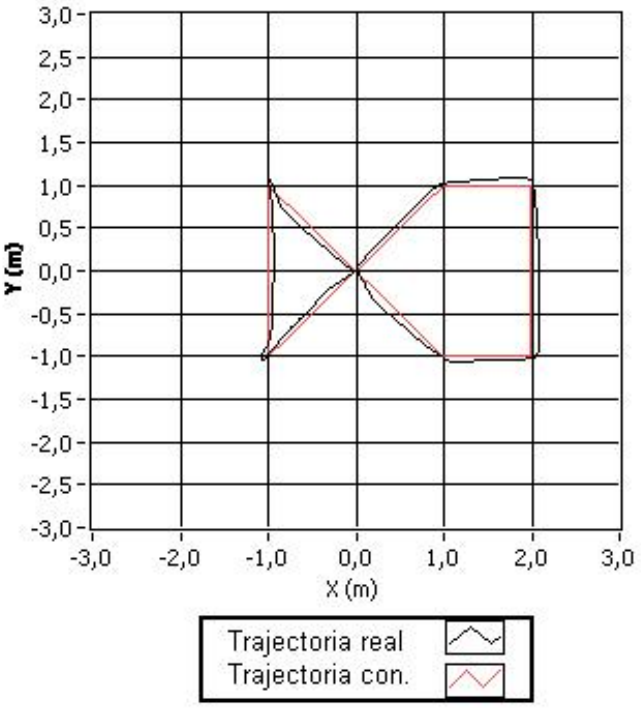

Figura 99. Resultats de la prova 4

La Taula 21 detalla el seguit de consignes donades i els errors mesurats. En aquest cas s'observa com no hi ha diferència pel que fa la precisió canviant el mode de funcionament del PWM.

Consigna x (m)	Consigna y (m)	Error posició (m)
1	-1	0.01
2	-1	0.02
2	1	0.01
1	1	0.03
-1	-1	0.02
-1	1	0.04
0	0	0.04

Taula 21. Consignes de la prova 3

9.6. Prova 4

Amb aquesta última prova es mostren els resultats obtinguts treballant amb el control de velocitat PID. La Figura 100 mostra el recorregut del robot.

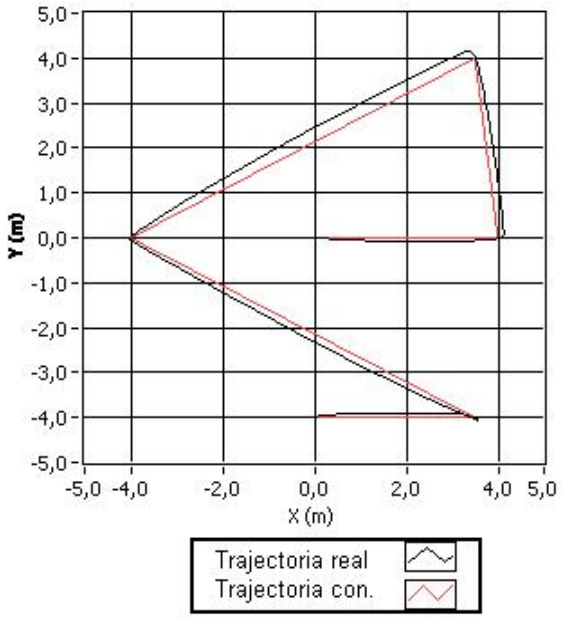

Figura 100. Resultats de la prova 4

Tal com es pot observar en la Taula 22 en aquest cas tampoc s'observen diferències significatives pel que fa la precisió del sistema.

Consigna x (m)	Consigna y (m)	Error posició (m)
4	0	0.02
3.5	4	0.01
-4	0	0.03
3.5	-4	0.04
0	-4	0.04

Taula 22. Consignes de la prova 4

10. RESUM DEL PRESSUPOST

El pressupost total per dur a terme el present projecte inclou les hores de treball i tot el material necessari per l'execució. També s'inclou el cost resultant d'elaborar i muntar els diversos fotolits dels circuits dissenyats.

No obstant no s'inclou en el pressupost el cost d'adquisició del robot Wifibot. Així mateix el programador necessari per programar el PIC16F876 del programador i depurador ICD2 tampoc s'inclou donat que serà necessari només puntualment.

El pressupost total ascendeix a divuit mil vuit-cents trenta quatre euros amb vint-i-dos cèntims, IVA no inclòs.

11. CONCLUSIONS

En vista dels objectius presentats inicialment per aquest projecte s'han complert tots. S´ha analitzat en detall l'arquitectura del robot i s'ha documentat detalladament. A partir d'aquest anàlisi s'ha arribat a la conclusió que és pot augmentar notablement la capacitat del robot si s'efectuen modificacions en el hardware.

Les modificacions dissenyades i aplicades al robot han augmentat les seves capacitats pel que fa al control, seguretat, fiabilitat i camp d'aplicació.

En quant al sistema de localització i control de posició presentat com a objectiu principal d'aquest projecte s'ha proporcionat un sistema base fiable i amb bona precisió. El control per odometria implementat en els dsPIC's compleix amb les expectatives proposades amb els resultats obtinguts.

Aquesta base s'ha dissenyat amb la perspectiva de poder incorporar nous sistemes complementaris capaços d'interactuar amb el sistema existent, d'aquesta manera es podran combinar diferents tecnologies i sistemes augmentant les capacitats i la precisió d'actuació.

Aquest projecte recull tota la documentació i les especificacions tant de hardware com de software sobre l'estat actual del robot. El conjunt de manuals descriuen totes les eines necessàries per poder realitzar futurs projectes amb el robot.

Tot i que les perspectives d'aquest projecte pròpiament s'han complert, es plantegen noves perspectives i propostes que permetrien millorar les prestacions del robot.

11.1. Perspectives immediates

A curt termini es proposa incorporar sistemes d'acceleració i frenada del robot durant el control de posició. Aquest sistema s'hauria d'incorporar a la programació dels dsPIC's. D'aquesta manera en aquest nivell de control s'esgotarien mes o menys les capacitats.

Per al nivell de control del cub es proposa el disseny d'aplicacions per tal de poder prendre decisions en quant a evitar obstacles en la trajectòria. Aquesta és una necessitat bàsica per tal de donar autonomia al robot.

11.2. Treball Futur

A més llarg termini es plantegen les aplicacions del nivell mes alt. Donada l'intenció del projecte global que inclou l'adquisició de nous robots es proposa implementar el treball en equip.

Els nous robots adquirits pel projecte tindran arquitectures i capacitats diferents i òbviament també tindran les seves limitacions. L'implementació del treball en equip en aquest nivell permetrà la combinació de les capacitats de cada robot augmentat l'efectivitat global.

La idea global es que les observacions fetes per un robot serveixin com a ajuda a tota la resta de l'equip de forma que un robot que es trobi davant na limitació pugui passar la tasca a un altre capaç de superar la barrera.

Pel que fa al disseny del circuit de control aquest es podrà adaptar per funcionar en altres robots. Donat que acúa en el nivell més baix es podrà aplicar a qualsevol robot de quatre rodes. Donada l'arquitectura simètrica del circuit també es pot adaptar sense gaires canvis per funcionar en robots de més de quatre rodes, simplement es tracta d'afegir un dsPIC per cada parella de rodes.

Rafael Hesse
Enginyer Tècnic Industrial en Electrònica

Girona, 20 de juny de 2008

12. RELACIÓ DE DOCUMENTS

Aquest projecte es conforma per cinc documents independents.

El primer i més important document és la memòria, seguit en aquest es troben els plànols, el plec de condicions, l'estat d'amidaments i finalment el pressupost.

13. BIBLIOGRAFIA

AVR and Robotics. Technical Papers
(http://www.barello.net/Papers/Motion_Control/index.htm, 12 de març de 2008)

Cooperative CoLinux. Guia d'instal·lació
(http://colinux.wikia.com/wiki/Getting_Started_with_coLinux_-_Long_manual, 18 de novembre de 2007)

Cooperative CoLinux. Homepage
(http://www.colinux.org, 14 d'octubre de 2007)

Gentoo Linux Wiki. Manual de compilació creuada
(http://gentoo-wiki.com/HOWTO_Cross_Compile, 13 de novembre de 2007)

Gentoo. Guia d'instal·lació
(http://www.gentoo.org/doc/es/handbook/handbook-ppc.xml?part=1&chap=1, 13 de novembre de 2007)

Gentoo. Handbooks
(http://www.gentoo.org/doc/en/handbook/index.xml, 12 de desembre de 2007)

Lynxmotion. Documentació sobre els motors i les rodes
(http://www.lynxmotion.com, 2 de març de 2008)

Microchip. 10-Bit A/D Converter - dsPIC30F FRM
(http://ww1.microchip.com/downloads/en/DeviceDoc/70064D.pdf, 18 de gener de 2008)

Microchip. Datasheet del dsPIC30f2010
(http://ww1.microchip.com/downloads/en/DeviceDoc/70118G.pdf, 19 d'octubre de 2007)

Microchip. DsPIC30F Family reference Manual
(http://ww1.microchip.com/downloads/en/DeviceDoc/70046E.pdf, 18 de gener de 2008)

Microchip. Fe d'errates del datasheet del dsPIC30f2010
(http://ww1.microchip.com/downloads/en/DeviceDoc/80353A.pdf, 19 d'octubre de 2007)

Microchip. Fe d'errates Silicon del dsPIC30f2010 ver. A0
(http://ww1.microchip.com/downloads/en/DeviceDoc/80178f.pdf, 19 d'octubre de 2007)

Microchip. Inter-Integrated Circuit (I2C) - dsPIC30F FRM
(http://ww1.microchip.com/downloads/en/DeviceDoc/70068E.pdf, 18 de gener de 2008)

Microchip. Interrupts - dsPIC30F FRM
(http://ww1.microchip.com/downloads/en/DeviceDoc/70053D.pdf, 18 de gener de 2008)

Microchip. Motor Control PWM - dsPIC30F FRM
(http://ww1.microchip.com/downloads/en/DeviceDoc/70062E.pdf, 18 de gener de 2008)

Microchip. MPLAB IDE
(http://www.microchip.com, 12 de gener de 2008)

Microchip. Quadrature Encoder Interface (QEI) - dsPIC30F FRM
(http://ww1.microchip.com/downloads/en/DeviceDoc/Section%2016.%2070063c.pdf, 18 de gener de 2008)

Microchip. Timers - dsPIC30F FRM
(http://ww1.microchip.com/downloads/en/DeviceDoc/70059D.pdf, 18 de gener de 2008)

Microchop. Fe d'errates Silicon del dsPIC30f2010 ver. A1
(http://ww1.microchip.com/downloads/en/DeviceDoc/80186G.pdf, 19 d'octubre de 2007)

Philips Semiconductors. Datasheet del conersor AD
(http://www.nxp.com/acrobat_download/datasheets/PCF8591_6.pdf, 19 d'octubre de 2007)

Seattle Robotics Society. Using a PID-based Technique for Competitive Odometry and Dead-Reckoning (http://www.seattlerobotics.org/encoder/200108/using_a_pid.html, 12 de març de 2008)

Sharp. Datasheet del sensor de distància
(http://document.sharpsma.com/files/gp2y0a02yk_e.pdf, 2 de març de 2008)

ST Microelectronics. Datasheet del driver de motors L298
(http://www.st.com/stonline/products/literature/ds/1773/l298.pdf, 19 d'octubre de 2007)

Stolz, Lothar. ICD2 Clone
(http://stolz.de.be/icd/main.html, 12 de gener de 2008)

US Digital. Datasheet de l'encoder òptic E4P
(http://www.usdigital.com/data-sheets/E4P%20Data%20Sheet.pdf, 4 de febrer de 2008)

Wifibot. Download center
(http://www.wifibot.com/page9.php, 16 de gener de 2008)

14. GLOSSARI

DIP: (Dual Inline Pins). Tipus d'encapsulat per circuits integrats en que les connexions es disposen en dues fileres paral·leles i simètriques de pins.

DHCP: (Dinamic Host Configuration Protocol) Sistema desenvolupat per Microsoft que permet assignar a un dispositiu una nova adreça IP cada vegada que aquest es connecta a una xarxa. Aquesta adreça també pot canviar un cop el dispositiu es troba connectat a la xarxa.

Direcció IP: Part del protocol IP que defineix les adreces en un format de 32 dígits binaris.

DNS: (Domain Name System) Servei d'internet que permet traduir els noms de dominis alfanumèrics en adreces IP per tal de poder establir connexions.

Driver: En electrònica. Dispositiu capaç de controlar càrregues d'alt consum a partir d'una senyal de control dèbil.

dsPIC: (Digital Signal Programmable Interrupt Controller) Microcontrolador que usualment s'encarrega de la comunicació entre el processador i els perifèrics.

Duty Cycle: D'una senyal periòdica es refereix a la proporció del temps que aquesta es troba en estat actiu.

Ethernet: Sistema de comunicació LAN més conegut. Existeixen 3 modalitats diferents, 10 Mbps (Traditional ethernet), 100Mbps (Fast ethernet) i 1.000Mbps (Gigabit ethernet)

Gateway: Sistema d'enllaç que permet comunicació entre dos sistemes de diferent tipologia

Host device: Dispositiu electrònic amb port de comunicacions que permet controlar altres dispositius

I^2C: (Inter-Integrated Circuit). Bus de comunicació bifilar dissenyat i patentat per Philips. Destinat a la comunicació entre controladors i memòries electròniques.

IP: (Internet Protocol) Protocol estàndard que gestiona les comunicacions per internet o xarxes locals així com l'adreçament dels dispositius.

JAVA: Llenguatge de programació dissenyat especialment per descarregar de forma segura aplicacions des de internet.

LAN: (Local Area Network) Xarxa de tipologia ethernet limitada a una àrea relativament petita.

Perl: (Extraction and Reporting Language). Llenguatge de programació per Unix especialment dissenyat per comunicacions.

Pin: Referència a la connexió d'un terminal d'un circuit integrat

PWM: (Pulse-Width-Modulated). Tecnologia utilitzada per controlar la velocitat de motors de corrent contínua mitjançant modulació de l'ample de polsos en l'alimentació.

Router: Dispositiu que s'encarrega de gestionar la transmissió de paquets entre xarxes

Target device: Dispositiu electrònic controlat per un Host device

WiFi: (Wireless Fidelity). Tecnologia de comunicació inalàmbrica per xarxes informàtiques.

ANNEX A. CODI FONT DELS PROGRAMES DISSENYATS

A continuació es detalla el codi font per les diverses aplicacions dissenyades per aquest projecte.

A.1. Programes per al robot original

Els codis detallats a continuació requereixen l'estructura de hardware original del robot, aquests programes s'han dissenyat per tal d'explorar les possibilitats de control del robot

A.1.1. Programa "simple_server_ADC"

```
////////////////////////////////////////////////////////////////////////
// Nom del programa:    simple_server_main.c                          //
// Plataforma:          Nylon Linux del Wifibot                       //
// Paràmetre d'entrada: Cap                                           //
// Autor:               Rafael Hesse (modificació)                    //
// Data:                12/12/2007 (modificació)                      //
// Versió:              1.1                                           //
// Compilador:          GCC cross compiler mipsel-linux-gcc           //
// Dependències:        Hardware i software del Wifibot i els dsPIC's  //
// Comentaris:          Es tracta del codi que corre de forma original //
//                      en el robot Wifibot. Respecte al codi original //
//                      el funcionament és idèntic i només s'han       //
//                      optimitzat algunes operacions així com extret  //
//                      comandes sense funcionalitat.                 //
////////////////////////////////////////////////////////////////////////

//INCLUDES
#include "wifibot.h"

// MAIN: PROGRAMA PRINCIPAL //
int main(int argc, char *argv[])
{
buffso_send[0]=0;       // Nivell de bateria
buffso_send[1]=0;       // Encoder davant esquerra
buffso_send[2]=0;       // Encoder darrera esquerra
buffso_send[3]=0;       // Encoder davant dreta
buffso_send[4]=0;       // Encoder darrera dreta
buffso_send[5]=0;       // Infraroig esquerra
buffso_send[6]=0;       // Infraroig dreta
bufad[0]=0;             // Buffer intermig pel conversor AD
printf("running....\n");
initI2c();              // Inicialització de les funcions I2C

//Inicialitzem el fil que funcionarà a mode de "watchdog"
if ((ret = pthread_create (& thread [0], NULL, fn_thread_dog, (void *)
      0)) != 0) exit (1);
      // si ret = 0 el fil s'ha creat correctament
      //Inicialitzem el fil que gestionarà les comunicacions
if ((ret = pthread_create (& thread [1], NULL, fn_thread_com, (void *)
```

```
      1)) != 0) exit (1);
      // si ret = 0 el fil s'ha creat correctament
      // Definim una variable del tipus estructura sigaction
struct sigaction action;
// El manejador de de les senyals capturades es la funcio gestionaire
action . sa_handler = gestionnaire;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (action . sa_mask));
// Tots els flags desactivats
action . sa_flags = 0;
// Capturem la senyal SIGQUIT causada per premer les tecles control
if (sigaction (SIGQUIT, & action, NULL) != 0) exit (1);
// El manejador de de les senyals capturades es la funcio gestionaire
action . sa_handler = gestionnaire;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (action . sa_mask));
// Aquests flags permeten reiniciar el programa despres de l'interrupcio
action . sa_flags = SA_RESTART | SA_RESETHAND;
// Capturem la senyal SIGINT causada per premer Ctrl + c
if (sigaction (SIGINT, & action, NULL) != 0) exit (1);
while ( 1 )
{
sleep(5);
}
// main queda en suspensió fins que el fil acabi o es suspengui
pthread_join (thread [0], NULL);
// main queda en suspensió fins que el fil acabi o es suspengui
pthread_join (thread [1], NULL);
return 0;
}
// Fil [0]
// Aquest fil supervisa l'execució del fil [2], la variable dog augmenta
// un cop per segon, si arriba a valer 3 es para tot el procés
void * fn_thread_dog (void * num)
{
int numero = (int) num;
while (1)
{
sleep (1);                          // Espera un segon
pthread_mutex_lock (& mutex_dog);   // Reservem la variable dog
dog ++;                             // Augmentem el contador
pthread_mutex_unlock (& mutex_dog); // Es suspen la reserva
if (dog>3)                          // S'ha superat el temps
{
dog=0;
if (connected)                      // Ens desconectem del client
{
connected=0;
stop();
                                    // Cancelem l'execució del fil [1]
int ret=pthread_cancel (thread[1]);
close(clntSock);                    // Tanquem el socket de client
close(servSock);                    // Tanquem el socket de serv.
sleep(2);                           // Esperem 2 segons i reiniciem
if ((ret = pthread_create (& thread [1], NULL,
fn_thread_com, (void *) 1)) != 0) exit (1);
}
}
}
pthread_exit (NULL);
}
```

```
// Fil de comunicacions
void * fn_thread_com (void * num)
{
int numero = (int) num;
int comptador = 0;
// Creem el socket servidor per atendre les peticions entrents,
//alhora comprovem si ha donat error (cas per al qual retorna -1)
servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
// PF_INET = domini de la comunicació
// SOCK_STREAM = transmissió de "streams"
// IPPROTO_TCP = Protocol de comunicació TCP/IP
// Construim una estructura que contindrà la adreça local
// Omplim de zeros la estructura
memset(&echoServAddr, 0, sizeof(echoServAddr));
// Familia de direccions de internet de qualsevol tipus
echoServAddr.sin_family = AF_INET;
// IP qualsevol
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
// Port local de comunicació (per defecte 15000)
echoServAddr.sin_port = htons(echoServPort);
// Relacionem el socket amb una direcció local
int autorisation=1;
// Autoritzem les peticions de comunicació entrants
// Configura les opcions del socket
setsockopt(servSock,SOL_SOCKET,SO_REUSEADDR,&autorisation,sizeof(in
t));
// lliguem el socket amb la adreça local
bind(servSock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr));
for (;;)          // Bucle infinit
{
      printf("\n%d: Connexió estable...\n", comptador);
      comptador +=1;
      // Posem el socket servidor a la escolta
      listen(servSock, MAXPENDING);
      // Definim la longitud dels missatges a rebre
      clntLen = sizeof(echoClntAddr);
      // Ens posem a la espera de rebre un misatge del client
      clntSock = accept(servSock, (struct sockaddr *)
            &echoClntAddr, &clntLen);
      // En el moment en que s'estableix la comunicació fem:
      do
      {
      // Si el missatge no te la longitud esperada donem un error
      if ((recvMsgSize = recv(clntSock, buffso_rcv, 2, 0)) < 1)
      {
            shutdown(clntSock,1);
      }
      else  // Si el missatge es correcte proseguim a tractar-lo
      {
            printf("\nHem rebut %s",buffso_rcv);
            // Reservem la variable dog per aquest fil
            pthread_mutex_lock (& mutex_dog);
            connected=1;
            // Posem a 0 el comtador dog
            dog =0;
            // Alliberem la variable dog
            pthread_mutex_unlock (& mutex_dog);
            getAD();          // Rutina del conversor AD
            setI2cL();        // Enviem dades al motor esquerra
            getI2cL();        // Rebem dades del encoder esquerra
```

```
            setI2cR();        // Enviem dades al motor dret
            getI2cR();        // Rebem dades del encoder dret
            // Enviem dades al client conectat
            send(clntSock,buffso_send,7,0);
      }
      }
while(recvMsgSize>0);
connected=0;
shutdown(clntSock,1);
close(clntSock);
sleep(1);
}//end for(;;)
pthread_exit (NULL);
}//end thread

//////////////////////////
//WIFIBOT I2C Functions///
//////////////////////////

// Inicialització del bus I2C
void initI2c()
{
      sprintf(filename3,"/dev/i2c/%d",i2cbus);
      file = open(filename3,O_RDWR);
}

// Enviem la consigna de velocitat del motor esquerra
void setI2cL()
{
      bufset[0]=buffso_rcv[0];
      bufset[1]=buffso_rcv[0];
      ioctl(file,I2C_SLAVE,0x51);
      write(file,bufset,2);
}

// Rebem el valor del encoder esquerra i del infraroig esquerra
void getI2cL()
{
      read(file,bufget,3);
      buffso_send[1]=bufget[0];
      buffso_send[2]=bufget[1];
      buffso_send[5]=bufget[2];
}

// Enviem la consigna de velocitat al motor dret
void setI2cR()
{
      bufset[0]=buffso_rcv[1];
      bufset[1]=buffso_rcv[1];
      ioctl(file,I2C_SLAVE,0x52);
      write(file,bufset,2);
}

// Rebem el valor del encoder dret i del infraroig dret
void getI2cR()
{
      read(file,bufget,3);
      buffso_send[3]=bufget[0];
      buffso_send[4]=bufget[1];
      buffso_send[6]=bufget[2];
}
```

```
// Configurem el conversor AD i llegim el valor de bateria
void getAD()
{
      bufset[0]=0x40;
      bufset[1]=250;//DAC
      ioctl(file,I2C_SLAVE,0x48);
      write(file,bufset,2);
      read(file,bufad,1);
      buffso_send[0]=bufad[0]/2;
}

// RUTINA PER PARAR COMPLETAMENT EL ROBOT INDEPENDENMENT DE LES CONSIGNES
void stop()
{
      bufset[0]=0;
      bufset[1]=0;
      ioctl(file,I2C_SLAVE,0x51);
      write(file,bufset,2);
      ioctl(file,I2C_SLAVE,0x52);
      write(file,bufset,2);
}

// RUTINA PER GESTIONAR LES SENYALS CAPTURADES
void gestionnaire (int numero)
{
switch (numero)
{
      //Qualsevol senyal capturada provocarà que es talli el
      //programa i es pari el robot
      case SIGQUIT :
      stop();
      break;
      case SIGINT :
      stop();
      break;
}
}
```

A.1.2. Makefile programa “simple_server_ADC”

```
#############################################################################
## Nom del programa:    Makefile                                           ##
## Plataforma:          Gentoo linux                                       ##
## Autor:               Rafael Hesse                                       ##
## Data:                12/12/2007                                         ##
## Versió:              1.0                                                ##
## Compilador:          GCC cross compiler mipsel-linux-gcc                ##
## Dependències:        Cap                                                ##
## Comentaris:          Directrius per compilar el programa bateria.c      ##
#############################################################################

CC =/stuff/tmp/cross/bin/mipsel-linux-gcc
CFLAGS =-c -Wall
simple_server_objects = simple_server_main.o

## Compila i envia per ethernet al robot
all_lan :         simple_server send_lam clean
## Compila i envia per Wifi al robot
all_wifi :        simple_server send_wifi clean
## Només compila
simple_server:    $(simple_server_objects)
                  $(CC) -o simple_server_ADC $(simple_server_objects) -
                  lpthread
simple_server_main.o:   simple_server_main.c
                        $(CC) -c simple_server_main.c
## Envia per ethernet
send_lan:
            scp ./simple_server_ADC root@192.168.0.103:./
## Envia per Wifi
send_wifi:
            scp ./simple_server_ADC root@192.168.1.103:./
## Neteja
clean:
            rm -f *.o simple_server
```

A.1.3. Programa “bateria”

```
////////////////////////////////////////////////////////////////////////
// Nom del programa:    bateria.c                                     //
// Plataforma:          Nylon Linux del Wifibot                       //
// Paràmetre d'entrada: Cap                                           //
// Autor:               Rafael Hesse                                  //
// Data:                08/02/2008                                    //
// Versió:              1.0                                           //
// Compilador:          GCC cross compiler mipsel-linux-gcc           //
// Dependències:        Cap                                           //
// Comentaris:          Retorna l'estat de les bateries en volts      //
//                      El primer valor retornat després d'un reset del//
//                      conversor no és vàlid donat al retard per     //
//                      estabilitzar-se del cristall.                 //
////////////////////////////////////////////////////////////////////////

//INCLUDES
#include "wifibot2.h"

// Definicio de les funcions:
void initI2c();
static void getAD();                //Llegir I2C estat de la bateria
void gestio_int (int numero);       //Gestio de les interrupcions

// Definició de variables globals
static unsigned char bufad[1];      //Buffer de lectura per al bus I2C
static unsigned char bufset[2];     //Buffer per la configuració del
                                    //conversor A/D
int bateria;

// MAIN: PROGRAMA PRINCIPAL //
int main(int argc, char *argv[])
{
      bufset[0]=bufset[1]=0;        //Inicialització del buffer
      bateria=0;                    //Inicialització de la variable
      initI2c();                    //Inicialització de les funcions I2C
      getAD();                      //Accedim al bus
      printf("\nEl nivell de bateria es de:\n %d.%d
            volts\n\n",bateria/10, bateria%10);
}

// FUNCIONS DEL BUS I2C

// Inicialització del bus I2C
void initI2c()
{
      sprintf(filename3,"/dev/i2c/%d",i2cbus);
      if ((file = open(filename3,O_RDWR)) < 0) printf("***Error al
            iniciar I2C\n");
}
//Comunicació amb el conversor AD
void getAD()
{
      bufset[0]=0x40; //0 1 0 0 0 0 0 0 byte 1 de configuració
      bufset[1]=250; //1 1 1 1 1 0 1 0 byte 2 de configuració
      if (ioctl(file,I2C_SLAVE,0x48) < 0) printf("***Error al iniciar I2C
            esclau en conversor A/D\n");
```

```
	//Enviem els dos bytes de configuració
	if (write(file,bufset,2) !=2)
	{
		printf("***Error al escriure en I2C conversor A/D \n");
	}
	//Llegim el bus I2C
	if (read(file,bufad,1) !=1)
	{
		printf("***Error al llegir I2C en conversor A/D\n");
	}
	//Si tot ha anat bé procedim a tractar el valor
	else
	{
		bateria=bufad[0];
		/*******************************************************
		Conversió a volts * 10
                                  lectura * 5     lectura * 25
		Vbat = Vain*2.5 = 2.5 * ------------- = --------------
                                    255              51
		*******************************************************/
		bateria=25*bateria/51;
}
}
```

A.1.4. Makefile programa “bateria”

```
#############################################################################
## Nom del fitxer:      Makefile                                           ##
## Plataforma:          Gentoo linux                                       ##
## Autor:               Rafael Hesse                                       ##
## Data:                08/02/2008                                         ##
## Versió:              1.0                                                ##
## Compilador:          GCC cross compiler mipsel-linux-gcc                ##
## Dependències:        Cap                                                ##
## Comentaris:          Directrius per compilar el programa bateria.c      ##
#############################################################################

CC =/stuff/tmp/cross/bin/mipsel-linux-gcc
CFLAGS=-c -Wall

objects = bateria.o

## Compila i envia per ethernet al robot
all_lan :   bateria send_lan clean
## Compila i envia per Wifi al robot
all_wifi :  bateria send_wifi clean
## Només compila
bateria:    $(objects)
            $(CC) -o bateria $(objects) -lpthread -lm

bateria.o:  bateria.c
            $(CC) -c bateria.c
## Envia per ethernet
send_lan:
            scp ./bateria root@192.168.0.103:./
## Envia per Wifi
send_wifi:
            scp ./bateria root@192.168.1.103:./
## Neteja
clean:
            rm -f *.o
```

A.1.5. Programa “recte”

```
////////////////////////////////////////////////////////////////////
// Nom del programa:    recte.c                                   //
// Plataforma:          Nylon Linux del Wifibot                   //
// Paràmetre d'entrada: Distància a recorrer de -32768 cm a +32768 cm  //
// Autor:               Rafael Hesse                              //
// Data:                08/02/2008                                //
// Versió:              1.0                                       //
// Compilador:          GCC cross compiler mipsel-linux-gcc       //
// Dependències:        Harware i software original del robot     //
// Comentaris:          Fa desplaçar a velocitat constant el robot la  //
//                      distància marcada com a consigna i es para     //
////////////////////////////////////////////////////////////////////

//INCLUDES
#include "wifibot2.h"
#include <tgmath.h>
#include <sys/time.h>

// Definicio de les funcions:
static void initI2c();        // Inicialització del bus I2C
static void setI2cL();        // Llegir I2C del DSPIC esquerra
static void getI2cL();        // Escriure I2C en el DSPIC esquerra
static void setI2cR();        // Llegir I2C del DSPIC Dret
static void getI2cR();        // Escriure I2C en el DSPIC dret
static void stop();           // Parar el robot
void gestio_int (int numero); // Gestio de les interrupcions

//DEFINICIÓ DE LES VARIABLES GLOBALS
long x;           // Posició en l'eix X en milèsimes de milimetre
long y;           // Posició en l'eix Y en milèsimes de milímetre
long w;           // Posició angular en milèsimes de radiants
long w_D;         // Velocitat angular roda dreta en mil.rad/s
long w_E;         // Velocitat angular roda esq. en mil.rad/s
int sentit_D;     // Sentit rodes dreta (1=endavant, -1=enrrera)
int sentit_E;     // Sentit rodes esquerra (1=endavant, -1=enrrera)
static unsigned char ordre[2];  // Consignes que enviem als PIC's
                                // [0]=Costat esq., [1]=Costat dret
static unsigned char bufset[2]; // Buffer d'escriptura per al bus I2C
                                // [0]=Costat esq., [1]=Costat dret
static unsigned char bufget[3]; // Buffer de lectura per al bus I2C
                                // [0]=[1]=Encoder, [2]=sensor dist.
int tics_E;       // Velocitat llegida en tics/40ms costat esquerra
int tics_D;       // Velocitat llegida en tics/41ms costat dret

// MAIN: PROGRAMA PRINCIPAL //
int main(int argc, char *argv[])
{
// Definició de variables locals
struct itimerval val;         // Variable per la configuració del timer
struct sigaction interrupcio; // Variable configuració d'interrupcions
int marxa;                    // Marcador de la seqüència de desplaçament
int v1;                       // Consigna velocitat per girar endavant
int v2;                       // Consigna de velocitat per girar enrera
int cons_x;                   // Consigna de posició 'X' rebuda en cm
// Inicialització de variables locals
marxa=1;
```

```
v1=25;
v2=25+64;
cons_x = atoi(argv[1]);       // Agafem el paràmetre d'entrada 'X'
// Inicialització de variables globals
x=0;
y=0;
w=0;
w_D=0;
w_E=0;
sentit_D=1;
sentit_E=1;
ordre[0]=ordre[1]=0;
bufset[0]=bufset[1]=0;
bufget[0]=bufget[1]=bufget[2]=0;
tics_E=0;
tics_D=0;
// Configurem el timer per que doni una alarma cada 50 ms
val.it_interval.tv_sec=0;
val.it_interval.tv_usec=50000;
val.it_value.tv_sec=0;
val.it_value.tv_usec=50000;
setitimer(ITIMER_REAL, &val, NULL);
cons_x=cons_x*10000;    // Passem la consigna a mil. de milímetre
initI2c();              // Inicialització de les funcions I2C
// El manejador de de les senyals capturades es la funcio gestio_int
interrupcio . sa_handler = gestio_int;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (interrupcio . sa_mask));
// Tots els flags desactivats
interrupcio . sa_flags = 0;
// Capturem la senyal SIGQUIT causada per premer les tecles control
if (sigaction (SIGQUIT, & interrupcio, NULL) != 0) exit (1);
// El manejador de de les senyals capturades es la funcio gestio_int
interrupcio . sa_handler = gestio_int;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (interrupcio . sa_mask));
// Aquests flags permeten reiniciar el programa despres de l'interrupcio
interrupcio . sa_flags = SA_RESTART | SA_RESETHAND;
// Capturem la senyal SIGINT causada per premer Ctrl + c
if (sigaction (SIGINT, & interrupcio, NULL) != 0) exit (1);
// El manejador de de les senyals capturades es la funcio gestio_int
interrupcio . sa_handler = gestio_int;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (interrupcio . sa_mask));
// Tots els flags desactivats
interrupcio . sa_flags = 0;
// Capturem la senyal SIGALRM causada pel timer cada 50ms
if (sigaction (SIGALRM, & interrupcio, NULL) != 0) exit (1);
while (marxa)
{
      if (cons_x>0 && x<cons_x)     // Es demana desplaçament endavant
      {
            ordre[0]=v2;
            ordre[1]=v2;
      }
      else if (cons_x<0 && x>cons_x)// Es demana gir desplaçament enrrera
      {
            ordre[0]=v1;
            ordre[1]=v1;
      }
      else                          // Hem assolit la consigna i parem
```

```
	{
		stop();
		marxa=0;
	}
}
return 0;
}

// FUNCIONS DEL BUS I2C

// RUTINA PER INICIAR EL BUS I2C
void initI2c()
{
	sprintf(filename3,"/dev/i2c/%d",i2cbus);
	if ((file = open(filename3,O_RDWR)) < 0) printf("***Error al
		iniciar I2C\n");
}

// RUTINA PER ENVIAR DADES AL dsPIC DE LA ESQUERRA
void setI2cL()
{
	bufset[0]=ordre[0];
	bufset[1]=ordre[0];
	if (ioctl(file,I2C_SLAVE,0x51) < 0) printf("***Error al iniciar I2C
		esclau en DSPIC esquerra\n");
	if (write(file,bufset,2) !=2)
	{
		printf("***Error al escriure en I2C DSPIC esquerra \n");
	}
}

// RUTINA PER REBRE DADES DEL dsPIC DE LA ESQUERRA
void getI2cL()
{
	if (read(file,bufget,3) !=3)
	{
		printf("***Error al llegir I2C en DSPIC esquerra\n");
	}
	else
	{
		tics_E=bufget[0];
	}
}

// RUTINA PER ENVIAR DADES AL dsPIC DE LA DRETA
void setI2cR()
{
	bufset[0]=ordre[1];
	bufset[1]=ordre[1];
	if (ioctl(file,I2C_SLAVE,0x52) < 0) printf("***Error al iniciar I2C
		esclau en DSPIC dret\n");
	if (write(file,bufset,2) !=2)
	{
		printf("***Error al escriure en I2C DSPIC dret \n");
	}
}

// RUTINA PER REBRE DADES DEL dsPIC DE LA DRETA
void getI2cR()
{
	if (read(file,bufget,3) !=3)
```

```
	{
		printf("***Error al llegir I2C en DSPIC dret\n");
	}
	else
	{
		tics_D=bufget[0];
	}
}

// RUTINA PER PARAR EL ROBOT
void stop()
{
	ordre[0]=ordre[1]=0;
	bufset[0]=0;
	bufset[1]=0;
	ioctl(file,I2C_SLAVE,0x51);
	write(file,bufset,2);
	ioctl(file,I2C_SLAVE,0x52);
	write(file,bufset,2);
}

// RUTINA PER GESTIONAR LES INTERRUPCIONS
void gestio_int (int numero)
{
	switch (numero)
	{
	case SIGQUIT :
		stop();
		break;
	case SIGINT :
		stop();
	break;
	case SIGALRM :
		setI2cL(); // Enviem dades al motor esquerra
		getI2cL(); // Rebem dades del encoder esquerra
		setI2cR(); // Enviem dades al motor dret
		getI2cR(); // Rebem dades del encoder dret
		if (ordre[0] !=0 || ordre[1] !=0) // Determinem sentit gir
		{
			if(ordre[1]>=128)
			{
				if(ordre[1]>=192) sentit_D=1;
				else sentit_D=-1;
			}
			else
			{
				if(ordre[1]>=64) sentit_D=1;
				else sentit_D=-1;
			}
			if(ordre[0]>=128)
			{
				if(ordre[0]>=192) sentit_E=1;
				else sentit_E=-1;
			}
			else
			{
				if(ordre[0]>=64) sentit_E=1;
				else sentit_E=-1;
			}
		}
		// Filtrem valors dels encoders impossibles
```

```
            if((tics_D<<6)<5000 && (tics_E<<6)<5000)
            {
                  /*************************************
                  velocitat angular:
                        tics    1000ms    2*pi rad
                  w = ----- * ------- * -----------
                        41ms      1s      480 tics
                  *************************************/
                  // velocitat angular dreta en mili_rad/s
                  w_D=((tics_D<<6)*327.249*sentit_D)/50;
                  // velocitat angular esquerra en mili_rad/s
                  w_E=((tics_E<<6)*327.249*sentit_E)/50;

                  /*************************************
                  Posició angular:
                              w1 - w2
                  Wn = Wn-1 + --------- * t
                                 B
                        on t = increment de temps (50ms)
                           B = distància entre rodes
                  *************************************/
                  // posició angular en mili_rad
                  w=w+((((w_D-w_E)*62)/340)*0.05);

                  /*************************************
                  Posició cartesiana:
                              (w1 + w2) * r
                  Xn = Xn-1 + --------------- * cos(Wn)
                                    2
                              (w1 + w2) * r
                  Yn = Yn-1 + --------------- * sin(Wn)
                                    2
                        on r = radi roda
                  *************************************/
                  // posicio x en milesimes de mm
                  x=x+(((w_D+w_E)*31*0.05)*cos(w/1000));
                  // posicio y en milesimes de mm
                  y=y+(((w_D+w_E)*31*0.05)*sin(w/1000));
            }
      // Avisem si hem rebut dades erronies
      else printf("+++++++++++++++ error ++++++++++\n");
      break;
      }
}
```

A.1.6. Makefile programa “recte”

```
#############################################################################
## Nom del programa:    Makefile                                           ##
## Plataforma:          Gentoo linux                                       ##
## Autor:               Rafael Hesse                                       ##
## Data:                08/02/2008                                         ##
## Versió:              1.0                                                ##
## Compilador:          GCC cross compiler mipsel-linux-gcc                ##
## Dependències:        Cap                                                ##
## Comentaris:          Directrius per compilar el programa recte.c        ##
#############################################################################

CC =/stuff/tmp/cross/bin/mipsel-linux-gcc
CFLAGS =-c -Wall

objects = recte.o

## Compila i envia per ethernet al robot
all_lan :   recte send_lan clean
## Compila i envia per Wifi al robot
all_wifi :  recte send_wifi clean
## Només compila
recte:      $(objects)
            $(CC) -o recte $(objects) -lpthread -lm

recte.o:    recte.c
            $(CC) -c recte.c

## Envia per ethernet
send_lan:
            scp ./recte root@192.168.0.103:./
## Envia per Wifi
send_wifi:
            scp ./recte root@192.168.1.103:./
## Neteja
clean:
            rm -f *.o recte
```

A.1.7. Programa “gir”

```
////////////////////////////////////////////////////////////////////
// Nom del programa:    gir.c                                     //
// Plataforma:          Nylon Linux del Wifibot                   //
// Paràmetre d'entrada: Angle a girar de -32768º a +32768º        //
// Autor:               Rafael Hesse                              //
// Data:                08/02/2008                                //
// Versió:              1.0                                       //
// Compilador:          GCC cross compiler mipsel-linux-gcc       //
// Dependències:        Harware i software original del robot     //
// Comentaris:          Fa girar el robot sobre el seu propi eix  //
//                      l'àngle marcat per la consigna            //
////////////////////////////////////////////////////////////////////

//INCLUDES
#include "wifibot2.h"
#include <tgmath.h>
#include <sys/time.h>

//DEFINICIO DE LES FUNCIONS:
static void initI2c();        // Inicialització del bus I2C
static void setI2cL();        // Llegir I2C del DSPIC esquerra
static void getI2cL();        // Escriure I2C en el DSPIC esquerra
static void setI2cR();        // Llegir I2C del DSPIC Dret
static void getI2cR();        // Escriure I2C en el DSPIC dret
static void stop();           // Parar el robot
void gestio_int (int numero); // Gestio de les interrupcions

//DEFINICIÓ DE LES VARIABLES GLOBALS
long w;                 // Posició angular en milèsomes de radiants
long w_D;               // V. angular roda dreta en milèsimes de rad/s
long w_E;               // V. angular roda esquerra en milèsimes de rad/s
int sentit_D;           // Sentit rodes dreta (1=endavant, -1=enrrera)
int sentit_E;           // Sentit rodes esquerra (1=endavant, -1=enrrera)
static unsigned char ordre[2]; // Consignes que enviem als PIC's
                               // [0]=Costat esq., [1]=Costat dret
static unsigned char bufset[2]; // Buffer d'escriptura per al bus I2C
                                // [0]=Costat esq., [1]=Costat dret
static unsigned char bufget[3]; // Buffer de lectura per al bus I2C
                                // [0]=[1]=Encoder, [2]=sensor dist.
static unsigned char bufad[1]; // Buffer de lectura per al bus I2C
int tics_E;             // Velocitat llegida tics/40ms esq.
int tics_D;             // Velocitat llegida tics/41ms dret

// MAIN: PROGRAMA PRINCIPAL //
int main(int argc, char *argv[])
{
// Definició de variables locals
struct itimerval val;         // Variable configuració del timer
struct sigaction interrupcio; // Variable configuració d'interrupcions
int girant;                   // Flag per indicar finalització del gir
int v1;                       // Consigna de velocitat per girar endavant
int v2;                       // Consigna de velocitat per girar enrera
int cons_w;                   // Consigna de posició 'W' rebuda en graus

// Inicialització de variables locals
girant=1;
```

```
v1=30;
v2=30+64;
cons_w = atoi(argv[1]);       // Agafem el paràmetre d'entrada 'consigna'
// Inicialització de variables globals
w_D=0;
w_E=0;
sentit_D=1;
sentit_E=1;
ordre[0]=ordre[1]=0;
bufset[0]=bufset[1]=0;
bufget[0]=bufget[1]=bufget[2]=0;
tics_E=0;
tics_D=0;
// Configurem el timer per que doni una alarma cada 50 ms
val.it_interval.tv_sec=0;
val.it_interval.tv_usec=50000;
val.it_value.tv_sec=0;
val.it_value.tv_usec=50000;
setitimer(ITIMER_REAL, &val, NULL);
cons_w=cons_w*17.4532;  // Passem la consigna a milèsimes de radiants
initI2c();              // Inicialització de les funcions I2C
// El manejador de de les senyals capturades es la funcio gestio_int
interrupcio . sa_handler = gestio_int;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (interrupcio . sa_mask));
// Tots els flags desactivats
interrupcio . sa_flags = 0;
// Capturem la senyal SIGQUIT causada per premer les tecles control
if (sigaction (SIGQUIT, & interrupcio, NULL) != 0) exit (1);
// El manejador de de les senyals capturades es la funcio gestio_int
interrupcio . sa_handler = gestio_int;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (interrupcio . sa_mask));
// Aquests flags permeten reiniciar el programa despres de l'interrupcio
interrupcio . sa_flags = SA_RESTART | SA_RESETHAND;
// Capturem la senyal SIGINT causada per premer Ctrl + c
if (sigaction (SIGINT, & interrupcio, NULL) != 0) exit (1);
// El manejador de de les senyals capturades es la funcio gestio_int
interrupcio . sa_handler = gestio_int;
// Especifiquem que no es bloquejarà cap senyal
sigemptyset (& (interrupcio . sa_mask));
// Tots els flags desactivats
interrupcio . sa_flags = 0;
// Capturem la senyal SIGALRM causada pel timer cada 50ms
if (sigaction (SIGALRM, & interrupcio, NULL) != 0) exit (1);

printf ("Es fará un gir de %d mm radiants sobre el propi eix\n",cons_w);
while (girant)
{
      if (cons_w>0 && w<cons_w) // Es demana un gir a dretes
      {
            ordre[0]=v1;
            ordre[1]=v2;
      }
      else if (cons_w<0 && w>cons_w) // Es demana gir a esquerra
      {
            ordre[0]=v2;
            ordre[1]=v1;
      }
      else
      {
```

```
            stop();
            girant=0;
      }
}
return 0;
}

// FUNCIONS DEL BUS I2C

// RUTINA PER INICIAR EL BUS I2C
void initI2c()
{
      sprintf(filename3,"/dev/i2c/%d",i2cbus);
      if ((file = open(filename3,O_RDWR)) < 0) printf("***Error al
            iniciar I2C\n");
}

// RUTINA PER ENVIAR DADES AL dsPIC DE LA ESQUERRA
void setI2cL()
{
      bufset[0]=ordre[0];
      bufset[1]=ordre[0];
      if (ioctl(file,I2C_SLAVE,0x51) < 0) printf("***Error al iniciar I2C
            esclau en DSPIC esquerra\n");
      if (write(file,bufset,2) !=2)
      {
            printf("***Error al escriure en I2C DSPIC esquerra \n");
      }
}

// RUTINA PER REBRE DADES DEL dsPIC DE LA ESQUERRA
void getI2cL()
{
      if (read(file,bufget,3) !=3)
      {
            printf("***Error al llegir I2C en DSPIC esquerra\n");
      }
      else {tics_E=bufget[0];}
}

// RUTINA PER ENVIAR DADES AL dsPIC DE LA DRETA
void setI2cR()
{
      bufset[0]=ordre[1];
      bufset[1]=ordre[1];
      if (ioctl(file,I2C_SLAVE,0x52) < 0) printf("***Error al iniciar I2C
            esclau en DSPIC dret\n");
      if (write(file,bufset,2) !=2)
      {
            printf("***Error al escriure en I2C DSPIC dret \n");
      }
}

// RUTINA PER REBRE DADES DEL dsPIC DE LA DRETA
void getI2cR()
{
      if (read(file,bufget,3) !=3)
      {
            printf("***Error al llegir I2C en DSPIC dret\n");
      }
      else {tics_D=bufget[0];}
```

```
}

// RUTINA PER PARAR EL ROBOT
void stop()
{
	ordre[0]=ordre[1]=0;
	bufset[0]=0;
	bufset[1]=0;
	ioctl(file,I2C_SLAVE,0x51);
	write(file,bufset,2);
	ioctl(file,I2C_SLAVE,0x52);
	write(file,bufset,2);
}

// RUTINA PER GESTIONAR LES INTERRUPCIONS
void gestio_int (int numero)
{
	switch (numero)
	{
	case SIGQUIT :
		stop();
		break;
	case SIGINT :
		stop();
		break;
	case SIGALRM :
		setI2cL(); // Enviem dades al motor esquerra
		getI2cL(); // Rebem dades del encoder esquerra
		setI2cR(); // Enviem dades al motor dret
		getI2cR(); // Rebem dades del encoder dret
		// Determinem el sentit de gir
		if (ordre[0] !=0 || ordre[1] !=0)
		{
			if(ordre[1]>=128)
			{
				if(ordre[1]>=192) sentit_D=1;
				else sentit_D=-1;
			}
			else
			{
				if(ordre[1]>=64) sentit_D=1;
				else sentit_D=-1;
			}
			if(ordre[0]>=128)
			{
				if(ordre[0]>=192) sentit_E=1;
				else sentit_E=-1;
			}
			else
			{
				if(ordre[0]>=64) sentit_E=1;
				else sentit_E=-1;
			}
		}
```

```
            // Filtrem dades errònies
            if((tics_D<<6)<5000 && (tics_E<<6)<5000)
            {
                  /*************************************
                  velocitat angular:
                        tics   1000ms     2*pi rad
                  w = ----- * ------- * -----------
                        41ms     1s       480 tics
                  *************************************/
                  // velocitat angular dreta en mili_rad/s
                  w_D=((tics_D<<6)*327.249*sentit_D)/50;
                  // velocitat angular esquerra en mili_rad/s
                  w_E=((tics_E<<6)*327.249*sentit_E)/50;

                  /*************************************
                  Posició angular:
                              w1 - w2
                  Wn = Wn-1 + --------- * t
                                B
                  on t = increment de temps (50ms)
                     B = distància entre rodes
                  *************************************/
                  // posició angular en mili_rad
                  w=w+((((w_D-w_E)*62)/340)*0.05);
            }
            else printf("+++++++++++++ error ++++++++++++\n");
            break;
      }
}
```

A.1.8. Makefile programa “gir”

```
#############################################################################
## Nom del programa:    Makefile                                           ##
## Plataforma:          Gentoo Linux                                       ##
## Autor:               Rafael Hesse                                       ##
## Data:                08/02/2008                                         ##
## Versió:              1.0                                                ##
## Compilador:          GCC cross compiler mipsel-linux-gcc                ##
## Dependències:        Cap                                                ##
## Comentaris:          Directrius per compilar el programa gir.c          ##
#############################################################################

CC =/stuff/tmp/cross/bin/mipsel-linux-gcc
CFLAGS =-c -Wall

objects = gir.o
## Compila i envia per ethernet al robot
all_lan :   gir send_lan clean
## Compila i envia per Wifi al robot
all_wifi :  gir send_wifi clean
## Només compila
gir:        $(objects)
            $(CC) -o gir $(objects) -lpthread -lm

gir.o:      gir.c
            $(CC) -c gir.c

## Envia per ethernet
send_lan:
            scp ./gir root@192.168.0.103:./
## Envia per Wifi
send_wifi:
            scp ./gir root@192.168.1.103:./
## Neteja
clean:
            rm -f *.o
```

A.2. Programació per al nou Wifibot

En aquest apartat es descriu el codi dels programes creats per a la nova arquitectura del robot. Es presenta el codi font per als programes dels dsPIC's i el programa per al Mesh Cube. El programa dissenyat per al control des d'un PC no es presenta en aquest annex donat que està fet amb un entorn gràfic (Lab View).

A.2.1. Programa dsPIC de darrera

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    Darrera.c                                    //
// AUTOR:         Rafael Hesse                                 //
// DATA:          01/04/2008                                   //
// PLATAFORMA:    dsPIC de darrera del Wifibot                 //
// LLENGUATGE:    C, compilador C18 Microchip                  //
// DESCRIPCIÓ:    Control de les rodes de darrera del Wifibot  //
////////////////////////////////////////////////////////////////

#include <bot.h>
#include <dsp.h>
#define I2C_SLAVE_ADDR 0x51

void I2CSetup (void); //Inicialització I2C
void composa_cub (void); //Composició de les dades rebudes
void descomposa_cub (void); //Descomposició de les dades rebudes
void descomposa_pic (void); //Descomposició de les dades rebudes
void composa_pic (void); //Composició de les dades rebudes
void set_PWM (void); //Aplicació del PWM

//DECLARACIO DE VARIABLES GLOBALS
tPID PID_D;       //Estructura per al PID de la dreta
tPID PID_E;       //Estructura per al PID de l'esquerra
fractional kCoeffs[3]; //Coeficients del PID
fractional abcCoefficient_D[3] __attribute__ ((section (".xbss, bss,
xmemory")));
fractional controlHistory_D[3] __attribute__ ((section (".ybss, bss,
ymemory")));
fractional abcCoefficient_E[3] __attribute__ ((section (".xbss, bss,
xmemory")));
fractional controlHistory_E[3] __attribute__ ((section (".ybss, bss,
ymemory")));
signed char signeD=1;   //Manté el signe per la velocitat
unsigned char Stopp=0;  //Flag de stop del dsPIC de davant
signed int Vel_D=0;     //Velocitat de la roda dreta (m/s)
signed int Vel_E=0;           //Velocitat de la roda esquerra (m/s)
unsigned char enc_valid=0;    //Flag per indicar que la lectura del
encoder es valida
unsigned char intercount=0;   //Comptador d'interrupsions
unsigned char ficom1=0;       //Flag final de comunicació
unsigned char ficom2=0;       //Flag final de comunicació
unsigned char byte;           //Dades enviades i rebudes per I2C
unsigned char countsend;      //Comptador de dades enviades
unsigned char countrcv;       //Comptador de dades rebudes
```

```
unsigned char slavestate;     //Control de comunicació esclau I2C
unsigned char buffsend[8];    //Buffer de dades enviades per I2C al cub
unsigned char buffrcv[4];     //Buffer de dades rebudes per I2C del cub
unsigned char buffsendP[3];   //Buffer de dades enviades per I2C al pic
de davant
unsigned char buffrcvP[3];    //Buffer de dades rebudes per I2C del pic
de davant
unsigned char ide=2;          //Identificacio de dades rebudes 0=Cub
1=PIC 2=No se
unsigned char Ctl_vel=0;      //Control de velocitat
unsigned char Ctl_pos=0;      //Control de posició
unsigned char Ctl_obs=0;      //Control d'obstacles
unsigned char Gir_ll=0;       //Gir lliure o duty 50%
unsigned char M_enc=0;        //Mode de lectura dels encoders
unsigned char Err_r=0;        //Reset dels errors
signed int Con_E=0;           //Consigna per la roda esquerra
signed int Con_D=0;           //Consigna per la roda dreta
unsigned char Err_des=0;      //Error de desbordament
unsigned char Err_cons=0;     //Error consigna fora de rang
unsigned char Err_com=0;      //Error de comunicació
unsigned char Cod_err=0;      //Codi de error
signed char Enc_E=0;          //Encoder esquerra (enviar al cub)
signed char Enc_D=0;          //Encoder dreta (enviar al cub)
unsigned char Enc_Ep=0;       //Encoder esquerra (enviar al dsPIC)
unsigned char Enc_Dp=0;       //Encoder dreta (enviar al dsPIC)
signed int Enc_D_ant=0;       //Lectura anterior de l'encoder de la dreta
signed int Enc_E_ant=0;       //Lectura anterior de l'encoder de
l'esquerra
signed int Enc_D_act=0;       //Lectura actual de l'encoder de la dreta
signed int Enc_E_act=0;       //Lectura actual de l'encoder de l'esquerra
unsigned char Sens_2=0;       //Sensor 2 (davant esquerra)
unsigned char Sens_4=0;       //Sensor 4 (lateral esquerra darrera)
unsigned char Sens_7=0;       //Sensor 7 (darrera dreta)
unsigned char Sens_6=0;       //Sensor 6 (lateral dret darrera)
unsigned char Obs_detF=0;     //Obstacle detectat pel dsPIC de davant
unsigned char Obs_detR=0;     //Obstacle detectat pel dsPIC de darrera
signed int Con_Ep=0;          //Consigna rebuda per al PWM esquerra del
dsPIC de davant
signed int Con_Dp=0;          //Consigna rebuda per al PWM dret del dsPIC
de davant

//RUTINA PER INICIALITZAR EL BUS 12C
void i2cSetup(unsigned char slaveAddress)
{
      unsigned int config1,config2;
      config2=66; // 100 kHz
      config1= I2C_ON & I2C_IDLE_CON & I2C_CLK_HLD &
      I2C_IPMI_DIS & I2C_7BIT_ADD & I2C_SLW_DIS &
      I2C_SM_DIS & I2C_GCALL_DIS & I2C_STR_EN &
      I2C_NACK & I2C_ACK_DIS & I2C_RCV_DIS &
      I2C_STOP_DIS & I2C_RESTART_DIS & I2C_START_DIS;
      I2CADD=slaveAddress;
      OpenI2C(config1,config2);
      ConfigIntI2C(SI2C_INT_ON & SI2C_INT_PRI_5 & MI2C_INT_OFF);
      IdleI2C();
}

//RUTINA PER CONFIGURAR EL CICLE DEL PWM
void set_PWM (void)
{
      signed int D;
```

```
signed int E;
if (Ctl_pos)
{
      D=Con_Dp;
      E=Con_Ep;
}
else
{
      D=Con_D;
      E=Con_E;
}
if (Ctl_vel)
{
      PID_D.controlReference = Q15(D/1000.0) ;
      PID_D.measuredOutput = Q15(Vel_D/1000.0) ;
      PID(&PID_D);
      D=(PID_D.controlOutput/32767.0)*1000.0;
      if (D>100) D=100;
      if (D<-100) D=-100;
      PID_E.controlReference = Q15(E/1000.0) ;
      PID_E.measuredOutput = Q15(Vel_E/1000.0) ;
      PID(&PID_E);
      E=(PID_E.controlOutput/32767.0)*1000.0;
      if (E>100) E=100;
      if (E<-100) E=-100;
}
if (D>0) signeD=1;
else if (D<0) signeD=-1;
if ((Obs_detF && Ctl_obs) || (Obs_detR && Ctl_obs) || Err_des ||
Err_cons || Err_com || Stopp)
{
      _POVD1L = 0;
      _POVD1H = 0;
      _POUT1L = 0;
      _POUT1H = 0;
      _POVD2L = 0;
      _POVD2H = 0;
      _POUT2L = 0;
      _POUT2H = 0;
}
else
{
//Motor esquerra
if (!Gir_ll)
{
      _POVD1L = 1;
      _POVD1H = 1;
      PWMCON1bits.PMOD1 = 0; //complementary mode
      if (E>=0) PDC1=300+(3*E);
      else if (E<0) PDC1=300+(3*E);
}
else if (Gir_ll)
{
      PWMCON1bits.PMOD1 = 1; //independent mode
      if (E>0)
      {
            _POVD1L = 0;
            _POVD1H = 1;
            _POUT1L = 0;
            PDC1=(6*E);
      }
```

```
            else if (E<0)
            {
                  _POVD1L = 1;
                  _POVD1H = 0;
                  _POUT1H = 0;
                  PDC1=6*(-E);
            }
            else if (E==0)
            {
                  _POVD1L = 0;
                  _POVD1H = 0;
                  _POUT1L = 0;
                  _POUT1H = 0;
            }
      }
      //Motor dret
      if (!Gir_ll)
      {
            _POVD2L = 1;
            _POVD2H = 1;
            PWMCON1bits.PMOD2 = 0; //complementary mode
            if (D>=0) PDC2=300+(3*D);
            else if (D<0) PDC2=300+(3*D);
      }
      else if (Gir_ll)
      {
            PWMCON1bits.PMOD2 = 1; //independent mode
            if (D>0)
            {
                  _POVD2L = 0;
                  _POVD2H = 1;
                  _POUT2L = 0;
                  PDC2=(6*D);
            }
            else if (D<0)
            {
                  _POVD2L = 1;
                  _POVD2H = 0;
                  _POUT2H = 0;
                  PDC2=6*(-D);
            }
            else if (D==0)
            {
                  _POVD2L = 0;
                  _POVD2H = 0;
                  _POUT2L = 0;
                  _POUT2H = 0;
            }
      }
      }
}

//RUTINA D'INTERRUPCIÓ BUS I2C ESCLAU
void __attribute__((__interrupt__, no_auto_psv)) _SI2CInterrupt(void)
{
IFS0bits.SI2CIF=0; //Baixem el flag de interrupció per adressa correcte
rebuda
if (_RBF)
{
      byte=SlaveReadI2C();
      if (!_D_A)
```

```
	{
		if (byte==0xA2)
		{
			ide=2;
			slavestate=0;
			countrcv=0;
		}
		if (byte==0xA3)
		{
			slavestate=2;
			countsend=0;
		}
	}
}

switch (slavestate)
{
	case 0:
	{
		slavestate=1;
		break;
	}

	case 1:
	{
		if (ide==2)
	{
	if ((byte&0x01)==0)
	{
		ide=0;
		composa_cub();
	}
	if ((byte&0x01)==1)
	{
		ide=1;
		composa_pic();
	}
	}
	if (ide==1)
	{
		buffrcvP[countrcv]=byte;
		countrcv++;
		if (countrcv==3) {slavestate=10;}
	}
	if (ide==0)
	{
		buffrcv[countrcv]=byte;
		countrcv++;
		if (countrcv==4) {slavestate=20;}
	}
	break;
	}

	case 2:
	{
		if (ide==0)
		{
			SlaveWriteI2C(buffsend[countsend]);
			countsend++;
			if (countsend==8)
			{
```

```
                        slavestate=30;
                        ide=2;
                        ficom1=1;
                        descomposa_cub();
                  }
            }
            if (ide==1)
            {
                  SlaveWriteI2C(buffsendP[countsend]);
                  countsend++;
                  if (countsend==3)
                  {
                        slavestate=40;
                        ficom2=1;
                        descomposa_pic();
                        ide=2;
                  }
            }
      break;
      }

      case 10:
      {
            //Si es genera interrupcio en aquest estat tenim un error
            Err_com=1;
            Cod_err=9;
            break;
      }

      case 20:
      {
            //Si es genera interrupcio en aquest estat tenim un error
            Err_com=1;
            Cod_err=7;
            break;
      }

      case 30:
      {
            //Interr. generada per l'últim ACK del master
            slavestate=31;
            break;
      }

      case 40:
      {
            //Interr. generada per l'últim ACK del master
            slavestate=41;
            break;
      }

      case 31:
      {
            //Si es genera interrupcio en aquest estat tenim un error
            Err_com=1;
            Cod_err=8;
            break;
      }
      case 41:
      {
            //Si es genera interrupcio en aquest estat tenim un error
```

```
            Err_com=1;
            Cod_err=10;
            break;
      }
} //end switch
I2CCONbits.SCLREL=1;
} //end slave iterrupt

// RUTINA PER INTERPRETAR LES DADES REBUDES DES DEL CUB
void descomposa_cub (void)
{
//Byte de configuració
Ctl_vel     = (buffrcv[0] & 0x02)>>1;
Ctl_pos     = (buffrcv[0] & 0x04)>>2;
Ctl_obs     = (buffrcv[0] & 0x08)>>3;
Err_r       = (buffrcv[0] & 0x10)>>4;
Gir_ll      = (buffrcv[0] & 0x20)>>5;
M_enc       = (buffrcv[0] & 0x40)>>6;

//Consignes de velocitat
Con_E       = buffrcv[1] & 0x7F;
if ((buffrcv[1] & 0x80)==0) Con_E=Con_E * -1;
Con_D       = buffrcv[2] & 0x7F;
if ((buffrcv[2] & 0x80)==0) Con_D=Con_D * -1;

//Ports auxiliars
TRISEbits.TRISE4 = (buffrcv[3] & 0x01);
TRISEbits.TRISE5 = (buffrcv[3] & 0x02)>>1;
TRISEbits.TRISE8 = (buffrcv[3] & 0x04)>>2;
if (TRISEbits.TRISE4==0) PORTEbits.RE4 = (buffrcv[3] & 0x08)>>3;
if (TRISEbits.TRISE5==0) PORTEbits.RE5 = (buffrcv[3] & 0x10)>>4;
if (TRISEbits.TRISE8==0) PORTEbits.RE8 = (buffrcv[3] & 0x20)>>5;

//Reset dels errors
if (Err_r) Err_des=Err_cons=Err_com=Cod_err=Stopp=0;

//Detecció de errors de consigna
if (Ctl_vel==0 && ((buffrcv[1]&0x7F)>100)) {Err_cons=1; Cod_err=15;}
if (Ctl_vel==0 && ((buffrcv[2]&0x7F)>100)) {Err_cons=1; Cod_err=15;}
if (Ctl_vel==1 && ((buffrcv[1]&0x7F)>128)) {Err_cons=1; Cod_err=15;}
if (Ctl_vel==1 && ((buffrcv[2]&0x7F)>128)) {Err_cons=1; Cod_err=15;}
}

// RUTINA PER INTERPRETAR LES DADES REBUDES DEL dsPIC de DAVANT
void descomposa_pic (void)
{
Obs_detF    = (buffrcvP[0] & 0x02)>>1;
Stopp       = (buffrcvP[0] & 0x10)>>4;
Con_Ep      = buffrcvP[1];
if ((buffrcvP[0] & 0x08)==0) Con_Ep=Con_Ep * -1; //Apliquem el signe de
la consigna
Con_Dp      = buffrcvP[2];
if ((buffrcvP[0] & 0x04)==0) Con_Dp=Con_Dp * -1; //Apliquem el signe de
la consigna
}

//RUTINA PER COMPOSAR LES DADES ENVIADES AL CUB
void composa_cub (void)
{
      buffsend[0] = (Err_des<<7) | (Err_cons<<6) | (Err_com<<5) |
      (Cod_err);
```

```
	if (Enc_E>=0) buffsend[1] = Enc_E | 0x80;
	else buffsend[1] = -Enc_E;
	if (Enc_D>=0) buffsend[2] = Enc_D;
	else buffsend[2] = -Enc_D;
	buffsend[3] = Sens_2;
	buffsend[4] = Sens_4;
	buffsend[5] = Sens_6;
	buffsend[6] = Sens_7;
	buffsend[7] = (PORTEbits.RE8<<5) | (PORTEbits.RE5<<4) |
	(PORTEbits.RE4<<3) |
	(TRISEbits.TRISE8<<2) | (TRISEbits.TRISE5<<1) | (TRISEbits.TRISE4);
}

// RUTINA PER COMPOSAR LES DADES ENVIADES AL dsPIC DE DAVANT
void composa_pic (void)
{
buffsendP[0] = Obs_detR<<1;
if ((Obs_detR && Ctl_obs) || Err_des || Err_cons || Err_com)
{
buffsendP[0] =buffsendP[0] | 0x04;
}
buffsendP[1] = Enc_Ep;
buffsendP[2] = Enc_Dp;
}
//RUTINA D'INTERRUPCIO PER AL CONVERSOR AD
void __attribute__((__interrupt__, auto_psv)) _ADCInterrupt(void)
{
	float volt; //Valor en volts llegit dels sensors
	float dist; //Valor en cm llegit en els sensors
	/*************************************************************
		    lectura       5
	voltatge = --------- * ------ (volts)
		       4         1023
	distància = (voltatge ^ -1.198) * 61.835 (centímetres)
	*************************************************************/
	_ADIF = 0;
	volt=((ADCBUF0+ADCBUF4+ADCBUF8+ADCBUFC)/4.0)*5.0/1023.0;
	dist=(pow(volt,-1.198))*61.835;
	if (dist<=255) Sens_7=(char)(dist);
	else Sens_7=255;
	volt=((ADCBUF1+ADCBUF5+ADCBUF9+ADCBUFD)/4.0)*5.0/1023.0;
	dist=(pow(volt,-1.198))*61.835;
	if (dist<=255) Sens_2=(char)(dist);
	else Sens_2=255;
	volt=((ADCBUF2+ADCBUF6+ADCBUFA+ADCBUFE)/4.0)*5.0/1023.0;
	dist=(pow(volt,-1.198))*61.835;
	if (dist<=255) Sens_4=(char)(dist);
	else Sens_4=255;
	volt=((ADCBUF3+ADCBUF7+ADCBUFB+ADCBUFF)/4.0)*5.0/1023.0;
	dist=(pow(volt,-1.198))*61.835;
	if (dist<=255) Sens_6=(char)(dist);
	else Sens_6=255;
	if ((Sens_2<= dist_obs) || (Sens_7<= dist_obs))
	{
		if (Ctl_obs)
		{
			Obs_detR=1;
			Cod_err=12;
		}
	}
	else Obs_detR=0;
```

```
}

//RUTINA D'INTERRUPCIÓ DEL TIMER 1 (CANAL A DE L'ENCODER)
void __attribute__((__interrupt__, auto_psv)) _T1Interrupt(void)
{
	_T1IF = 0;
	TMR1 = 0;
	Cod_err = 6; //Desbordament d'encoders
	T1CONbits.TON = 1; //Enable module
}

//RUTINA D'INTERRUPCIÓ DEL TIMER 2 (CANAL B DE L'ENCODER)
void __attribute__((__interrupt__, auto_psv)) _T2Interrupt(void)
{
	_T2IF = 0;
	TMR2 = 0;
	Cod_err = 6; //Desbordament d'encoders
	T2CONbits.TON = 1; //Enable module
}

//RUTINA D'INTERRUPCIÓ DEL TIMER 3 (CADA 5ms)
void __attribute__((__interrupt__, auto_psv)) _T3Interrupt(void)
{
	_T3IF = 0;
	intercount++;
	TMR3 = 0;
	ADCON1bits.SAMP = 0; //Start AD conversion
	if (intercount>=44)
	{
		Cod_err=13;
		Err_com=1;
	}
	Enc_D_ant=Enc_D_act;
	Enc_E_ant=Enc_E_act;
	if (POSCNT>=32767) Enc_E_act=-((POSCNT-32767)/4);
	else Enc_E_act=((32767-POSCNT)/4);
	Enc_D_act=(TMR1+TMR2)/2;
	if (enc_valid)
	{
		Vel_D=(Enc_D_act-Enc_D_ant)*VEL*signeD;
		Vel_E=(Enc_E_act-Enc_E_ant)*VEL;
	}
	if ((Vel_D>127)|(Vel_E>127)) Cod_err=14;
	if (Ctl_vel && enc_valid) set_PWM();
	else enc_valid=1;
	if (M_enc)
	{
		Enc_D=Vel_D;
		Enc_E=Vel_E;
	}
}

//RUTINA PRINCIPAL
int main (void)
{
	PID_D.abcCoefficients = &abcCoefficient_D[0]; //Punter als
	coeficients PID derivats
	PID_D.controlHistory = &controlHistory_D[0]; //Punter al historial
	de control
	PID_E.abcCoefficients = &abcCoefficient_E[0]; //Punter als
	coeficients PID derivats
```

```
	PID_E.controlHistory = &controlHistory_E[0]; //Punter al historial
	de control
	PIDInit(&PID_D);		//Inicialització del PID
	PIDInit(&PID_E);		//Inicialització del PID
	kCoeffs[0] = Q15(kp);	//Escalat dels coeficients
	kCoeffs[1] = Q15(ki);	//Escalat dels coeficients
	kCoeffs[2] = Q15(kd);	//Escalat dels coeficients
	PIDCoeffCalc(&kCoeffs[0], &PID_D); //Càlcul dels coeficients
	derivats
	PIDCoeffCalc(&kCoeffs[0], &PID_E); //Càlcul dels coeficients
	derivats
	Delay_ms(5000);
	i2cSetup(I2C_SLAVE_ADDR);	//Inici I2C
	init_PWM();			//Inici PWM
	init_T_QEI();			//Inici QUEI i timers
	init_AD();			//Inici conversor A/D
	slavestate=0;

	while(1) //Bucle infinit
	{
		if (ficom1)
		{
			intercount=0;
			Enc_Dp = (TMR1+TMR2)/20;
			TMR2 = 0;
			TMR1 = 0;
			if (POSCNT>=32767) Enc_Ep=-((POSCNT-32767)/40);
			else Enc_Ep=((32767-POSCNT)/40);
			POSCNT = 32767;
			enc_valid=0;
			if ((Enc_Ep>127) | (Enc_Dp>127)) Cod_err=14;
			if (!M_enc)
			{
				Enc_E=Enc_Ep;
				Enc_D=Enc_Dp;
			}
			ficom1=0;
		}
		if (ficom2)
		{
			if (Ctl_pos)
			{
				Con_E=Con_Ep;
				Con_D=Con_Dp;
			}
			if (!Ctl_vel) set_PWM();
			ficom2=0;
		}
	}
	return 0;
} //end main
```

A.2.2. Programa dsPIC de davant

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    Davant.c                                    //
// AUTOR:         Rafael Hesse                                //
// DATA:          01/04/2008                                  //
// PLATAFORMA:    dsPIC davanter del Wifibot                  //
// LLENGUATGE:    C, compilador C18 Microchip                 //
// DESCRIPCIÓ:    Control de les rodes davanteres del Wifibot //
////////////////////////////////////////////////////////////////

#include <bot.h>
#include <dsp.h>
#define I2C_SLAVE_ADDR 0x52 //Adreça I2C

void I2CSetup (void);         //Inicialització I2C
void descomposa_cub (void);   //Descomposició de les dades rebudes
void descomposa_pic (void);   //Descomposició de les dades rebudes
void composa_pic (void);      //Composició de les dades rebudes
void composa_cub (void);      //Composició de les dades rebudes
void set_PWM (void);          //Aplicació del PWM
void control_angle (void);    //Control de l'angle final

//DECLARACIÓ DE VARIABLES GLOBALS
tPID PID_D; //Estructura per al PID de la dreta
tPID PID_E; //Estructura per al PID de l'esquerra
fractional kCoeffs[3]; //Coeficients del PID
fractional abcCoefficient_D[3] __attribute__ ((section (".xbss, bss,
xmemory")));
fractional controlHistory_D[3] __attribute__ ((section (".ybss, bss,
ymemory")));
fractional abcCoefficient_E[3] __attribute__ ((section (".xbss, bss,
xmemory")));
fractional controlHistory_E[3] __attribute__ ((section (".ybss, bss,
ymemory")));
unsigned char countsend;      //Comptador de bytes enviats
unsigned char countrcv;       //Comptador de bytes rebuts
unsigned char byteS;          //Byte esclau
unsigned char countsendS;     //Contador de dades enviades com esclau
unsigned char countrcvS;      //Contador de dades rebudes com esclau
unsigned char Con_V=0;        //Consigna de velocitat (Ctl_pos i Ctl_vel)
unsigned char ficom=0;        //Flag final de comunicacions
unsigned char intercount=0;   //Comptador d'interrupsions
unsigned char Stop=0;         //Flag de stop
unsigned char Stopp=0;        //Flag de stop del pic de darrera
signed long X=0;              //Posició X (tics)
signed long Y=0;              //Posició Y (tics)
signed int iX=0;              //Increment de posició X (tics)
signed int iY=0;              //Increment de posició Y (tics)
signed long Tet=0;            //Posició angular (radiants)
signed int iPos=0;            //Increment de posició del robot (tics)
signed int iN_D=0;            //Increment de tics de la dreta
signed int iN_E=0;            //Increment de tics de l'esquerra
signed int Vel_D=0;           //Velocitat de la roda dreta (m/s)
signed int Vel_E=0;           //Velocitat de la roda esquerra (m/s)
signed char signeE=1;         //Manté el signe per la velocitat
unsigned char enc_valid=0;    //Flag per indicar que la lectura del
encoder es valida
```

```
unsigned char arribat=0;      //Flag per al control de posicio
unsigned char slavestate;     //Control de comunicació esclau I2C
unsigned char masterstate;    //Control de comunicació mestre I2C
unsigned char buffsend[13];   //Buffer de dades enviades per I2C al cub
unsigned char buffrcv[14];    //Buffer de dades rebudes per I2C del cub
unsigned char buffsendP[3];   //Buffer de dades enviades per I2C al pic
de darrera
unsigned char buffrcvP[3];    //Buffer de dades rebudes per I2C del pic
de darrera
unsigned char Ctl_vel=0;      //Control de velocitat
unsigned char Ctl_pos=0;      //Control de posició
unsigned char Ctl_obs=0;      //Control d'obstacles
unsigned char Gir_ll=0;       //Mode gir lliure o duty 50%
unsigned char Ctl_Tf=0;       //Control de l'angle final
unsigned char M_enc=0;        //Mode de lectura dels encoders
unsigned char Corr_pos=0;     //Correcció de la posició interna
unsigned char Err_r=0;        //Reset dels errors
signed int Con_E=0;           //Consigna per la roda esquerra
signed int Con_D=0;           //Consigna per la roda dreta
signed int Pos_x=0;           //Pos del robot eix x (-25.500-25.500cm)
signed int Pos_y=0;           //Pos del robot eix y (-25.500-25.500cm)
unsigned int Pos_T=0;         //Posició del robot angular (0 - 360º)
signed int Con_x=0;           //Consigna de posició en l'eix x (-25.500 -
25.500 cm)
signed int Con_y=0;           //Consigna de posició en l'eix y (-25.500 -
25.500 cm)
unsigned int Con_T=0;         //Consigna de posició angular (0 - 360º)
unsigned char Err_des=0;      //Error de desbordament
unsigned char Err_cons=0;     //Error consigna fora de rang
unsigned char Err_com=0;      //Error de comunicació
unsigned char Cod_err=0;      //Codi de error
signed char Enc_E=0;          //Encoder esquerra
signed char Enc_D=0;          //Encoder dreta
signed int Enc_D_ant=0;       //Lectura anterior de l'encoder de la dreta
signed int Enc_E_ant=0;       //Lectura anterior de l'encoder de
l'esquerra
signed int Enc_D_act=0;       //Lectura actual de l'encoder de la dreta
signed int Enc_E_act=0;       //Lectura actual de l'encoder de l'esquerra
unsigned char Sens_1=0;       //Sensor 1 (davant dreta)
unsigned char Sens_3=0;       //Sensor 3 (lateral esquerra davant)
unsigned char Sens_8=0;       //Sensor 8 (darrera esquerra)
unsigned char Sens_5=0;       //Sensor 5 (lateral dret davant)
unsigned char Obs_detF=0;     //Obstacle detectat pel dsPIC de davant
unsigned char Obs_detR=0;     //Obstacle detectat pel dsPIC de darrera
signed int Con_Ep=0;          //Consigna enviada al PWM esquerra del
dsPIC de darrera
signed int Con_Dp=0;          //Consigna enviada al PWM esquerra del
dsPIC de darrera
unsigned char Enc_Dp=0;       //Valor del encoder dret del dsPIC de
darrera
unsigned char Enc_Ep=0;       //Valor del encoder esquerra del dsPIC de
darrera

//RUTINA PER INICIALITZAR EL BUS 12C
void i2cSetup(unsigned char slaveAddress)
{
      unsigned int config1,config2;
      config2=66; // 100 kHz
      config1= I2C_ON & I2C_IDLE_CON & I2C_CLK_HLD &
      I2C_IPMI_DIS & I2C_7BIT_ADD & I2C_SLW_DIS &
      I2C_SM_DIS & I2C_GCALL_DIS & I2C_STR_EN &
```

```
	I2C_NACK & I2C_ACK_DIS & I2C_RCV_DIS &
	I2C_STOP_DIS & I2C_RESTART_DIS & I2C_START_DIS;
	I2CADD=slaveAddress;
	OpenI2C(config1,config2);
	ConfigIntI2C(SI2C_INT_ON & SI2C_INT_PRI_5 & MI2C_INT_ON &
	MI2C_INT_PRI_5);
	IdleI2C();
}

//RUTINA PER CONFIGURAR EL CICLE DEL PWM
void set_PWM (void)
{
	signed int D;
	signed int E;
	D=Con_D;
	E=Con_E;
	if (Ctl_vel)
	{
		PID_D.controlReference = Q15(D/1000.0) ;
		PID_D.measuredOutput = Q15(Vel_D/1000.0) ;
		PID(&PID_D);
		D=(PID_D.controlOutput/32767.0)*1000.0;
		if (D>100) D=100;
		if (D<-100) D=-100;
		PID_E.controlReference = Q15(E/1000.0) ;
		PID_E.measuredOutput = Q15(Vel_E/1000.0) ;
		PID(&PID_E);
		E=(PID_E.controlOutput/32767.0)*1000.0;
		if (E>100) E=100;
		if (E<-100) E=-100;
		Con_Ep=E;
		Con_Dp=D;
	}

	if (E>0) signeE=1;
	else if (E<0) signeE=-1;
	if ((Obs_detF && Ctl_obs) || (Obs_detR && Ctl_obs) || Err_des ||
	Err_cons || Err_com || Stop || Stopp)
	{
		_POVD1L = 0;
		_POVD1H = 0;
		_POUT1L = 0;
		_POUT1H = 0;
		_POVD2L = 0;
		_POVD2H = 0;
		_POUT2L = 0;
		_POUT2H = 0;
	}
	else
	{
		//Motor esquerra
		if (!Gir_ll)
		{
		_POVD1L = 1;
		_POVD1H = 1;
		PWMCON1bits.PMOD1 = 0; //complementary mode
		if (E>=0) PDC1=300+(3*E);
		else if (E<0) PDC1=300+(3*E);
	}
	else if (Gir_ll)
	{
```

```
            PWMCON1bits.PMOD1 = 1; //independent mode
            if (E>0)
            {
                  _POVD1L = 0;
                  _POVD1H = 1;
                  _POUT1L = 0;
                  PDC1=(6*E);
            }
            else if (E<0)
            {
                  _POVD1L = 1;
                  _POVD1H = 0;
                  _POUT1H = 0;
                  PDC1=6*(-E);
            }
            else if (E==0)
            {
                  _POVD1L = 0;
                  _POVD1H = 0;
                  _POUT1L = 0;
                  _POUT1H = 0;
            }
      }
      //Motor dret
      if (!Gir_ll)
      {
            _POVD2L = 1;
            _POVD2H = 1;
            PWMCON1bits.PMOD2 = 0; //complementary mode
            if (D>=0) PDC2=300+(3*D);
            else if (D<0) PDC2=300+(3*D);
      }
      else if (Gir_ll)
      {
            PWMCON1bits.PMOD2 = 1; //independent mode
            if (D>0)
            {
                  _POVD2L = 0;
                  _POVD2H = 1;
                  _POUT2L = 0;
                  PDC2=(6*D);
            }
            else if (D<0)
            {
                  _POVD2L = 1;
                  _POVD2H = 0;
                  _POUT2H = 0;
                  PDC2=6*(-D);
            }
            else if (D==0)
            {
                  _POVD2L = 0;
                  _POVD2H = 0;
                  _POUT2L = 0;
                  _POUT2H = 0;
            }
      }
      }
}
```

```
//RUTINA D'INTERRUPCIÓ BUS I2C ESCLAU
void __attribute__((__interrupt__, no_auto_psv)) _SI2CInterrupt(void)
{
	IFS0bits.SI2CIF=0; //Baixem el flag de interrupció per adressa
	correcte rebuda
	if (_RBF)
	{
		byteS=SlaveReadI2C();
		if (!_D_A)
		{
			if (byteS==0xA4)
			{
				slavestate=0;
				countrcvS=0;
			}
			if (byteS==0xA5)
			{
				slavestate=2;
				countsendS=0;
			}
		}
	}
	switch (slavestate)
	{
		case 0:
		{
			slavestate=1;
			composa_cub();
			break;
		}
		case 1:
		{
			buffrcv[countrcvS]=byteS;
			countrcvS++;
			if (countrcvS==14)
			{
				slavestate=10;
			}
		break;
		}
		case 2:
		{
			SlaveWriteI2C(buffsend[countsendS]);
			countsendS++;
			if (countsendS==13)
			{
				intercount=0;
				slavestate=20;
			}
		break;
		}
		case 10:
		{
			//Si es genera interrupcio en aquest estat tenim un
			error
			Err_com=1;
			Cod_err=7;
			break;
		}
		case 20:
		{
```

```
                    //Interr. generada per l'últim ACK del master
                    composa_pic();
                    descomposa_cub();
                    Delay_ms(1);
                    StartI2C();
                    slavestate=21;
                    break;
              }
              case 21:
              {
                    //Si es genera interrupcio en aquest estat tenim un
                    error
                    Err_com=1;
                    Cod_err=8;
                    break;
              }
} //end switch

I2CCONbits.SCLREL=1;
} //end slave iterrupt

//RUTINA D'INTERRUPCIÓ BUS I2C MESTRE
void __attribute__((__interrupt__, no_auto_psv)) _MI2CInterrupt(void)
{
IFS0bits.MI2CIF=0;
switch (masterstate)
{
      case 0:
      {
            MasterWriteI2C(0xA2);
            countsend=0;
            masterstate=1;
            break;
      }
      case 1:
      {
            MasterWriteI2C(buffsendP[countsend]);
            countsend++;
            if (countsend==3) masterstate=2;
            break;
      }
      case 2:
      {
            StopI2C();
            masterstate=3;
            break;
      }
      case 3:
      {
            StartI2C();
            masterstate=20;
            break;
      }
      case 20:
      {
            MasterWriteI2C(0xA3);
            countrcv=0;
            masterstate=21;
            break;
      }
```

```
	case 21:
	{
		buffrcvP[countrcv]=MasterReadI2C();
		countrcv++;
		if (countrcv==2) masterstate=23;
		else masterstate=22;
		break;
	}
	case 22:
	{
		AckI2C();
		masterstate=21;
		break;
	}
	case 23:
	{
		NotAckI2C();
		masterstate=24;
		break;
	}
	case 24:
	{
		StopI2C();
		masterstate=25;
		break;
	}
	case 25:
	{
		masterstate=0;
		descomposa_pic();
		ficom=1;
		break;
	}
} //end switch
} //end master iterrupt

//RUTINA D'INTERRUPCIO PER AL CONVERSOR AD
void __attribute__((__interrupt__, auto_psv)) _ADCInterrupt(void)
{
float volt; //Valor en volts llegit dels sensors
float dist; //Valor en cm llegit en els sensors

/*********************************************************
            lectura       5
voltatge = --------- * ------ (volts)
               4        1023
distància = (voltatge ^ -1.198) * 61.835 (centímetres)
*********************************************************/

_ADIF = 0;

volt=((ADCBUF0+ADCBUF4+ADCBUF8+ADCBUFC)/4.0)*5.0/1023.0;
dist=(pow(volt,-1.198))*61.835;
if (dist<=255) Sens_8=(unsigned char)(dist);
else Sens_8=255;

volt=((ADCBUF1+ADCBUF5+ADCBUF9+ADCBUFD)/4.0)*5.0/1023.0;
dist=(pow(volt,-1.198))*61.835;
if (dist<=255) Sens_1=(unsigned char)(dist);
else Sens_1=255;
```

```
volt=((ADCBUF2+ADCBUF6+ADCBUFA+ADCBUFE)/4.0)*5.0/1023.0;
dist=(pow(volt,-1.198))*61.835;
if (dist<=255) Sens_3=(unsigned char)(dist);
else Sens_3=255;

volt=((ADCBUF3+ADCBUF7+ADCBUFB+ADCBUFF)/4.0)*5.0/1023.0;
dist=(pow(volt,-1.198))*61.835;
if (dist<=255) Sens_5=(unsigned char)(dist);
else Sens_5=255;

if ((Sens_1<= dist_obs) || (Sens_8<= dist_obs))
{
      if (Ctl_obs)
      {
            Obs_detF=1;
            Cod_err=11;
      }
}
else Obs_detF=0;
}

// RUTINA PER INTERPRETAR LES DADES REBUDES DES DEL CUB
void descomposa_cub (void)
{
//Byte de configuració
M_enc       = (buffrcv[0] & 0x01);
Ctl_vel     = (buffrcv[0] & 0x02)>>1;
Ctl_pos     = (buffrcv[0] & 0x04)>>2;
Ctl_obs     = (buffrcv[0] & 0x08)>>3;
Err_r       = (buffrcv[0] & 0x10)>>4;
Gir_ll      = (buffrcv[0] & 0x20)>>5;
Ctl_Tf      = (buffrcv[0] & 0x40)>>6;
Corr_pos    = (buffrcv[0] & 0x80)>>7;
//Consignes de velocitat
if (!Ctl_pos)
{
      Con_E = buffrcv[1] & 0x7F;
      if ((buffrcv[1] & 0x80)==0) Con_E=Con_E * -1;
      Con_D = buffrcv[2] & 0x7F;
      if ((buffrcv[2] & 0x80)==0) Con_D=Con_D * -1;
}
else Con_V = buffrcv[1] & 0x7F;

//consignes de posicio
Con_x = (buffrcv[3] & 0x7F) + (buffrcv[5]*100);
if ((buffrcv[3] & 0x80)==0) Con_x=Con_x * -1;
Con_y = (buffrcv[4] & 0x7F) + (buffrcv[6]*100);
if ((buffrcv[4] & 0x80)==0) Con_y=Con_y * -1;
Con_T = ((buffrcv[7] * 360.0)/255.0)+0.5;
if (Con_x>25599) {Err_cons=1; Cod_err=3;}
if (Con_y>25599) {Err_cons=1; Cod_err=4;}
if ((Con_x+Pos_x)>25599) {Err_des=1; Cod_err=1;}
if ((Con_y+Pos_y)>25599) {Err_des=1; Cod_err=2;}

//correcció de posició
if (Corr_pos)
{
      Pos_x = (buffrcv[8] & 0x7F) + (buffrcv[10]*100);
      if ((buffrcv[8] & 0x80)==0) Pos_x=Pos_x * -1;
      if (Pos_x>=0) X=(((signed long)Pos_x*(signed long)10000)/K)+0.5;
      else X=(((signed long)Pos_x*(signed long)10000)/K)-0.5;
```

```
      Pos_y = (buffrcv[9] & 0x7F) + (buffrcv[11]*100);
      if ((buffrcv[9] & 0x80)==0) Pos_y=Pos_y * -1;
      if (Pos_y>=0) Y=(((signed long)Pos_y*(signed long)10000)/K)+0.5;
      else Y=(((signed long)Pos_y*(signed long)10000)/K)-0.5;
      Pos_T = (unsigned int)(((buffrcv[12] * 360.0)/255.0)+0.5);
      Tet = (Pos_T*rad)*1000.0;
      iX = 0;
      iY = 0;
}

//Ports auxiliars
TRISEbits.TRISE4 = (buffrcv[13] & 0x01);
TRISEbits.TRISE5 = (buffrcv[13] & 0x02)>>1;
TRISEbits.TRISE8 = (buffrcv[13] & 0x04)>>2;
if (TRISEbits.TRISE4==0) PORTEbits.RE4 = (buffrcv[13] & 0x08)>>3;
if (TRISEbits.TRISE5==0) PORTEbits.RE5 = (buffrcv[13] & 0x10)>>4;
if (TRISEbits.TRISE8==0) PORTEbits.RE8 = (buffrcv[13] & 0x20)>>5;
//Reset dels errors
if (Err_r) Err_des=Err_cons=Err_com=Cod_err=Stopp=0;
//Detecció de errors de consigna
if (Ctl_vel==0 && ((buffrcv[1]&0x7F)>100)) {Err_cons=1; Cod_err=15;}
if (Ctl_vel==0 && ((buffrcv[2]&0x7F)>100)) {Err_cons=1; Cod_err=15;}
if (Ctl_vel==1 && ((buffrcv[1]&0x7F)>128)) {Err_cons=1; Cod_err=15;}
if (Ctl_vel==1 && ((buffrcv[2]&0x7F)>128)) {Err_cons=1; Cod_err=15;}
}

//RUTINA PER INTERPRETAR LES DADES REBUDES DEL dsPIC DE DARRERA
void descomposa_pic (void)
{
Obs_detR    = (buffrcvP[0] & 0x02)>>1;
Stopp       = (buffrcvP[0] & 0x04)>>2;
Enc_Ep      = buffrcvP[1];
Enc_Dp      = buffrcvP[2];
}

//RUTINA PER COMPOSAR LES DADES ENVIADES AL CUB
void composa_cub (void)
{
buffsend[0] = (Err_des<<7) | (Err_cons<<6) | (Err_com<<5) | (Cod_err);
if (Enc_E>0) buffsend[1] = Enc_E;
else buffsend[1] = -Enc_E;
if (Enc_D>=0) buffsend[2] = Enc_D | 0x80;
else buffsend[2] = -Enc_D;
buffsend[3] = Sens_1;
buffsend[4] = Sens_3;
buffsend[5] = Sens_5;
buffsend[6] = Sens_8;
if (Pos_x>=0)
{
      buffsend[7] = (unsigned char)(Pos_x % 100); //Extraiem la part de
      centímetres
      buffsend[9] = (unsigned char)(Pos_x / 100); //Extraiem la part de
      metres
      buffsend[7] = buffsend[7] | 0x80; //Afegim el bit de signe
}
else
{
      buffsend[7] = (unsigned char)((Pos_x*-1) % 100); //Extraiem la part
      de
      centímetres
```

```
      buffsend[9] = (unsigned char)((Pos_x*-1) / 100); //Extraiem la part
      de metres
}
if (Pos_y>=0)
{
      buffsend[8] = (unsigned char)(Pos_y % 100); //Extraiem la part de
      centímetres
      buffsend[10]= (unsigned char)(Pos_y / 100); //Extraiem la part de
      metres
      buffsend[8] = buffsend[8] | 0x80; //Afegim el bit de signe
}
else
{
      buffsend[8] = (unsigned char)((Pos_y*-1) % 100); //Extraiem la part
      de
      centímetres
      buffsend[10] = (unsigned char)((Pos_y*-1) / 100); //Extraiem la
      part de metres
}
buffsend[11]= ((Pos_T * 255.0) / 360.0)+0.5; //Valor escalat (0-360º ~ 0-
255)
//Sumem 0.5 per corregir l'arrodoniment
buffsend[12] = (PORTEbits.RE8<<5) | (PORTEbits.RE5<<4) |
(PORTEbits.RE4<<3) |
(TRISEbits.TRISE8<<2) | (TRISEbits.TRISE5<<1) | (TRISEbits.TRISE4);
}

//RUTINA PER COMPOSAR LES DADES ENVIADES AL dsPIC DE DARRERA
void composa_pic (void)
{
buffsendP[0] = (1) | (Obs_detF<<1);
if ((Obs_detF && Ctl_obs) || Err_des || Err_cons || Err_com || Stop)
{
      buffsendP[0] =buffsendP[0] | 0x10;
}
if (Con_Ep>=0)
{
      buffsendP[0] = buffsendP[0] | 0x08;
      buffsendP[1] = (unsigned char) (Con_Ep);
}
else buffsendP[1] = (unsigned char) (-Con_Ep);
if (Con_Dp>=0)
{
      buffsendP[0] = buffsendP[0] | 0x04;
      buffsendP[2] = (unsigned char) (Con_Dp);
}
else buffsendP[2] = (unsigned char) (-Con_Dp);
}

//RUTINA D'INTERRUPCIÓ DEL TIMER 1 (CANAL A DE L'ENCODER)
void __attribute__((__interrupt__, auto_psv)) _T1Interrupt(void)
{
      _T1IF = 0;
      TMR1 = 0;
      Cod_err = 6; //Desbordament d'encoders
      T1CONbits.TON = 1; //Enable module
}

//RUTINA D'INTERRUPCIÓ DEL TIMER 2 (CANAL B DE L'ENCODER)
void __attribute__((__interrupt__, auto_psv)) _T2Interrupt(void)
{
```

```
	_T2IF = 0;
	TMR2 = 0;
	Cod_err = 6; //Desbordament d'encoders
	T2CONbits.TON = 1; //Enable module
}

//RUTINA D'INTERRUPCIÓ DEL TIMER 3 (CADA 5ms)
void __attribute__((__interrupt__, auto_psv)) _T3Interrupt(void)
{
	_T3IF = 0;
	intercount ++;
	if (intercount>=44)
	{
		Cod_err=13;
		Err_com=1;
	}
	TMR3 = 0;
	ADCON1bits.SAMP = 0; //Start AD conversion
	Enc_D_ant=Enc_D_act;
	Enc_E_ant=Enc_E_act;
	if (POSCNT>=32767) Enc_D_act=(POSCNT-32767)/4;
	else Enc_D_act=-((32767-POSCNT)/4);
	Enc_E_act=(TMR1+TMR2)/2;
	if (enc_valid)
	{
		Vel_D=(Enc_D_act-Enc_D_ant)*VEL;
		Vel_E=(Enc_E_act-Enc_E_ant)*VEL*signeE;
	}
	if (M_enc)
	{
		Enc_D=Vel_D;
		Enc_E=Vel_E;
	}
if (Ctl_vel && enc_valid) set_PWM();
enc_valid=1;
}

//RUTINA PER REALITZAR ELS CÀLCULS D'ODOMETRIA
void Odometria (void)
{
	float c;
	float s;
	iN_E = (TMR1+TMR2)/2;
	TMR2 = 0;
	TMR1 = 0;
	if (Con_E<0) iN_E=-iN_E;
	if (POSCNT>=32767) iN_D=(POSCNT-32767)/4;
	else iN_D=-((32767-POSCNT)/4);
	POSCNT = 32767;
	enc_valid=0;
	iPos = ((iN_D+iN_E)/2);
	Tet = Tet+(((iN_D-iN_E)*B)/1000);
	while (Tet>=6283) Tet=Tet-6283;
	while (Tet<0) Tet=Tet+6283;
	c=cos(Tet/1000.0);
	s=sin(Tet/1000.0);
	iX = iPos * c;
	iY = iPos * s;
	X = X+iX;
	Y = Y+iY;
	if (X>=0) Pos_x = ((X*K)/10000)+0.5;
```

```
	else Pos_x = ((X*K)/10000)-0.5;
	if (Y>=0) Pos_y = ((Y*K)/10000)+0.5;
	else Pos_y = ((Y*K)/10000)-0.5;
	Pos_T = (Tet*graus)/1000.0;
	if (!M_enc)
	{
		if ((iN_E>1270)||(iN_D>1270)) Cod_err=14;
		else
		{
		Enc_E=iN_E/10;
		Enc_D=iN_D/10;
		}
	}
}
//RUTINA PER CALCULAR LA TRAJECTORIA
void control_trajectoria (void)
{
	signed long Err_x;
	signed long Err_y;
	signed long D;
	float Err_T;
	float T_D;
	float fact;
	Err_x=Con_x-Pos_x;
	Err_y=Con_y-Pos_y;
	D=sqrt((signed long)(Err_x*Err_x)+(signed long)(Err_y*Err_y));
	if (D<=10) arribat=1;
	else arribat=0;
	if (!arribat)
	{
		T_D=atan2((Err_y*1.0),(Err_x*1.0));
		while (T_D>(2*PI)) T_D=T_D-(2*PI);
		while (T_D<0) T_D=T_D+(2*PI);
		Err_T=T_D-(Tet/1000.0);
		while (Err_T>(2*PI)) Err_T=Err_T-(2*PI);
		while (Err_T<0) Err_T=Err_T+(2*PI);
		if (Err_T<=0.79) //Error 0~45º 0~0.79 rad
		{
			fact=pow(1-((Err_T)/PI),5);
			Con_D=Con_V;
			Con_E=Con_V*fact;
		}
		else if ((Err_T>0.79)&(Err_T<=PI)) //Error 45~180º 0.79~pi
		rad
		{
			Con_D=Con_V;
			Con_E=-Con_V;
		}
		else if ((Err_T>PI)&(Err_T<5.5)) //Error 180-315º pi~5.5 rad
		{
			Con_D=-Con_V;
			Con_E=Con_V;
		}
		else if (Err_T>=5.5) //Error 315-360º 5.5~2pi rad
		{
			fact=pow(1-((Err_T-PI)/PI),5);
			Con_D=Con_V*fact;
			Con_E=Con_V;
		}
		else
		{
```

```
                Con_D=Con_V;
                Con_E=Con_V;
            }
            Con_Ep=Con_E;
            Con_Dp=Con_D;
      }
      else control_angle();
}

//RUTINA PER CONTROLAR L'ANGLE FINAL DEL ROBOT
void control_angle (void)
{
      if (Ctl_Tf)
      {
            signed int Err;
            Err = Pos_T-Con_T;
            if ( ((Err>0)&(Err<180)) || ((Err>-360)&(Err<-180)) )
            {
                Con_E=Con_V;
                Con_D=-Con_V;
            }
            else
            {
                Con_E=-Con_V;
                Con_D=Con_V;
            }
            if ((Err<5) && (Err>-5))
            {
                Stop=1;
            }
            Con_Ep=Con_E;
            Con_Dp=Con_D;
      }
      else
      {
            Stop=1;
      }
}

//RUTINA PRINCIPAL
int main (void)
{
      PID_D.abcCoefficients = &abcCoefficient_D[0]; //Punter als
      coeficients PID derivats
      PID_D.controlHistory = &controlHistory_D[0]; //Punter al historial
      de control
      PID_E.abcCoefficients = &abcCoefficient_E[0]; //Punter als
      coeficients PID derivats
      PID_E.controlHistory = &controlHistory_E[0]; //Punter al historial
      de control
      PIDInit(&PID_D);        //Inicialització del PID
      PIDInit(&PID_E);        //Inicialització del PID
      kCoeffs[0] = Q15(kp);   //Escalat dels coeficients
      kCoeffs[1] = Q15(ki);   //Escalat dels coeficients
      kCoeffs[2] = Q15(kd);   //Escalat dels coeficients
      PIDCoeffCalc(&kCoeffs[0], &PID_D); //Càlcul dels coeficients
      derivats
      PIDCoeffCalc(&kCoeffs[0], &PID_E); //Càlcul dels coeficients
      derivats
      Delay_ms(5000);
      i2cSetup(I2C_SLAVE_ADDR);     //Inici I2C
```

```
      init_PWM();                   //Inici PWM
      init_AD();                    //Inici conversor A/D
      init_T_QEI();                 //Inici QUEI i timers
      masterstate=0;

      while(1) //Bucle infinit
      {
            if (ficom)
            {
                  Odometria();
                  if (Ctl_pos) control_trajectoria();
                  else Stop=0;
                  if (Stop) Con_Ep=Con_Dp=0;
                  if (!Ctl_vel) set_PWM();
                  ficom=0;
            }
      } //end while
      return 0;
} //end main
```

A.2.3. Codi auxiliar

A continuació es detalla el codi de diverses funcions que son comunes als programes de tots dos dsPIC's. De les funcions aquí detallades algunes pertanyen a les llibreries de Microchip, aquestes s'han inclòs en aquest apartat donat que han estat modificades per tal d'optimitzar el funcionament per al cas concret del robot.

A.2.3.1. ACKI2C.C

Aquesta funció genera la condició d'aknowledge per una transmicó I^2C. Quan acabi de transmetre's es generarà automàticament una interrupció

```
/*********************************************************************
* Function Name:  AckI2C
* Description:    This routine generates acknowledge condition
*                 during master receive.
* Parameters:     void
* Return Value:   void
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void AckI2C(void)
{
      I2CCONbits.ACKDT = 0;
      I2CCONbits.ACKEN = 1;
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.2. NOTACK.C

Aquesta funció genera la condició de no aknowledge per una transmicó I2C. Quan acabi de transmetre's es generarà automàticament una interrupció

```
/*********************************************************************
* Function Name:  NotAckI2C
* Description:    This routine generates not acknowledge condition
*                 during master receive.
* Parameters:     void
* Return Value:   void
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void NotAckI2C(void)
{
      I2CCONbits.ACKDT = 1;
      I2CCONbits.ACKEN = 1;
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.3. CONFIGINTI2C.C

Amb aquesta funció s'aplica la configuració de les interrupcions del bus I^2C. El paràmetre d'entrada defineix quines interrupcions s'activen i amb quina prioritat.

```
/*********************************************************************
* Function Name:  ConfigIntI2C
* Description:    This routine enables/disables the SI2C & MI2C
*                 interrupts and sets their priorities
* Parameters:     unsigned int : config
* Return Value:   void
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void ConfigIntI2C(unsigned int config)
{
      /* clear the MI2C & SI2C Interrupts */
      _SI2CIF = 0;
      _MI2CIF = 0;
      _SI2CIP = (config & 0x0007); /* set the SI2C priority */
      _MI2CIP = (config & 0x0070) >> 4; /* set the MI2C priority */
      _SI2CIE = (config & 0x0008)>> 3; /* enable/disable SI2C Int */
      _MI2CIE = (config & 0x0080) >> 7; /* enable/disable MI2C Int */
}
#else
```

```
#warning "Does not build on this target"
#endif
```

A.2.3.4. IDLEI2C.C

Aquesta funció comprova que el mòdul I2C estigui lliure, en cas de no estar-hi genera una espera fins que s'acabin totes les transaccions iniciades.

```
/*********************************************************************
* Function Name:  IdleI2C
* Description:    This routine generates wait condition intil I2C
*                 bus is Idle.
* Parameters:     void
* Return Value:   void
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void IdleI2C(void)
{
      /* Wait until I2C Bus is Inactive */
      while(I2CCONbits.SEN || I2CCONbits.PEN || I2CCONbits.RCEN ||
      I2CCONbits.ACKEN || I2CSTATbits.TRSTAT);
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.5. MASTERREADI2C.C

Funció per rebre un byte pel bus I^2C com a mestre d'un esclau. En finalitzar la transmissió es genera una interrupció I^2C mestre.

```
/*********************************************************************
* Function Name:  MasterReadI2C
* Description:    This routine reads a single byte from the I2C Bus.
*                 To enable master receive,RCEN bit is set.
*                 The RCEN bit is checked until it is cleared.When
*                 cleared, the receive register is full and it's contents
*                 are returned.
* Parameters:     void
* Return Value:   unsigned char
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
unsigned char MasterReadI2C(void)
{
      I2CCONbits.RCEN = 1;
      while(I2CCONbits.RCEN);
```

```
	I2CSTATbits.I2COV = 0;
	return(I2CRCV);
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.6. MASTERWRITEI2C.C

Funció per enviar un byte pel bus I^2C com a mestre a un esclau. En finalitzar la transmissió es genera una interrupció I^2C mestre.

```
/*********************************************************************
* Function Name:  MasterWriteI2C
* Description:    This routine is used to write a byte to the I2C bus.
*                 The input parameter data_out is written to the
*                 I2CTRN register. If IWCOL bit is set,write collision
*                 has occured and -1 is returned, else 0 is returned.
* Parameters:     unsigned char : data_out
* Return Value:   unsigned int
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
char MasterWriteI2C(unsigned char data_out)
{
	I2CTRN = data_out;
	if(I2CSTATbits.IWCOL) /* If write collision occurs,return -1 */
	return -1;
	else
	{
		return 0;
	}
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.7. OPENI2C.C

Funció per inicialitzar i configurar el mòdul I^2C dels dsPIC's. Inclou la configuració correcta dels ports.

```
/*********************************************************************
* Function Name:  OpenI2C
* Description:    This function configures the I2C module for enable bit,
*                 disable slew rate, SM bus input levels, SCL release,
*                 Intelligent Peripheral Management Interface enable,
*                 sleep mode, general call enable,acknowledge data bit,
*                 acknowledge sequence enable, receive enable, stop
*                 condition enable, restart condition enable and start
*                 condition enable. The Baud rate value is also
*                 configured
* Parameters:     unsigned int : config1
*                 unsigned int : config2
* Return Value:   void
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void OpenI2C(unsigned int config1, unsigned int config2)
{
I2CBRG = config2;
I2CCON = config1;
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.8. SLAVEREADI2C

Llegeix un byte del bus I^2C. Quan s'acaba de rebre el byte es genera ina interrupció I^2C esclau.

```
/*********************************************************************
* Function Name:  SlaveReadI2C
* Description:    This routine reads a single byte from the I2C Bus.
*                 The RBF bit is checked until it is set.When set,
*                 the receive register is full and it's contents are
*                 returned.
* Parameters:     void
* Return Value:   unsigned char
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
unsigned char SlaveReadI2C(void)
{
```

```
      while(!I2CSTATbits.RBF);
      I2CSTATbits.I2COV = 0;
      return(I2CRCV);
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.9. SLAVEWRITEI2C.C

Envia un byte pel bus I^2C. Quan s'acaba d'enviar l'últim bit es genera una interrupció I^2C esclau. Respecte la funció original no s'allibera al bus I^2C dins aquesta funció. Aquesta acció es fa al final de la rutina d'interrupció esclau per tal d'assegurar que el mestre no envii el aknowledge abans d'hora.

```
/*********************************************************************
* Function Name:  SlaveWriteI2C
* Description:    This routine is used to write a byte to the I2C bus.
*                 The input parameter data_out is written to the
*                 I2CTRN register.
* Parameters:     unsigned char : data_out
* Return Value:   None
* Autor:          Microchip.inc
*********************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void SlaveWriteI2C(unsigned char data_out)
{
      I2CTRN = data_out; /* data transferred to I2CTRN reg */
      //I2CCONbits.SCLREL = 1; /* Release the clock */
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.10. STARTI2C.C

Genera la condició de Start en una transmissió I^2C. En finalitzar es genera una interrupció.

```
/*********************************************************************
* Function Name:  StartI2C
* Description:    This routine generates Start condition
*                 during master mode.
* Parameters:     void
* Return Value:   void
* Autor:          Microchip.inc
*********************************************************************/
#include "i2c.h"
#ifdef _MI2CIF
void StartI2C(void)
```

```
{
I2CCONbits.SEN = 1; /* initiate Start on SDA and SCL pins */
}
#else
#warning "Does not build on this target"
#endif
```

A.2.3.11. STOPI2C.C

Genera la condició de stop en una transmissió I^2C. En finalitzar es genera una interrupció.

```
/*****************************************************************
* Function Name:  StopI2C
* Description:    This routine generates Stop condition
*                 during master mode.
* Parameters:     void
* Return Value:   void
* Autor:          Microchip.inc
*****************************************************************/

#include "i2c.h"
#ifdef _MI2CIF
void StopI2C(void)
{
      I2CCONbits.PEN = 1; /* initiate Stop on SDA and SCL pins */
}
#else
#warning "Does not build on this target"
#endif
```

pid.s: Funció que implementa un control PID genèric. En el cas del Wifibot aquesta funció es fa servir per al control de velocitat.

```
/*****************************************************************
* Function Name:  pid
* Description:    Implementation of generic PIC control system
* Parameters:     struct tPID
* Return Value:   struct tPID
* Autor:          Microchip.inc
*****************************************************************/
```

```
;*****************************************************************************
;                                                                            *
; Software License Agreement                                                 *
;                                                                            *
; The software supplied herewith by Microchip Technology                     *
; Incorporated (the "Company") for its dsPIC controller                      *
; is intended and supplied to you, the Company's customer,                   *
; for use solely and exclusively on Microchip dsPIC                          *
; products. The software is owned by the Company and/or its                  *
; supplier, and is protected under applicable copyright laws. All            *
; rights are reserved. Any use in violation of the foregoing                 *
; restrictions may subject the user to criminal sanctions under              *
; applicable laws, as well as to civil liability for the breach of           *
; the terms and conditions of this license.                                  *
;                                                                            *
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO                      *
; WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING,              *
; BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND              *
; FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE               *
; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,            *
; INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.            *
;                                                                            *
; (c) Copyright 2003 Microchip Technology, All rights reserved.              *
;*****************************************************************************

; Local inclusions.
      .nolist
      .include "dspcommon.inc" ; fractsetup
      .list
      .equ offsetabcCoefficients, 0
      .equ offsetcontrolHistory, 2
      .equ offsetcontrolOutput, 4
      .equ offsetmeasuredOutput, 6
      .equ offsetcontrolReference, 8

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      .section .libdsp, code
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; controlOutput[n]     = controlOutput[n-1]
;                      + controlHistory[n] * abcCoefficients[0]
;                      + controlHistory[n-1] * abcCoefficients[1]
;                      + controlHistory[n-2] * abcCoefficients[2]
;
;where:
;     abcCoefficients[0] = Kp + Ki + Kd
;     abcCoefficients[1] = -(Kp + 2*Kd)
;     abcCoefficients[2] = Kd
;     controlHistory[n] = measuredOutput[n] - referenceInput[n]
; where:
;     abcCoefficients, controlHistory, controlOutput, measuredOutput and
;     controlReference are all members of the data structure tPID.
;
;Input:
;     w0 = Address of tPID data structure

; Return:
;     w0 = Address of tPID data structure
;
;System resources usage:
;     {w0..w5} used, not restored
```

```
;     {w8,w10} saved, used, restored
;     AccA, AccB used, not restored
;     CORCON saved, used, restored
;
;DO and REPEAT instruction usage.
;     0 level DO instruction
;     0 REPEAT intructions
;
;Program words (24-bit instructions):
;     28
;
;Cycles (including C-function call and return overheads):
;     30
;.....................................................................

      .global _PID      ; provide global scope to routine
_PID:
      ; Save working registers.
push        w8
push        w10
push        CORCON
; Prepare CORCON for fractional computation.
fractsetup  w8
mov         [w0 + #offsetabcCoefficients], w8
; w8 = Base Address of _abcCoefficients array [(Kp+Ki+Kd), -(Kp+2Kd), Kd]
mov         [w0 + #offsetcontrolHistory], w10
; w10 = Address of _ControlHistory array(state/delay line)
mov         [w0 + #offsetcontrolOutput], w1
mov         [w0 + #offsetmeasuredOutput], w2
mov         [w0 + #offsetcontrolReference], w3
; Calculate most recent error with saturation, no limit checking required
lac         w3, a       ; A = tPID.controlReference
lac         w2, b       ; B = tPID.MeasuredOutput
sub         a           ; A = tPID.controlRef. - tPID.measuredOut.
sac.r       a, [w10]    ; tPID.ControlHistory[n] = Sat(Rnd(A))
; Calculate PID Control Output
clr         a, [w8]+=2, w4, [w10]+=2, w5
                        ; w4 = (Kp+Ki+Kd), w5 =_ControlHistory[n]
lac         w1, a       ; A = ControlOutput[n-1]
mac         w4*w5, a, [w8]+=2, w4, [w10]+=2, w5
                        ; A += (Kp+Ki+Kd) *_ControlHistory[n]
                        ; w4 = -(Kp+2Kd), w5 =_ControlHistory[n-1]
mac         w4*w5, a, [w8], w4, [w10]-=2, w5
                        ; A += -(Kp+2Kd) *_ControlHistory[n-1]
                        ; w4 = Kd, w5 = _ControlHistory[n-2]
mac         w4*w5, a, [w10]+=2, w5
                        ; A += Kd * _ControlHistory[n-2]
                        ; w5 = _ControlHistory[n-1]
                        ; w10 = &_ControlHistory[n-2]
sac.r       a, w1       ; ControlOutput[n] = Sat(Rnd(A))
mov         w1, [w0 + #offsetcontrolOutput]
                        ;Update the error history on the delay line
mov         w5, [w10]   ; _ControlHistory[n-2] = _ControlHistory[n-1]
mov         [w10 + #-4], w5   ; _ControlHistory[n-1] = ControlHistory[n]
mov         w5, [--w10]
pop         CORCON            ; restore CORCON.
pop         w10               ; Restore working registers.
pop         w8
return
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _PIDInit:
;
;Prototype:
;     void PIDInit ( tPID *fooPIDStruct )
;
;Operation: This routine clears the delay line elements in the array
;           _ControlHistory, as well as clears the current PID output
;           element, _ControlOutput
;
;Input:
;     w0 = Address of data structure tPID (type defined in dsp.h)
;
;Return:
;     (void)
;
;System resources usage:
;     w0 used, restored
;
;DO and REPEAT instruction usage.
;     0 level DO instruction
;     0 REPEAT intructions
;
;Program words (24-bit instructions):
;     11
;
;Cycles (including C-function call and return overheads):
;     13
;.....................................................................

      .global _PIDInit ; provide global scope to routine
_PIDInit:
push        w0
add         #offsetcontrolOutput, w0
                  ; Set up pointer for controlOutput
clr         [w0]  ; Clear controlOutput
pop         w0
push        w0
;Set up pointer to the base of
;controlHistory variables within tPID
mov         [w0 + #offsetcontrolHistory], w0
                        ; Clear controlHistory variables
                        ; within tPID
clr         [w0++]      ; ControlHistory[n]=0
clr         [w0++]      ; ControlHistory[n-1] = 0
clr         [w0]        ; ControlHistory[n-2] = 0
pop         w0          ;Restore pointer to base of tPID
return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _PIDCoeffCalc:
;
;Prototype:
;     void PIDCoeffCalc (fractional *fooPIDGainCoeff, tPID *fooPIDStruct)
;
;Operation: This routine computes the PID coefficients to use based on
;           values of Kp, Ki and Kd provided. The calculated coefficients
;           are:
;                 A = Kp + Ki + Kd
;                 B = -(Kp + 2*Kd)
;                 C = Kd
```

```
; Input:
; w0 = Address of coefficient array containing Kp, Ki and Kd in sequence
; w1 = Address of PID data structure, tPID (type defined in dsp.h)
;
;Return:
;     (void)
;
;System resources usage:
;     {w0..w2} used, not restored
;     AccA, AccB used, not restored
;     CORCON saved, used, restored
;
;DO and REPEAT instruction usage.
;     0 level DO instruction
;     0 REPEAT intructions
;
;Program words (24-bit instructions):
;     18
;
;Cycles (including C-function call and return overheads):
;     20
;.....................................................................

      .global _PIDCoeffCalc
_PIDCoeffCalc:
push        CORCON ; Prepare CORCON for fractional computation.
fractsetup  w2
      ;Calculate Coefficients from Kp, Ki and Kd
mov         [w1], w1
lac         [w0++], a   ; ACCA = Kp
lac         [w0++], b   ; ACCB = Ki
add         a           ; ACCA = Kp + Ki
lac         [w0--], b   ; ACCB = Kd
add         a           ; ACCA = Kp + Ki + Kd
sac.r       a, [w1++]   ; _abcCoefficients[0] = Sat(Rnd(Kp + Ki + Kd))
lac         [--w0], a   ; ACCA = Kp
add         a           ; ACCA = Kp + Kd
add         a           ; ACCA = Kp + 2Kd
neg         a           ; ACCA = -(Kp + 2Kd)
sac.r       a, [w1++]   ; _abcCoefficients[1] = Sat(Rnd(-Kp - 2Kd))
sac         b, [w1]     ; _abcCoefficients[2] = Kd
pop         CORCON
return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.end
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; OEF
```

A.2.3.12. INIT_ADC.C

Inicialitza el mòdul del conversor analògic-digital dels dsPIC's. Aplica la corresponent configuració per tal de llegir el valor dels sensors de distancia del robot. Iclou la configuració dels ports i de les interrupcions.

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    init_AD.c                                   //
// AUTOR:         Rafael Hesse                                //
// DATA:          29/04/2008                                  //
// PLATAFORMA:    dsPIC Wifibot                               //
// LLENGUATGE:    C, compilador C30 Microchip                 //
// DESCRIPCIÓ:    Inicialitza el mòdul ADC per convertir les  //
//                lectures dels sensors de distància          //
////////////////////////////////////////////////////////////////

/////////////////MODE DE FUNCIONAMENT//////////////////////////
//Tenim 4 entrades i 16 buffers de lectura. Llegim cada canal 4 cops
//abans no es genera interrupció. Així podem calcular la distància
//a partir de la mitja de 4 lectures.
//No podem fer servir el mode automàtic donat que el temps de mostreig
//màxim en aquest mode ronda els 1.4ms, temps insuficient per fer els
//càlculs per convertir a centímetres.
//El timer 3 inicia una conversió cada 5ms de forma que tenim una
//interrupció (lectura vàlida) cada 20ms

#include <p30f2010.h>
void init_AD (void)
{
      ADCON1bits.ADSIDL = 0;  //Continua durant IDLE
      ADCON1bits.FORM = 0;    //Format de sortida com a integer
      ADCON1bits.SSRC = 0;    //Sample begins at end of conversion
                              //and conversion begins on clearing SAM bit
      ADCON1bits.SIMSAM = 0;  //Mode individual seqüencial
      ADCON1bits.ASAM = 1;    //Bit SAMP es automàtic
      ADCON1bits.SAMP = 1;    //Start of first sample
      ADCON2bits.VCFG = 0;    //Referencia a AVdd and AVss
      ADCON2bits.CSCNA = 0;   //No fer scan al canal 0
      ADCON2bits.CHPS = 3;    //Convert CH0, 1, 2 and 3
      ADCON2bits.SMPI = 15;   //Interrupt freq.

      /*************************************************************
      SMPI=16 --> cada interrupcio son 16 conversions aixi podem fer
      mitja entre 4 valors de cada sensor.
            ADCBUF0 , ADCBUF4 , ADCBUF8 , ADCBUFC --> AN3
            ADCBUF1 , ADCBUF5 , ADCBUF9 , ADCBUFD --> AN0
            ADCBUF2 , ADCBUF6 , ADCBUFA , ADCBUFE --> AN1
            ADCBUF3 , ADCBUF7 , ADCBUFB , ADCBUFF --> AN2
      *************************************************************/

      ADCON2bits.BUFM = 0;    //16 bit buffer
      ADCON2bits.ALTS = 0;    //Only MUX A
      ADCON3bits.SAMC = 31;   //T. entre mostreig i conversio (SAMC*TAD)
      ADCON3bits.ADRC = 0;    //Use system clk
```

```
/*************************************************************
        Tcy
TAD = ------- * (ADCS + 1)
         2
              TAD
ADCS = 2 * ------- -1
              Tcy
ADCS=Maxim=63 --> TAD=4.34us
temps_conversio = 12 * TAD
freq.conversió = 5ms (controlat per timer 3)
Comprovem que temps de lectura no superi freq. conversió:
Temps per una lectura = (SAMC * TAD) + (nº_canals * temps_conversio)
                      = (31*3.34us) + (4*12*3.34us) = 212us < 5ms --> OK!
                            SMPI+1                       16
Temps per una interrupció = -------- * freq.conversio = ---- * 5ms = 20ms
                           nº_canals                      4
*************************************************************/

      ADCON3bits.ADCS = 63;   //Conversion clk
      ADCHSbits.CH123NA = 0;  //CH1(-)=CH2(-)=CH3(-)= -VREF
      ADCHSbits.CH123SA = 0;  //CH1(+)=AN0 CH2(+)=AN1 CH3(+)=AN2
      ADCHSbits.CH0NA = 0;    //CH0(-)=-VREF
      ADCHSbits.CH0SA = 3;    //CH0(+)=AN3
      TRISBbits.TRISB0 = 1;   //Pin es entrada
      TRISBbits.TRISB1 = 1;   //Pin es entrada
      TRISBbits.TRISB2 = 1;   //Pin es entrada
      TRISBbits.TRISB3 = 1;   //Pin es entrada
      ADPCFGbits.PCFG0 = 0;   //Pin as AD input
      ADPCFGbits.PCFG1 = 0;   //Pin as AD input
      ADPCFGbits.PCFG2 = 0;   //Pin as AD input
      ADPCFGbits.PCFG3 = 0;   //Pin as AD input
      ADCSSLbits.CSSL0 = 1;   //Select chanel
      ADCSSLbits.CSSL1 = 1;   //Select chanel
      ADCSSLbits.CSSL2 = 1;   //Select chanel
      ADCSSLbits.CSSL3 = 1;   //Select chanel
      _ADIE = 1;              //Enable interrupt
      _ADIP = 2;              //Interrupt priority
      ADCON1bits.ADON = 1;    //Activate chanel
}
```

A.2.3.13. INIT_PWM.C

Inicialitza els corresponents mòduls PWM per tal de controlar la velocitat dels motors del robot. Inclou la configuració dels ports i les interrupcions.

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    init_PWM.c                                  //
// AUTOR:         Rafael Hesse                                //
// DATA:          04/05/2008                                  //
// PLATAFORMA:    dsPIC Wifibot                               //
// LLENGUATGE:    C, compilador C30 Microchip                 //
// DESCRIPCIÓ:    Inicialitza el mòdul PWM_1 i PWM_2 dels dsPIC's//
//                per controlar la velocitat dels motors      //
////////////////////////////////////////////////////////////////

#include <p30f2010.h>
void init_PWM (void)
```

```
{
      _PWMIE = 0;             //Sense interrupcions
      PTCONbits.PTMOD = 0;    //Free running mode
      PTCONbits.PTCKPS = 0;   //Prescaler=1
      /*****************************************************
                  Fcy               7372800
      F.PWM = ------------------ = ---------- = 24.576 Hz
               PTPER * Prescaler     300 * 1
      *****************************************************/
      PTPER = 300;
      PWMCON1bits.PEN1H = 1;  //Activem els PWM's 1 i 2
      PWMCON1bits.PEN1L = 1;
      PWMCON1bits.PEN2H = 1;
      PWMCON1bits.PEN2L = 1;
      PWMCON1bits.PEN3H = 0;  //El tercer PWM no es fa servir
      PWMCON1bits.PEN3L = 0;
      DTCON1bits.DTAPS = 0;   //Dead time prescaler = Tcy
      DTCON1 = DTCON1 | 0x02; //Dead time value
      OVDCONbits.POVD1L = 1;  //No oveerride
      OVDCONbits.POVD1H = 1;
      OVDCONbits.POVD2L = 1;
      OVDCONbits.POVD2H = 1;
      PWMCON1bits.PMOD1 = 1;  //independent mode
      PWMCON1bits.PMOD2 = 1;  //independent mode
      PTCONbits.PTEN = 1;     //Enable PWM
}
```

A.2.3.14. INIT_T_QEI.C

Inicialitza el mòdul de lectura dels encoders (QEI), esl timers que actúen com a comptadors dels encoders i el timer 3 que genera una interrupció cada 5 ms per al control de velocitat.

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    init_T_QEI.c                                //
// AUTOR:         Rafael Hesse                                //
// DATA:          04/05/2008                                  //
// PLATAFORMA:    dsPIC Wifibot                               //
// LLENGUATGE:    C, compilador C30 Microchip                 //
// DESCRIPCIÓ:    Inicialització dels timers 1, 2 i 3 i del mòdul//
//                QEI dels dsPIC's                            //
////////////////////////////////////////////////////////////////

#include <p30f2010.h>
void init_T_QEI(void)
{
      //INICIALITZACIÓ DEL MÒDUL QEI
      _QEIM = 0;        //Disable for configuration
      _CNTERR = 0;      // Clear any count errors
      _QEISIDL= 0;      //Continue Operation in Idl mode
      _CEID = 1;        //Dissable interrupt on error
      _PCDOUT = 0;      // Normal I/O pin operation
      _PCFG4 = 1;       //Pin as digital I/O
      _PCFG5 = 1;       //Pin as digital I/O
      _TRISB4 = 1;      //Pin as input
      _TRISB5 = 1;      //Pin as input
```

```
        _POSRES = 0;      //No position reset on index pulse
        _IMV = 0;         //State of chA and chB on index pulse
        _QEOUT = 1;       //Digital filter enable for QEA/QEB
        _QECK = 5;        //CLK divider for filter QEA/QEB
        _QEIIE = 0;       //Disable interrupt
        _SWPAB = 1;       //QEA and QEB swaped
        MAXCNT = 0xFFFF;  //Comptatge màxim
        _QEIM = 7;        //Enable x4 mode with reset on MAXCNT match

        //INICIALITZACIÓ DEL TIMER 1 (CANAL A DE L'ENCODER)
        T1CONbits.TSIDL = 0;    //Continue operatin during sleep
        T1CONbits.TGATE = 0;    //Not gated
        T1CONbits.TCKPS = 0;    //Prescaler 1:1
        T1CONbits.TSYNC = 0;    //Asynchronous mode
        T1CONbits.TCS = 1;      //External pin as clock source
        TMR1 = 0;               //Reset counting value
        PR1 = 10000;            //Generate interrupt on value
        _T1IE = 1;              //Enable interrups
        _T1IP = 2;              //Interrupt priority
        _T1IF = 0;              //Clear flag
        T1CONbits.TON = 1;      //Enable module

        //INICIALITZACIÓ DEL TIMER 2 (CANAL B DE L'ENCODER
        T2CONbits.TSIDL = 0;    //Continue operatin during sleep
        T2CONbits.TGATE = 0;    //Not gated
        T2CONbits.TCKPS = 0;    //Prescaler 1:1
        T2CONbits.T32 = 0;      //16 bit mode
        T2CONbits.TCS = 1;      //External pin as clock source
        TMR2 = 0;               //Reset counting value
        PR2 = 10000;            //Generate interrupt on value
        _T2IE = 1;              //Enable interrups
        _T2IP = 2;              //Interrupt priority
        _T2IF = 0;              //Clear flag
        T2CONbits.TON = 1;      //Enable module

        //INICIALITZACIÓ DEL TIMER 3
        T3CONbits.TSIDL = 0;    //Continue operatin during sleep
        T3CONbits.TGATE = 0;    //Not gated
        T3CONbits.TCKPS = 0;    //Prescaler 1:1
        T3CONbits.TCS = 0;      //Tcy as clock source
        TMR3 = 0;               //Reset counting calue

        /*****************************************************
        Count value:
                    T
        PR3 = ------------- : prescaler
                    1
                 -------
                 7372800
        T = 50ms --> prescaler = 8 --> PR3 = 46080
        T = 20ms --> prescaler = 8 --> PR3 = 18432
        T = 10ms --> prescaler = 8 --> PR3 = 9216
        T = 5ms --> prescaler = 1 --> PR3 = 36864
        *****************************************************/

        PR3 = 36864;            //Generate interrupt on value
        _T3IE = 1;              //Enable interrups
        _T3IP = 3;              //Interrupt priority
        _T3IF = 0;              //Clear flag
        T3CONbits.TON = 1;      //Enable module
}
```

A.2.3.15. RETARD.C

Funció per generar diversos retards. Tot i que es definiesen unitats en les funcions no es poden entendre com a valors exactes, la durada exacte del retard generat depèn de les directives de compilació.

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    retard.c                                    //
// AUTOR:         Rafael Hesse                                //
// DATA:          04/05/2008                                  //
// PLATAFORMA:    dsPIC Wifibot                               //
// LLENGUATGE:    C, compilador C30 Microchip                 //
// DESCRIPCIÓ:    Generació de retards en ms, segons i cicles.  //
////////////////////////////////////////////////////////////////

#define CLKtics 73728
#define m_seconds CLKtics/140
#define seconds (CLKtics*100/14)
void Delay_ms (unsigned int count)
{
      unsigned long n;
      n=(unsigned long)count*(unsigned long)m_seconds;
      while(n>0) n--;
}
void Delay_i (unsigned int count)
{
      unsigned int i;
      for (i=0;i<=count;i++);
}
void Delay_s (unsigned int count)
{
      unsigned long n;
      n=(unsigned long)count*(unsigned long)seconds;
      while(n>0) n--;
}
```

A.2.4. Programa pont

```
////////////////////////////////////////////////////////////////////////
// Nom del programa:    pont.c                                        //
// Plataforma:          Nylon Linux del Wifibot                       //
// Paràmetre d'entrada: Cap                                           //
// Autor:               Rafael Hesse                                  //
// Data:                08/02/2008                                    //
// Versió:              1.0                                           //
// Compilador:          GCC cross compiler mipsel-linux-gcc           //
// Dependències:        Nova arquitectura del Wifibot i nova          //
//                      programació dels dsPIC's                      //
// Comentaris:          Actua com a pont de comunicacions entre el bus //
//                      I2C i el Wifi. Permet l'interacció des de fora //
//                      i a distància amb el robot.                   //
////////////////////////////////////////////////////////////////////////

//INCLUDES
#include "wifibot2.h"
#include <sys/time.h>

/*
action: estructura del tipus sigaction (estandar per maneig de senyals en
linux GNU)
sa_restirer: parametre obsolet, no el fem servir
sa_handler: Apunta a la funció que manetjarà la senyal capturada
sa_mask: Proporciona una màscara per bloquejar determinades senyals per
al
manejador
sa_flags: Conjunt de flags que permeten condicionar el manejador
signum: Senyal que es capturarà per tractar amb el manejador
*/

//Nombre de fils per executar simultàniament
#define NB_THREADS 2
//Temps màxim d'espera a comunicacions
#define MAXPENDING 5

//DECLARACIÓ DE LES FUNCIONS
void gestio_int (int);
void I2Ctransit (void);
void initI2C (void);
void convertir_bateria (void);
void vompoa_buffsend (void);
void vompoa_buffsendPF (void);
void vompoa_buffsendPR (void);
void gestio_int (int);
void * fn_thread_dog (void * numero);
void * fn_thread_com (void * numero);
pthread_t thread [NB_THREADS];
pthread_mutex_t mutex_dog = PTHREAD_MUTEX_INITIALIZER;

//DECLARACIÓ DE LES VARIABLES GLOBALS
unsigned char buffsendPF[14];      //buffer escriptura dsPIC davant
unsigned char buffrcvPF[13];       //buffer lectura dsPIC darrera
unsigned char buffsendPR[4];       //buffer escriptura dsPIC darrera
unsigned char buffrcvPR[8];        //buffer lectura dsPIC darrera
unsigned char buffsendAD[2];       //buffer escriptura conversor AD
```

```
unsigned char buffrcvAD[1];        //buffer lectura conversor AD
unsigned char buffsend[21];        //buffer escriptura PC
unsigned char buffrcv[18];         //buffer lectura PC
unsigned char bateria;             //Valor en volts * 10 de la bateria

//Estructura per configurar el bus I2C
struct i2c_rdwr_ioctl_data work_queue;
static struct sockaddr_in echoServAddr; //Local address
static struct sockaddr_in echoClntAddr; //Client address
int i;                        //Comptador
int ret;                      //Variable de retorn de les comunicacions
int dog=0;                    //Comptador
static int connected=0;       //Flag que indica comunicació
static int servSock;          //Socket descriptor for server
static int clntSock;          //Socket descriptor for client
static unsigned short echoServPort=15000;      //Server port
static unsigned int clntLen;  //Length of client address data
static int recvMsgSize=0;

//RUTINA PER CONVERTIR LA LECTURA DEL CONVERSOR AD EN VOLTS * 10
void convertir_bateria (void)
{
      /*******************************************************
                              lectura * 5    lectura * 25
      Vbat=Vain*2.5 = 2.5 * ------------- = --------------
                                  255              51
      *******************************************************/
      bateria=(25*buffrcvAD[0])/51;
}

//RUTINA PER COMPOSAR EL BUFFER DE ESCRIPTURA PC
void composa_buffsend (void)
{
      buffsend[0] = bateria;
      buffsend[1] = buffrcvPF[0]; //Codi error dsPIC davant
      buffsend[2] = buffrcvPR[0]; //Codi error dsPIC darrera
      buffsend[3] = buffrcvPF[1]; //Encoder esquerra davant
      buffsend[4] = buffrcvPR[1]; //Encoder esquerra darrera
      buffsend[5] = buffrcvPF[2]; //Encoder dreta davant
      buffsend[6] = buffrcvPR[2]; //Encoder dreta darrera
      buffsend[7] = buffrcvPF[3]; //Sens_1
      buffsend[8] = buffrcvPR[3]; //Sens_2
      buffsend[9] = buffrcvPF[4]; //Sens_3
      buffsend[10] = buffrcvPR[4]; //Sens_4
      buffsend[11] = buffrcvPF[5]; //Sens_5
      buffsend[12] = buffrcvPR[5]; //Sens_6
      buffsend[13] = buffrcvPR[6]; //Sens_7
      buffsend[14] = buffrcvPF[6]; //Sens_8
      buffsend[15] = buffrcvPF[7]; //Pos x cm + signe
      buffsend[16] = buffrcvPF[8]; //Pos y cm + signe
      buffsend[17] = buffrcvPF[9]; //Pos x m
      buffsend[18] = buffrcvPF[10]; //Pos y m
      buffsend[19] = buffrcvPF[11]; //Pos T
      buffsend[20] = (buffrcvPR[7]&0x38) | ((buffrcvPF[12]&0x38)>>3);
      //Ports auxiliars
}

//RUTINA PER COMPOSAR EL BUFFER DE ESCRIPTURA DEL dsPIC DE DAVANT
void composa_buffsendPF (void)
{
      buffsendPF[0] =((buffrcv[1]&0x01)<<7) | ((buffrcv[0]&0x20)) |
```

```
                    ((buffrcv[1]&0x02)<<3) | ((buffrcv[1]&0x08)<<3) |
                    ((buffrcv[0]&0x04)<<1) | ((buffrcv[0]&0x02)<<1) |
                    ((buffrcv[0]&0x01)<<1) | ((buffrcv[1]&0x10)>>4);
                    //Configuració
      buffsendPF[1] = buffrcv[2];   //Consigna motor esquerra
      buffsendPF[2] = buffrcv[3];   //Consigna motor dreta
      buffsendPF[3] = buffrcv[6];   //Consigna x cm + signe
      buffsendPF[4] = buffrcv[7];   //Consigna y cm + signe
      buffsendPF[5] = buffrcv[8];   //consigna x m
      buffsendPF[6] = buffrcv[9];   //consigna y m
      buffsendPF[7] = buffrcv[10];  //consigna T
      buffsendPF[8] = buffrcv[11];  //Posició x cm + signe
      buffsendPF[9] = buffrcv[12];  //Posició y cm + signe
      buffsendPF[10] = buffrcv[13]; //Posició x m
      buffsendPF[11] = buffrcv[14]; //Posició y m
      buffsendPF[12] = buffrcv[15]; //Posició T
      buffsendPF[13] = buffrcv[16]; //Ports auxiliars
}

//RUTINA PER COMPOSAR EL BUFFER DE ESCRIPTURA DEL dsPIC DE DARRERA
void composa_buffsendPR (void)
{
      buffsendPR[0] = ((buffrcv[1]&0x10)<<2) | ((buffrcv[0]&0x40)>>1) |
                      ((buffrcv[1]&0x04)<<2) | ((buffrcv[0]&0x04)<<1) |
                      ((buffrcv[0]&0x02)<<1) | ((buffrcv[0]&0x01)<<1) |
                      (0x00); //Configuració
      buffsendPR[1] = buffrcv[4]; //Consigna motor esquerra
      buffsendPR[2] = buffrcv[5]; //Consigna motor dreta
      buffsendPR[3] = buffrcv[17]; //Ports auxiliars
}

//RUTINA PRINCIPAL
int main(int argc, char *argv[])
{
      // Definició de variables locals
      struct itimerval val;        // Configuració del timer
      struct sigaction interrupcio; // Configuració d'interrupcions
      struct sigaction action;
      int k=0;

      //Inicialitzem tots els buffers
      while (k<=14)
      {
            buffsendPF[k]=0;
            k++;
      }
      k=0;
      while (k<=4)
      {
            buffsendPR[k]=0;
            k++;
      }
      k=0;
      while (k<=13)
      {
            buffrcvPF[k]=0;
            k++;
      }
      k=0;
      while (k<=8)
      {
```

```
        buffrcvPR[k]=0;
        k++;
}
k=0;
while (k<=21)
{
        buffsend[k]=0;
        k++;
}
k=0;
while (k<=18)
{
        buffrcv[k]=0;
        k++;
}
work_queue.nmsgs = 6; //definim 6 paquets de dades per al bus I2C
printf("running....\n");
//Apliquem estructura de configuració
work_queue.msgs =
  (struct i2c_msg*)malloc(work_queue.nmsgs*sizeof(struct i2c_msg));
//Enviem els dos bytes de configuració al conversor AD
buffsendAD[0] = 0x40;//0 1 0 0 0 0 0 0
buffsendAD[1] = 250; //1 1 1 1 1 0 1 0
work_queue.msgs[0].addr = 0x48;
work_queue.msgs[0].len = 2;
work_queue.msgs[0].flags = 0;
work_queue.msgs[0].buf = buffsendAD;
//Rebem el resultat del conversor AD
work_queue.msgs[1].addr = 0x48;
work_queue.msgs[1].len = 1;
work_queue.msgs[1].flags = 1;
work_queue.msgs[1].buf = buffrcvAD;
//Enviem 4 bytes al PIC de darrera
work_queue.msgs[2].addr = 0x51;
work_queue.msgs[2].len = 4;
work_queue.msgs[2].flags = 0;
work_queue.msgs[2].buf = buffsendPR;
//Rebem 8 bytes de el PIC de darrera
work_queue.msgs[3].addr = 0x51;
work_queue.msgs[3].len = 8;
work_queue.msgs[3].flags = 1;
work_queue.msgs[3].buf = buffrcvPR;
//Enviem 14 bytes al PIC de devant.
work_queue.msgs[4].addr = 0x52;
work_queue.msgs[4].len = 14;
work_queue.msgs[4].flags = 0;
work_queue.msgs[4].buf = buffsendPF;
//Rebem 14 bytes de el PIC de devant.
work_queue.msgs[5].addr = 0x52;
work_queue.msgs[5].len = 13;
work_queue.msgs[5].flags = 1;
work_queue.msgs[5].buf = buffrcvPF;
// Configurem el timer per que doni una alarma cada 100 ms
val.it_interval.tv_sec=0;
val.it_interval.tv_usec=100000;
val.it_value.tv_sec=0;
val.it_value.tv_usec=100000;
setitimer(ITIMER_REAL, &val, NULL);
initI2C(); // Inicialització de les funcions I2C
//Inicialitzem el fil que funcionarà a mode de "watchdog"
if ((ret = pthread_create (& thread [0], NULL, fn_thread_dog,
```

```
            (void *) 0)) != 0) exit(1);
            // si ret = 0 el fil s'ha creat correctament
      //Inicialitzem el fil que gestionarà les comunicacions
      if ((ret = pthread_create (& thread [1], NULL, fn_thread_com,
            (void *) 1)) != 0) exit(1);
            // si ret = 0 el fil s'ha creat correctament
      //Interrupció per premer les tecles control
      interrupcio . sa_handler = gestio_int;
      // El manejador de les senyals capturades es la funcio gestio_int
      sigemptyset (& (interrupcio . sa_mask));
      // Especifiquem que no es bloquejarà cap senyal
      interrupcio . sa_flags = 0; // Tots els flags desactivats
      if (sigaction (SIGQUIT, & interrupcio, NULL) != 0) exit (1);
      // Capturem la senyal SIGQUIT causada per premer les tecles control
      //Interrupció per premer les tecles control + c
      interrupcio . sa_handler = gestio_int;
      // El manejador de les senyals capturades es la funcio gestio_int
      sigemptyset (& (interrupcio . sa_mask));
      // Especifiquem que no es bloquejarà cap senyal
      interrupcio . sa_flags = SA_RESTART | SA_RESETHAND;
      // Flags permeten reiniciar el programa despres de l'interrupcio
      if (sigaction (SIGINT, & interrupcio, NULL) != 0) exit (1);
      // Capturem la senyal SIGINT causada per premer Ctrl + c

      //Interrupció cada 50ms
      interrupcio . sa_handler = gestio_int;
      // El manejador de les senyals capturades es la funcio gestio_int
      sigemptyset (& (interrupcio . sa_mask));
      // Especifiquem que no es bloquejarà cap senyal
      interrupcio . sa_flags = 0; // Tots els flags desactivats
      if (sigaction (SIGALRM, & interrupcio, NULL) != 0) exit (1);
      // Capturem la senyal SIGALRM causada pel timer cada 50ms

      while ( 1 ) //Bucle infinit
      {
      }

      pthread_join (thread [0], NULL);
      // main queda en suspensió fins que el fil acabi o es suspengui
      pthread_join (thread [1], NULL);
      // main queda en suspensió fins que el fil acabi o es suspengui
      return 0;
} // Fi main

// Fil [0]
// Aquest fil supervisa l'execució del fil [2], la variable dog augmenta
un cop per segon, si arriba a valer 3 es para tot el procés
void * fn_thread_dog (void * num)
{
      int numero = (int) num;
      while (1)
      {
            sleep (1);                          // Espera un segon
            pthread_mutex_lock (& mutex_dog);   // Reservem variable dog
            dog ++;                             // Augmentem el contador
            pthread_mutex_unlock (& mutex_dog); // Es suspen la reserva
            if (dog>3) // S'ha superat el temps: Es suspen la comunicació
            {
                  dog=0;
                  if (connected) // Si estavem conectats ens desconectem
                  {
```

```
                connected=0;
                int ret=pthread_cancel (thread[1]);
                // Cancelem l'execució del fil [1]
                close(clntSock);  // Tanquem el socket de client
                close(servSock);  // Tanquem el socket de servidor
                sleep(2);         // Esperem 2 seg. reiniciem el fil
                                  comunicacions
                if ((ret = pthread_create (& thread [1], NULL,
                      fn_thread_com, (void *) 1)) !=0) exit (1);
                printf ("\n\n********Error timeout**********\n\n");
                }
          }
      }
      pthread_exit (NULL);
}

// Fil [1]
// Fil de comunicacions
void * fn_thread_com (void * num)
{
      int numero = (int) num;
      int comptador = 0;
      int autoritzacio=1;
      // Autoritzem les peticions de comunicació entrants
      // Creem el socket servidor per atendre les peticions entrents,
      alhora comprovem si ha donat error (cas per al qual retorna -1)
      servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
      // PF_INET = domini de la comunicació
      // SOCK_STREAM = transmissió de "streams"
      // IPPROTO_TCP = Protocol de comunicació TCP/IP
      // Construim una estructura que contindrà la adreça local
      memset(&echoServAddr, 0, sizeof(echoServAddr));
      // Omplim de zeros la estructura
      echoServAddr.sin_family = AF_INET; // Familia de direccions de
            internet de qualsevol tipus
      echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); // IP qualsevol
      echoServAddr.sin_port = htons(echoServPort); // Port local de
            comunicació (per defecte 15000)
      // Relacionem el socket amb una direcció local
      setsockopt(servSock,SOL_SOCKET,SO_REUSEADDR,&autoritzacio,sizeof
            (int)); // Configura les opcions del socket
      bind(servSock, (struct sockaddr *) &echoServAddr,sizeof
            (echoServAddr)); //lliguem el socket amb la adreça local

      for (;;) // Bucle infinit
      {
      // Posem el socket servidor a la escolta
      listen(servSock, MAXPENDING);
      clntLen = sizeof(echoClntAddr); // Definim la longitud dels
            missatges a rebre
      // Ens posem a la espera de rebre un misatge del client
      clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr,
            &clntLen); // En el moment en que s'estableix comunicació fem
      do
      {
      // Si el missatge no te la longitud esperada donem un error
      if ((recvMsgSize = recv(clntSock, buffrcv, 18, 0)) < 1)
      {
            shutdown(clntSock,1);
            printf("Error en rebre dades del client\n");
      }
```

```
	// Si el missatge es correcte proseguim a tractar-lo
	else
	{
		//printf("rebut Con_ER=%d\n",buffrcv[4]);
		pthread_mutex_lock (& mutex_dog); // Reservem la variable dog
			per aquest fil
		connected=1;
		dog =0; // Posem a 0 el comtador dog (aixi evitem que el fil
			[1] talli la comunicació
		pthread_mutex_unlock (& mutex_dog); //Alliberem la var. dog
		send(clntSock,buffsend,21,0); // Enviem dades al client
			conectat
	}
	}
	while(recvMsgSize>0);
	connected=0;
	shutdown(clntSock,1);
	close(clntSock);
	sleep(1);
	}//end for(;;)
	pthread_exit (NULL);
}//end thread

// FUNCIONS DEL BUS I2C

// Inicialització del bus I2C
void initI2C()
{
	sprintf(filename3,"/dev/i2c/%d",i2cbus);
	if ((file = open(filename3,O_RDWR)) < 0) printf("***Error al
		iniciar I2C\n");
}

void I2Ctransit()
{
	int ret;
	work_queue.msgs[0].buf = buffsendPF;
	ret = ioctl(file,I2C_RDWR,&work_queue);
	if( ret < 0)
	{
		printf("***Error I2C ******\n", ret);
	}
}

// Gestió de les senyals capturades
void gestio_int (int numero)
{
switch (numero)
{
	case SIGQUIT :
		break;
	case SIGINT :
		break;
	case SIGALRM :
		composa_buffsendPF();
		//Composem el buffer per enviar al dsPIC de davant
		composa_buffsendPR();
		//Composem el buffer per enviar al dsPIC de darrera
		composa_buffsend();
		//Actualitzem el buufer per enviar dades al PC
		I2Ctransit(); //Enviem i llegim dades del bus I2C
```

```
            convertir_bateria();
            break;
}
}
```

A.2.5. Makefile per al programa pont

```
#############################################################################
## Nom del programa:    Makefile                                           ##
## Plataforma:          Gentoo Linux                                       ##
## Autor:               Rafael Hesse                                       ##
## Data:                08/02/2008                                         ##
## Versió:              1.0                                                ##
## Compilador:          GCC cross compiler mipsel-linux-gcc                ##
## Dependències:        Cap                                                ##
## Comentaris:          Directrius per compilar el programa pont.c         ##
#############################################################################

CC =/stuff/tmp/cross/bin/mipsel-linux-gcc
CFLAGS =-c -Wall

objects = pont.o
## Compila i envia per ethernet al robot
all_lan :   pont send_lan clean
## Compila i envia per Wifi al robot
all_wifi :  pont send_wifi clean
## Només compila
pont:       $(objects)
            $(CC) -o gir $(objects) -lpthread -lm

pont.o:     pont.c
            $(CC) -c pont.c

## Envia per ethernet
send_lan:
            scp ./ pont root@192.168.0.103:./
## Envia per Wifi
send_wifi:
            scp ./ pont root@192.168.1.103:./
## Neteja
clean:
            rm -f *.o
```

A.2.6. Programa per la Nintendo DS

```
//////////////////////////////////////////////////////////
// NOM FITXER:    main.c                                //
// AUTOR:         Rafel Hesse                           //
// DATA:          23/05/2008                            //
// PLATAFORMA:    Nintendo DS                           //
// LLENGUATGE:    C, Compilador PA_lib                  //
// DESCRIPCIÓ:    Control del Wifibot amb nou hardware  //
//////////////////////////////////////////////////////////

// Includes
#include <PA9.h> // Include for PA_Lib
#include <pthread.h>
```

```
#include "gfx/all_gfx.c" //Inclusió dels gràfics
#include "gfx/all_gfx.h" //Inclusió dels gràfics

//DECLARACIÓ DE VARIABLES GLOBALS
char text[50];                  //Text
Wifi_AccessPoint AP;            //Estructura de dades sobre un AP
Wifi_AccessPoint conectAP;      //Estructura de dades sobre un AP
u8 Ctl_vel=0;                   //Bit de control de velocitat PID
u8 Ctl_obs=0;                   //Bit de control d'obstacles
u8 Ctl_gir=0;                   //Bit de control de gir lliure
u8 VL=50;                       //Velocitat per al desplaçament en linia
recta
u8 VG=50;                       //Velocitat de gir suau
u8 VGL=50;                      //Velocitat de gir sobre el propi eix
u16 numAP;                      //Quantitat de AP dins la cobertura wifi
u16 j;                          //Comptador
u32 dsIP;                       //IP de la NDS
u32 rbIP;                       //IP del Wifibot
u32 rbPORT;                     //Port del robot
u32 dsGW;                       //Porta d'enllaç de la NDS
u32 dsSM;                       //Màscara de subxarxa de la NDS
u32 dsDNS1;                     //DNS 1 de la NDS
u32 dsDNS2;                     //DNS 2 de la NDS
u32 naddr;                      //Adreça editada (variable temporal
s8 selection;                   //Guarda el AP seleccionat
u8 buffsend[18];                //Buffer de dades enviades al robot
u8 buffrcv[21];                 //Buffer de dades rebudes del robot
s32 nletter = 0;                //Comptador de entrades del teclat virtual
u8 letter = 0;                  //Tecla 'premuda del teclat virtual
u8 estat=1;                     //Estat de l'execució
u8 Error=0;                     //Bit per avisar que hi ha un error
u8 Cod_err=0;                   //Codi de l'error
u16 Bateria=0;                  //Nivell de la bateria llegit
u16 BateriaS=0;                 //Sumatori dels nivells de bateria
u16 BateriaM=0;                 //Mitjana del nivell de bateria
u8 n=0;                         //Comptador per la mitja de la bateria
u8 rError=0;                    //Reset dels errors
u8 sock=0;                      //Socket de la connexió
int sockC=0;

//DECLARACIÓ DE LES FUNCIONS
void clear_screen (int screen);
void clear_text (void);
u32 edit_addr (void);
u32 edit_num (void);
void composa (void);
void descomposa (void);

//FUNCIÓ PER COMPOSAR LES DADES
void composa (void)
{
      buffsend[0] = (Ctl_gir<<6) | (Ctl_gir<<5) | (Ctl_obs<<2) |
      (Ctl_vel); //Config.
      buffsend[1] = (0x10) | (rError<<2) | (rError<<1) | (0x01);
      //Config.
      buffsend[6] = 0; //Con_X
      buffsend[7] = 0; //Con_Y
      buffsend[8] = 0; //Con_x
      buffsend[9] = 0; //Con_y
      buffsend[10]= 0; //Con_T
      buffsend[11]= 0; //Pos_X
```

```
	buffsend[12]= 0; //Pos_Y
	buffsend[13]= 0; //Pos_x
	buffsend[14]= 0; //Pos_y
	buffsend[15]= 0; //Pos_T
	buffsend[16]= 0; //Ports auxiliars
	buffsend[17]= 0; //Ports auxiliars
}

//FUNCIÓ PER INTERPRETAR LES DADES REBUDES
void descomposa (void)
{
	Bateria = buffrcv[0];
	if (((buffrcv[1]&0x80)>0) | ((buffrcv[1]&0x40)>0) |
	((buffrcv[1]&0x20)>0) | ((buffrcv[2]&0x80)>0)
	| ((buffrcv[2]&0x40)>0) | ((buffrcv[2]&0x20)>0)) Error=1;
	else Error=0;
	if ((buffrcv[1]&0x1F)>0) Cod_err=buffrcv[1]&0x1F;
	if ((buffrcv[2]&0x1F)>0) Cod_err=buffrcv[2]&0x1F;
	if (((buffrcv[1]&0x1F)==0) && ((buffrcv[2]&0x1F)==0)) Cod_err=0;
}

//FUNCIÓ PER BORRAR EL TEXT D'UNA PANTALLA
void clear_screen (int screen)
{
	for (j = 0; j < 23; j++)
	PA_OutputText(screen, 0, j, "                                ");
}

//FUNCIÓ PER BORRAR LA MATRIU DE TEXT
void clear_text (void)
{
	for (j = 0; j < 200; j++)
	text[j]=160;
}

//FUNCIÓ PER EDITAR UNA ADREÇA (tipus ip)
u32 edit_addr (void)
{
	clear_text();
	nletter=0;
	u32 addr;
	PA_ResetSpriteSys();
	clear_screen(0);
	clear_text();
	PA_EasyBgLoad (0, 3, fondo_3);
	// Exit button
	PA_CreateSprite (0, 0, (void*)sortir_Sprite, OBJ_SIZE_16X16, 1, 0,
	232, 8);
	//OK button
	PA_CreateSprite (0, 1, (void*)OK_Sprite, OBJ_SIZE_16X16, 1, 0, 232,
	32);
	PA_InitAllSpriteDraw();
	PA_InitKeyboard(2);		// Carrega el teclat virtual
	PA_KeyboardIn(20, 95);		// Posiciona el teclat
	PA_SetKeyboardColor(0, 1);	// Color del teclat (blau i vermell)
	while (1)
	{
		letter = PA_CheckKeyboard();
		if (letter>=48 && letter<=57 && nletter<15)
		{
			text[nletter] = letter;
```

```
                if (nletter==2)
                {
                      if (((text[0]-48)*100+(text[1]-48)*10+(text[2]-
                      48))>255)
                      {
                            text[0]=50;
                            text[1]=50;
                            text[2]=53;
                      }
                nletter++;
                text[nletter]=46;
                }
                if (nletter==6)
                {
                      if (((text[4]-48)*100+(text[5]-48)*10+(text[6]-
                      48))>255)
                      {
                            text[4]=50;
                            text[5]=50;
                            text[6]=53;
                      }
                nletter++;
                text[nletter]=46;
          }
          if (nletter==10)
          {
                if (((text[8]-48)*100+(text[9]-48)*10+(text[10]-
                48))>255)
                {
                      text[8]=50;
                      text[9]=50;
                      text[10]=53;
                }
                nletter++;
                text[nletter]=46;
          }
          if (nletter==14)
          {
                if (((text[12]-48)*100+(text[13]-48)*10+(text[14]-
                48))>255)
                {
                      text[12]=50;
                      text[13]=50;
                      text[14]=53;
                }
          }
          nletter++;
    }

    else if ((letter == PA_BACKSPACE)&&nletter)
    {
          nletter--;
          text[nletter] = ' ';
    }
    else if ((letter==46)&&(nletter==3 || nletter==7 || nletter==11))
    {
          text[nletter] = letter;
          nletter++;
    }
    if (PA_SpriteTouchedPix(0))
    {
```

```
            PA_KeyboardOut();
            return 0;
      }
      if (PA_SpriteTouchedPix(1))
      {
            PA_KeyboardOut();
            if (nletter==15)
            {
            addr=((text[0]-48)*100 + (text[1]-48)*10 + (text[2]-48));
            addr=addr|(((text[4]-48)*100 + (text[5]-48)*10 + (text[6]-
            48))*0x100);
            addr=addr|(((text[8]-48)*100 + (text[9]-48)*10 + (text[10]-
            48))*0x10000);
            addr=addr|(((text[12]-48)*100 + (text[13]-48)*10 + (text[14]-
            48))*0x1000000);
            return (u32)(addr);
            }
            else
            {
            PA_Print(1, "Invalid value!!\nIgnoring changes!\n");
            return 0;
            }
      }
      PA_OutputText(0, 7, 5, text);
      PA_WaitForVBL();
      } //End while
}

//FUNCIÓ PER EDITAR UN ENTER
u32 edit_num (void)
{
      u32 num=0;
      clear_text();
      nletter=0;
      clear_text();
      nletter=0;
      PA_ResetSpriteSys();
      clear_screen(0);
      clear_text();
      PA_EasyBgLoad (0, 3, fondo_3);
      // Exit button
      PA_CreateSprite (0, 0, (void*)sortir_Sprite, OBJ_SIZE_16X16, 1, 0,
      232, 8);
      //OK button
      PA_CreateSprite (0, 1, (void*)OK_Sprite, OBJ_SIZE_16X16, 1, 0, 232,
      32);
      PA_InitAllSpriteDraw();
      PA_InitKeyboard(2); // Carrega el teclat virtual
      PA_KeyboardIn(20, 95); // Posiciona el teclat
      PA_SetKeyboardColor(0, 1); // Color del teclat (blau i vermell)
      while(1)
      {
            letter = PA_CheckKeyboard();
            if (letter>=48 && letter<=57 && nletter<8)
            {
                  text[nletter] = letter;
                  nletter++;
            }
            else if ((letter == PA_BACKSPACE)&&nletter)
            {
                  nletter--;
```

```
                text[nletter] = ' ';
            }
            if (PA_SpriteTouchedPix(0))
            {
                PA_KeyboardOut();
                return 0;
            }
            if (PA_SpriteTouchedPix(1))
            {
                PA_KeyboardOut();
                int i=1;
                for (j = nletter; j > 0; j--)
                {
                    num=num+((text[j-1]-48)*i);
                    i=i*10;
                }
                return num;
            }
      PA_OutputText(0, 7, 5, text);
      PA_WaitForVBL();
      }
}

//FUNCIÓ PRINCIPAL
int main(int argc, char ** argv)
{
int sock;
int k=0;
dsIP=0xC801A8C0;        //192.168.1.200
dsGW=0x0;               //0.0.0.0
dsSM=0x00FFFFFF;        //255.255.255.0
rbIP=0x6701A8C0;        //192.168.1.103
rbPORT=15000;
dsDNS1=0x0;
dsDNS2=0x0;

while(k<21)             //Inicialitza buffrcv
{
      buffrcv[k]=0;
      k++;
}
k=0;
while(k<16)             //Inicialitza buffsend
{
      buffsend[k]=0;
      k++;
}

PA_Init();              // Initializes PA_Lib
PA_InitVBL();           // Initializes a standard VBL
//Inici tel texte de la pantalla 0 en la capa 0
PA_InitText (0, 0);
//Inici del texte de la pantalla 1 en la capa 0
PA_InitText (1, 0);
//Configuraxció del color del text (pantalla, vermell, verd, blau)
PA_SetTextCol (0, 0, 0, 31);
PA_SetTextCol (1, 31, 0, 0);
//Carreguem la paleta de sprites (pantalla, nº paleta, nom paleta)
PA_LoadSpritePal (0, 0, (void*)sprite0_Pal);
PA_Print(1, "Starting...\n");
estat=1;
```

```
while (1)
{
switch (estat)
{
	case 1: //Inicialitzacions
	{
		PA_Print(1, "Initialize...\n");
		PA_ResetSpriteSys();
		clear_screen(0);
		PA_InitWifi();
		if (Wifi_CheckInit())
		{
			PA_Print(1, "arm7 ready for wifi\n");
		}
		else
		{
			PA_Print(1, "arm7 failed\n");
		}
		Wifi_ScanMode();
		PA_Print(1,"Wifi set to scan mode\n");
		estat=2;
		break;
	}

	case 2: //construcció de la pantalla de configuració de la NDS
	{
		clear_screen(0);
		PA_ResetSpriteSys();
		//Carreguem el fondo de pantalla (pantalla, capa, nom del
		fondo)
		PA_EasyBgLoad (0, 3, fondo_2);
		//Edit IP button (pantalla, nº sprite, nom sprite, tamany
		sprite, paleta color, pos X, pos Y)
		PA_CreateSprite (0, 0, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
		0, 216, 44);
		//Edit GW button
		PA_CreateSprite (0, 1, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
		0, 216, 68);
		//Edit Mask button
		PA_CreateSprite (0, 2, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
		0, 216, 92);
		//Edit DNS1 button
		PA_CreateSprite (0, 3, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
		0, 216, 116);
		//Edit DNS2 button
		PA_CreateSprite (0, 4, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
		0, 216, 140);
		//OK button
		PA_CreateSprite (0, 5, (void*)OK_Sprite, OBJ_SIZE_16X16, 1,
		0, 232, 168);
		PA_Print(1, "configuring TCP/IP...\n");
		PA_InitAllSpriteDraw(); //Inicialitza el sistema per detectar
		quan es toca
		un sprite
		estat=3;
		break;
	}

	case 3: //Edició de la configuració de la NDS
	{
```

```
        PA_OutputText(0,10,6,"%d.%d.%d.%d",(dsIP&0xFF),
        (dsIP&0xFF00)>>8, (dsIP&
        0xFF0000)>>16, (dsIP&0xFF000000)>>24);
        PA_OutputText(0,10,9,"%d.%d.%d.%d",(dsGW&0xFF),
        (dsGW&0xFF00)>>8, (dsGW&
        0xFF0000)>>16, (dsGW&0xFF000000)>>24);
        PA_OutputText(0,10,12,"%d.%d.%d.%d",(dsSM&0xFF),
        (dsSM&0xFF00)>>8, (dsSM&
        0xFF0000)>>16, (dsSM&0xFF000000)>>24);
        PA_OutputText(0,10,15,"%d.%d.%d.%d",(dsDNS1&0xFF),
        (dsDNS1&0xFF00)>>8, (dsDNS1
        &0xFF0000)>>16, (dsDNS1&0xFF000000)>>24);
        PA_OutputText(0,10,18,"%d.%d.%d.%d",(dsDNS2&0xFF),
        (dsDNS2&0xFF00)>>8, (dsDNS2
        &0xFF0000)>>16, (dsDNS2&0xFF000000)>>24);
        if (PA_SpriteTouchedPix(0)) {estat=4; PA_Print(1, "Edit IP
        address\n");}
        if (PA_SpriteTouchedPix(1)) {estat=5; PA_Print(1, "Edit
        gateway\n");}
        if (PA_SpriteTouchedPix(2)) {estat=6; PA_Print(1, "Edit
        subnet mask\n");}
        if (PA_SpriteTouchedPix(3)) {estat=7; PA_Print(1, "Edit DNS
        1\n");}
        if (PA_SpriteTouchedPix(4)) {estat=8; PA_Print(1, "Edit DNS
        2\n");}
        if (PA_SpriteTouchedPix(5)) estat=10;
        if (Pad.Newpress.A) estat=10;
        break;
}



case 4: //Edició de la IP de la NDS
{
        naddr=edit_addr();
        if (naddr>0) dsIP=naddr;
        estat=2;
        break;
}
case 5: //Edició de la porta d'enllaç de la NDS
{
        naddr=edit_addr();
        if (naddr>0) dsGW=naddr;
        estat=2;
        break;
}
case 6: //Edició de la mascara de la NDS
{
        naddr=edit_addr();
        if (naddr>0) dsSM=naddr;
        estat=2;
        break;
}
case 7: //Edició de la DNS 1
{
        naddr=edit_addr();
        if (naddr>0) dsDNS1=naddr;
        estat=2;
        break;
}
case 8: //Edició de la DNS 2
```

```
{
	naddr=edit_addr();
	if (naddr>0) dsDNS2=naddr;
	estat=2;
	break;
}
case 10: //Construcció de la pantalla de selecció de AP
{
	clear_screen(0);
	//Carreguem el fondo de pantalla
	PA_EasyBgLoad (0, 3, fondo_1);
	PA_ResetSpriteSys();
	// Exit button
	PA_CreateSprite (0, 0, (void*)sortir_Sprite, OBJ_SIZE_16X16,
	1, 0, 232, 8);
	//OK button
	PA_CreateSprite (0, 1, (void*)OK_Sprite, OBJ_SIZE_16X16, 1,
	0, 232, 32);
	PA_InitAllSpriteDraw();
	Wifi_SetIP(dsIP, dsGW, dsSM, dsDNS1, dsDNS2);
	PA_Print(1, "Aply TCP/IP configuration\n");
	estat=11;
	PA_Print(1,"Scanning for wifi AP's\n");
	PA_Print(1,"Select the robot's AP");
	break;
}
case 11: //Selecció de la xarxa del robot
{
	clear_screen(0);
	numAP=Wifi_GetNumAP(); //Mirem quantes xarxes rebem
	for (j = 0; j < numAP; j++)
	{
		Wifi_GetAPData(j, &AP); //Obtenim les dades de cada
		xarxa
		if (j==selection)
		{
			conectAP=AP;
			PA_OutputText(0, 2, (5+j), "--> AP %d: %s
			CH:%d",j, AP.ssid,AP.channel);
		}
		else
		{
			PA_OutputText(0, 2, (5+j), " AP %d: %s CH:%d",j,
			AP.ssid,AP.channel);
		}
	}
	if(Pad.Newpress.Down)
	{
		selection=selection+1;
		if (selection>=numAP) {selection=0;}
	}
	if(Pad.Newpress.Up)
	{
		selection=selection-1;
		if (selection<0) {selection=numAP-1;}
	}
	if (Pad.Newpress.A)
	{
		estat=12;
	}
	if (PA_SpriteTouchedPix(0)) estat=2;
```

```
        if (PA_SpriteTouchedPix(1)) estat=12;
        break;
    }

    case 12: //Construcció de la pantalla de configuració de la
    connexio
    {
        clear_screen(0);
        PA_ResetSpriteSys();
        //Carreguem el fondo
        PA_EasyBgLoad (0, 3, fondo_4);
        //Exit button
        PA_CreateSprite (0, 0, (void*)sortir_Sprite, OBJ_SIZE_16X16,
        1, 0, 232, 8);
        //OK button
        PA_CreateSprite (0, 1, (void*)OK_Sprite, OBJ_SIZE_16X16, 1,
        0, 232, 168);
        //Edit IP button
        PA_CreateSprite (0, 2, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
        0, 216, 44);
        //Edit port
        PA_CreateSprite (0, 3, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
        0, 216, 68);
        PA_InitAllSpriteDraw();
        estat=13;
        break;
    }

    case 13: //Edició de la configuració de la connexió
    {
        PA_OutputText(0,10,6,"%d.%d.%d.%d",(rbIP&0xFF),
        (rbIP&0xFF00)>>8, (rbIP&
        0xFF0000)>>16, (rbIP&0xFF000000)>>24);
        PA_OutputText(0,10,9,"%d",rbPORT);
        if (PA_SpriteTouchedPix(0)) {estat=2;}
        if (PA_SpriteTouchedPix(1)) {estat=16;}
        if (Pad.Newpress.A) estat=16;
        if (PA_SpriteTouchedPix(2)) {estat=14; PA_Print(1, "Edit IP
        address\n");}
        if (PA_SpriteTouchedPix(3)) {estat=15; PA_Print(1, "Edit port
        number\n");}
        break;
    }
    case 14: //Edició de la IP del robot
    {
        naddr=edit_addr();
        if (naddr>0) rbIP=naddr;
        estat=12;
        break;
    }

    case 15: //Edició del port del robot
    {
        rbPORT=edit_num();
        estat=12;
        break;
    }

    case 16: //Construcció de la pantalla de control
    {
        PA_ResetSpriteSys();
```

```
        clear_screen(0);
        PA_EasyBgLoad (0, 3, fondo_5);
        // Exit button
        PA_CreateSprite (0, 0, (void*)sortir_Sprite, OBJ_SIZE_16X16,
        1, 0, 232, 8);
        //Ctl_vel button
        PA_CreateSprite (0, 1, (void*)on_off_Sprite, OBJ_SIZE_32X16,
        1, 0, 116, 116);
        //Ctl_PWM button
        PA_CreateSprite (0, 2, (void*)on_off_Sprite, OBJ_SIZE_32X16,
        1, 0, 116, 140);
        //Ctl_obs button
        PA_CreateSprite (0, 3, (void*)on_off_Sprite, OBJ_SIZE_32X16,
        1, 0, 116, 164);
        //Edit VL button
        PA_CreateSprite (0, 4, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
        0, 134, 36);
        //Edit VG button
        PA_CreateSprite (0, 5, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
        0, 134, 60);
        //Edit VGL button
        PA_CreateSprite (0, 6, (void*)edit_Sprite, OBJ_SIZE_32X16, 1,
        0, 134, 84);
        //Reset button
        PA_CreateSprite (0, 7, (void*)reset_Sprite, OBJ_SIZE_32X16,
        1, 0, 200, 56);
        //Error led
        PA_CreateSprite (0, 8, (void*)led_Sprite, OBJ_SIZE_32X16, 1,
        0, 200, 96);
        if (Ctl_vel==0) PA_SetSpriteAnim(0, 1, 1);
        else PA_SetSpriteAnim(0, 1, 0);
        if (Ctl_obs==0) PA_SetSpriteAnim(0, 3, 1);
        else PA_SetSpriteAnim(0, 3, 0);
        if (Ctl_gir==0) PA_SetSpriteAnim(0, 2, 1);
        else PA_SetSpriteAnim(0, 2, 0);
        PA_InitAllSpriteDraw();
        if (sock==0)
        {
              PA_Print(1, "Conecting to %s...\n",conectAP.ssid);
              //Wifi_ConnectAP(Wifi_AccessPoint * apdata, int
              wepmode, int wepkeyid, unsigned char * wepkey);
              Wifi_ConnectAP(&conectAP, 0, 0, 0);
              PA_Print(1, "Init Socket... wait!\n");
              PA_InitSocket(&sock,"192.168.1.103",rbPORT,PA_NORMAL_TC
              P);
              PA_Print(1, "Socket: %d\n",sock);
        }
        estat=17;
        break;
}
case 17: //Edicio dels controls
{
        clear_screen(0);
        if (PA_SpriteTouchedPix(0)) {estat=2;}
        if (PA_SpriteTouchedPix(1) && Ctl_vel==1)
        {PA_SetSpriteAnim(0, 1, 1); Ctl_vel=
        0; break;}
        if (PA_SpriteTouchedPix(1) && Ctl_vel==0)
        {PA_SetSpriteAnim(0, 1, 0); Ctl_vel=
        1; break;}
```

```
if (PA_SpriteTouchedPix(2) && Ctl_gir==1)
{PA_SetSpriteAnim(0, 2, 1); Ctl_gir=
0; break;}
if (PA_SpriteTouchedPix(2) && Ctl_gir==0)
{PA_SetSpriteAnim(0, 2, 0); Ctl_gir=
1; break;}
if (PA_SpriteTouchedPix(3) && Ctl_obs==1)
{PA_SetSpriteAnim(0, 3, 1); Ctl_obs=
0; break;}
if (PA_SpriteTouchedPix(3) && Ctl_obs==0)
{PA_SetSpriteAnim(0, 3, 0); Ctl_obs=
1; break;}
if (PA_SpriteTouchedPix(7)) {rError=1;}
else rError=0;
if (Error==1) PA_SetSpriteAnim(0, 8, 1);
else PA_SetSpriteAnim(0, 8, 0);
if (PA_SpriteTouchedPix(4)) {estat=18; PA_Print(1, "Edit
speed VL\n");}
if (PA_SpriteTouchedPix(5)) {estat=19; PA_Print(1, "Edit
speed VG\n");}
if (PA_SpriteTouchedPix(6)) {estat=20; PA_Print(1, "Edit
speed VGL\n");}
if (Ctl_vel==1 && VL>127) VL=127;
if (Ctl_vel==0 && VL>100) VL=100;
if (Ctl_vel==1 && VG>127) VG=127;
if (Ctl_vel==0 && VG>100) VG=100;
if (Ctl_vel==1 && VGL>127) VGL=127;
if (Ctl_vel==0 && VGL>100) VGL=100;
if (Pad.Held.Up && !Pad.Held.Left && !Pad.Held.Right)
{
      buffsend[2]=buffsend[4]=128+VL;
      buffsend[3]=buffsend[5]=128+VL;
}
else if (Pad.Held.Down && !Pad.Held.Left && !Pad.Held.Right)
{
      buffsend[2]=buffsend[4]=VL;
      buffsend[3]=buffsend[5]=VL;
}
else if (Pad.Held.Left && !Pad.Held.Up && !Pad.Held.Down)
{
      buffsend[2]=buffsend[4]=VGL;
      buffsend[3]=buffsend[5]=128+VGL;
}
else if (Pad.Held.Right && !Pad.Held.Up && !Pad.Held.Down)
{
      buffsend[2]=buffsend[4]=128+VGL;
      buffsend[3]=buffsend[5]=VGL;
}
else if (Pad.Held.Right && Pad.Held.Up)
{
      buffsend[2]=buffsend[4]=128+VG;
      buffsend[3]=buffsend[5]=0;
}
else if (Pad.Held.Right && Pad.Held.Down)
{
      buffsend[2]=buffsend[4]=VG;
      buffsend[3]=buffsend[5]=0;
}
else if (Pad.Held.Left && Pad.Held.Up)
{
      buffsend[2]=buffsend[4]=0;
```

```
                buffsend[3]=buffsend[5]=128+VG;
            }
            else if (Pad.Held.Left && Pad.Held.Down)
            {
                buffsend[2]=buffsend[4]=0;
                buffsend[3]=buffsend[5]=VG;
            }
            else if (!Pad.Held.Right && !Pad.Held.Up && !Pad.Held.Down &&
            !Pad.Held.Left)
            {
                buffsend[2]=buffsend[4]=0;
                buffsend[3]=buffsend[5]=0;
            }
            n++;
            BateriaS=BateriaS+Bateria;
            if (n==9)
            {
                BateriaM=BateriaS/10;
                BateriaS=0;
                n=0;
            }
            PA_OutputText(0, 24, 20, "%d.%d", BateriaM/10, BateriaM%10);
            PA_OutputText(0, 26, 10, "%d", Cod_err);
            PA_OutputText(0, 11, 5, "%d",VL);
            PA_OutputText(0, 11, 8, "%d",VG);
            PA_OutputText(0, 11, 11, "%d",VGL);
            composa();
            send(sock,buffsend,18,0);
            recv(sock,buffrcv,21,0);
            descomposa();
            break;
        }

        case 18: //Edit VL
        {
            VL=edit_num();
            estat=16;
            break;
        }

        case 19: //Edit VG
        {
            VG=edit_num();
            estat=16;
            break;
        }

        case 20: //Edit VGL
        {
            VGL=edit_num();
            estat=16;
            break;
        }

        }//End switch
        PA_WaitForVBL();
}//End while
PA_Print(1, "Exiting aplication ...");
return 0;
}//End main
```

A.3. Llibreries

En aquest apartat es detalla el codi font de totes les llibreries fetes servir en els programes descrits anteriorment

A.3.1. bot.h

```
////////////////////////////////////////////////////////////////
// NOM FITXER:    bot.h                                       //
// AUTOR:         Rafael Hesse                                //
// DATA:          04/05/2008                                  //
// PLATAFORMA:    dsPIC Wifibot                               //
// LLENGUATGE:    C, compilador C30 Microchip                 //
// DESCRIPCIÓ:    Include per al nou programa dels dsPIC's del   //
//                wifibot. Inclou declaracions comunes per ambdós//
// dsPIC's (davant i darrera)                                 //
////////////////////////////////////////////////////////////////

#include <p30f2010.h>
#include <dsp.h>
#include <I2C.h>
#include <pwm.h>
#include <adc10.h>

#define DI 127                //Diàmetre de les rodes en mm
#define K ((DI*PI)/6)         //constant de conversio entre metres i tics
#define B (K/0.33)            //Amplada del robot expresada en tics
#define graus (360/(2*PI))    //constant per passar de radiants a graus
#define rad ((2*PI)/360)      //constant per passar de graus a radiants
#define VEL ((PI*DI)/300.0)
#define dist_obs 10           //Distància per al control d'obstacles

void init_AD (void);
void init_PWM (void);
void Delay_ms (unsigned int);
void Delay_i (unsigned int);
void Delay_s (unsigned int);
void init_T_QEI (void);
```

A.3.2. wifibot.h

```
////////////////////////////////////////////////////////////////////////
// Nom de la llibreria:      wifibot.h                                //
// Plataforma:               Nylon Linux del Wifibot                  //
// Autor:                    Rafael Hesse (modificació)               //
// Data:                     12/12/2007 (modificació)                 //
// Versió:                   1.1                                      //
// Compilador:               GCC cross compiler mipsel-linux-gcc      //
// Comentaris:               Definicions especifiques per al programa //
//                           simple_server_ADC                        //
////////////////////////////////////////////////////////////////////////

#include <pthread.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include "i2c-dev.h"
int simu=0;
static char filename3[20];
static int i2cbus=0,file;
static unsigned char bufset[10];
static unsigned char bufget[3];
static unsigned char bufad[1];
#define NB_THREADS 2
void * fn_thread_dog (void * numero);
void * fn_thread_com (void * numero);
static void initI2c();
static void setI2cL();
static void getI2cL();
static void setI2cR();
static void getI2cR();
static void getAD();
static void stop();
void gestionnaire (int numero);
static int compteur = 0;
pthread_t thread [NB_THREADS];
int i;
int ret;
int dog=0;
pthread_mutex_t mutex_dog = PTHREAD_MUTEX_INITIALIZER;
#define MAXPENDING 5    /* Maximum outstanding connection requests */
static int servSock;    /* Socket descriptor for server */
static int clntSock;    /* Socket descriptor for client */
static struct sockaddr_in echoServAddr;         /* Local address */
static struct sockaddr_in echoClntAddr;         /* Client address */
static unsigned short echoServPort=15000;       /* Server port */
static unsigned int clntLen;                    /* Length of client
                                        address data structure */
```

```
static int recvMsgSize=0;
static unsigned char buffso_rcv[32];
static unsigned char buffso_send[7];
int first=1;
static int connected=0;
```

A.3.3. wifibot2.h

```
////////////////////////////////////////////////////////////////////
// Nom de la llibreria:      wifibot2.h                           //
// Plataforma:               Nylon Linux del Wifibot              //
// Autor:                    Rafael Hesse (modificació)           //
// Data:                     12/12/2007 (modificació)             //
// Versió:                   1.1                                  //
// Compilador:               GCC cross compiler mipsel-linux-gcc  //
// Comentaris:               Definicions per als programes del robot  //
////////////////////////////////////////////////////////////////////
/
#include <pthread.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include "i2c-dev.h"
static char filename3[20];
static int i2cbus=0,file;
```

ANNEX B. MANUALS

En aquest apartat es detallen diversos manuals creats per tal de facilitar el treball futur amb el robot.

B.1. Accés remot al robot

En aquest annex es descriuen les aplicacions que recomana el fabricant per accedir remotament al sistema operatiu del robot des d'un PC amb sistema operatiu Windows.

Tot i que aquests mètodes compleixen amb la funció son mètodes complicats i poc pràctics. Així l'últim apartat descriu com obtenir un compilador adient per un sistema operatiu Linux (distribució Gentoo) evitant la necessitat de fer servir aplicacions que ens permetin unir Linux i Windows.

B.1.1. Putty

Putty és una senzilla aplicació lliure que permet comunicar-nos amb altres sistemes via ssh, en el nostre cas el robot. Podem descarregar aquesta aplicació des de nombroses pàgines web o la podem trobar en el CD subministrat amb el robot pel fabricant.

Aquest programa no precisa d'instal·lació, simplement l'executarem. La Figura 101 mostra la finestra de configuració que apareix un cop iniciat el programa.

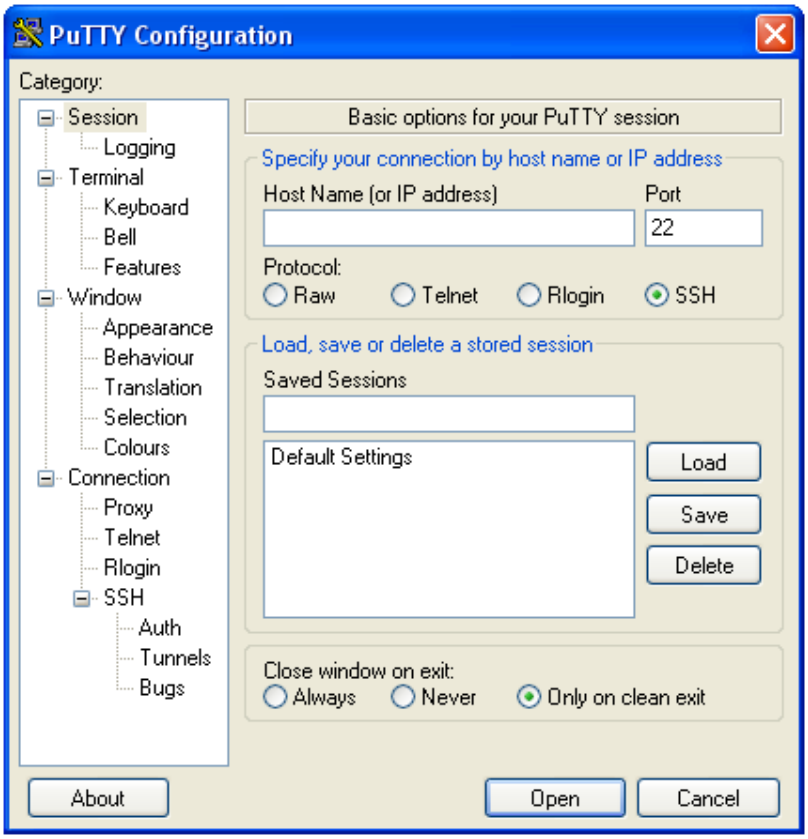

Figura 101. Configuració de Putty

En aquesta finestra entrarem els paràmetres de la connexió. En la casella “Host Name” entrarem la direcció IP del robot que dependrà de la forma en que ens connectem (LAN o Wifi). El port de comunicacions el deixarem tal i com ve configurat per defecte, port 22. El protocol de comunicació també el deixarem en l’opció per defecte, SSH.

A partir d’aquest punt tenim l’opció de guardar la configuració per futures connexions o simplement connectar-nos prement la tecla “Open”.

La finestra que ens apareixerà a continuació és similar a una consola de comandes Linux. Primerament ens demanarà el nom d’usuari per iniciar la sessió, per al robot entrarem “root”. Tot seguit demanarà la contrasenya per entrar i entrarem “wifibot”, donat que es un sistema Linux no es veuran caràcters de cap mena per pantalla quan escrivim la contrasenya. La Figura 102 mostra l’aspecte de la consola un cop hem iniciat sessió.

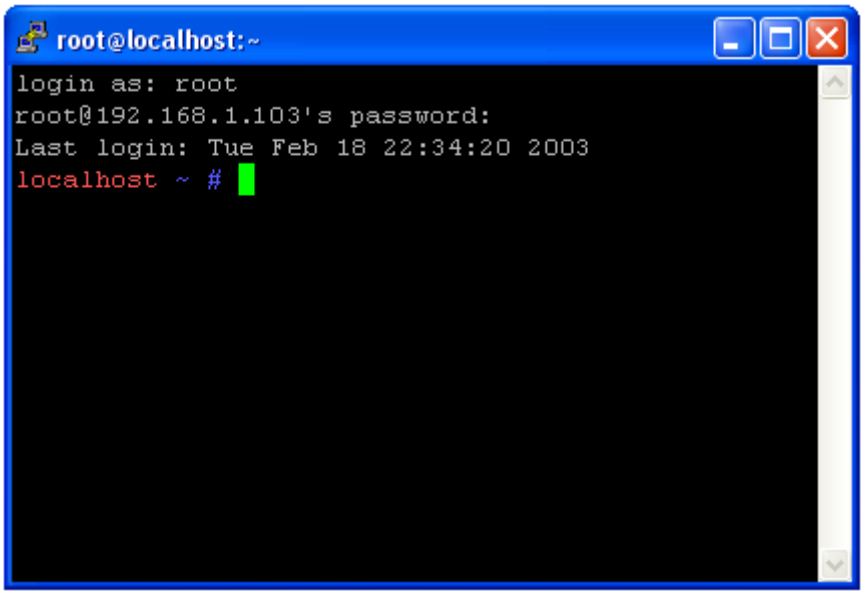

Figura 102. Consola Putty

Arribats en aquest punt podem fer servir les comandes típiques de Linux per moure'ns pel sistema d'arxius del robot. Si volem modificar algun arxiu hem de fer servir l'editor de textos "nano" que porta incorporat el robot. Així podem teclejar "nano /ruta_arxiu/nom_arxiu" per tal de editar-lo.

B.1.2. WinSCP

WinSCP és també una aplicació lliure que permet entrar remotament en sistemes d'arxius de forma gràfica. En aquest cas podrem visualitzar els arxius gràficament com si es tractés de l'explorer de Windows.

Al igual que Putty es tracta d'un executable que no precisa instal·lació. La Figura 103 mostra l'aspecte de la finestra d'inici de sessió que haurem de omplir amb les dades del robot.

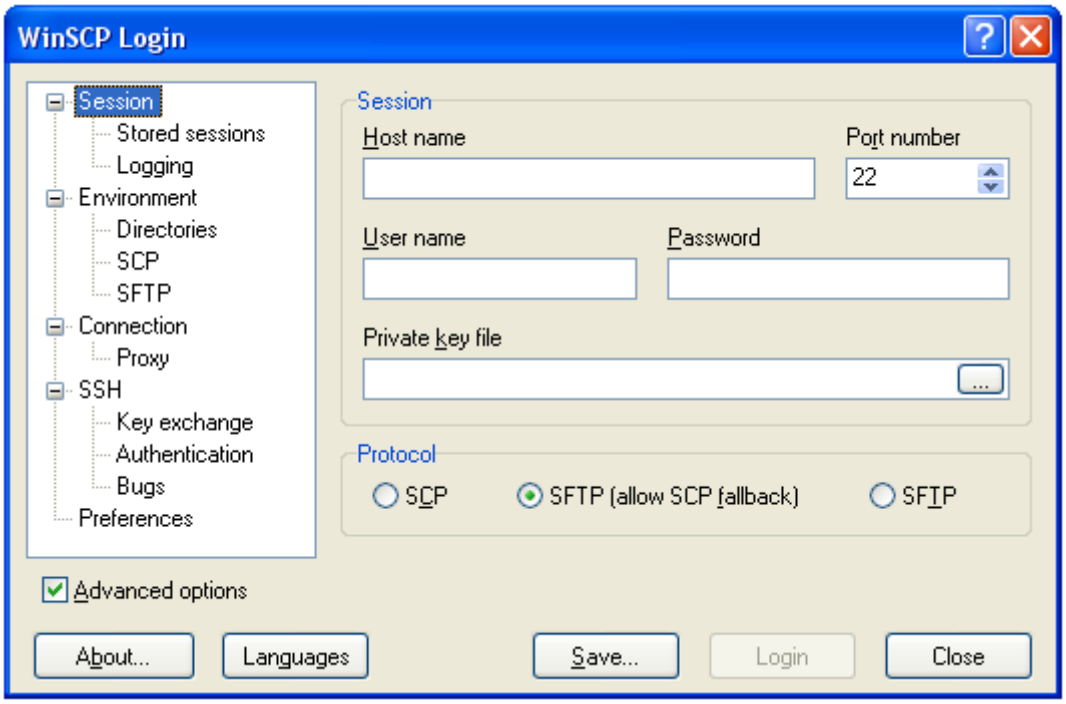

Figura 103. Inici de WinSCP

En la finestra “Host Name” entrarem la direcció IP del Robot la qual dependrà si ens connectem per LAN o Wifi. Les finestres “User Name” i “Pasword” les omplirem amb “root” i “wifibot” respectivament. A partir d’aquest punt podem guardar les configuracions per futures connexions o simplement iniciar la sessió prement “Login”.

La Figura 104 mostra la finestra que ens apareixerà l’aspecte de la qual pot variar segons la configuració, en aquest cas es tracta del mode de visualització símil a Norton Comander.

En el costat esquerra podem navegar pel sistema d’arxius del nostre PC i en el costat dret podem navegar pel sistema d’arxius del robot. D’aquesta manera podem copiar documents d’un costat a l’altre. Si copiem un document des del nostre PC cap al robot aquest per defecte tindrà els mínims permisos. Si volem poder executar o modificar aquest document posteriorment des del robot haurem de editar els permisos un cop copiat al robot.

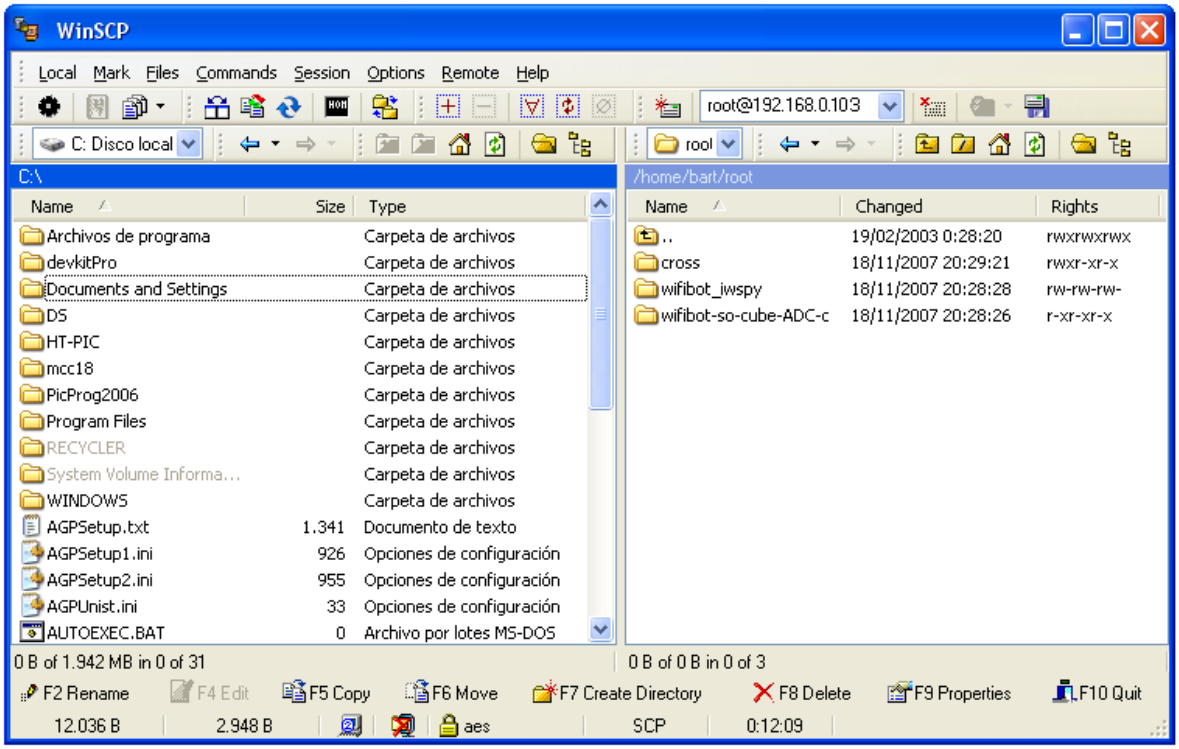

Figura 104. WinSCP

Donat que en aquest cas no disposem de consola per poder modificar els arxius ho hem de fer amb aplicacions ubicades en el PC des d'on ens connectem. En aquest sentit es important fer servir un editor compatible amb el format de text Linux i no els editors de Windows. Si volem modificar els arxius directament podem fer servir el propi WinSCP (botó dret sobre l'arxiu i "edit"). D'altra manera podem copiar els arxius al nostre PC i modificar-los posteriorment. En aquest cas es pot fer servir l'editor de textos que subministra el fabricant (WinVi32) en el seu CD.

B.2. Manual per Compilar codi per al robot.

Si volem crear aplicacions noves per executar en el cub precisem d'un compilador adient. Tot i que quasi per definició els sistemes Linux incorporen un compilador no es el cas del robot. Per tal de estalviar memòria aquest no disposa del seu propi compilador.

D'aquesta manera necessitem un compilador creuat, és a dir, un compilador instal·lat en un sistema que compili el codi per un altre sistema.

A continuació es descriuen dues possibles opcions per tal de compilar codi per al robot. Tot i que totes dues son funcionals la primera és poc pràctica però per contra molt senzilla.

B.2.1. CoLinux

CoLinux és una aplicació que ens permet córrer una imatge virtual d'un sistema operatiu Linux dins de Windows.

El fabricant del robot subministra en el CD una imatge d'un sistema Linux distribució Gentoo que incorpora el compilador creuat adient preinstal·lat. A continuació es descriuen els passos a seguir per tal de fer funcionar aquest sistema.

La ubicació de la instal·lació i els noms de les carpetes no han de ser obligatòriament com es descriu però si es modifiquen s'hauran de modificar també les configuracions.

B.2.1.1. CREAR L'ENTORN

Crearem una carpeta en l'arrel del nostre sistema Windows (en el nostre cas C:\CoLinux) i dins hi copiarem l'instal·lador de CoLinux i la carpeta "Image" que subministra el fabricant.

Dintre la carpeta "Image" trobarem la imatge virtual del sistema Linux i la imatge virtual de la memòria RAM per aquest sistema. Totes dues es troben comprimides. Per tal de descomprimir-les podem executar l'arxiu "install-wifibot-gentoo-cross-mips.bat" Aquest ens descomprimirà automàticament les dues imatges. Haurem d'assegurar-nos de tenir prou espai abans de descomprimir, la RAM virtual ocuparà 128 MB i el sistema operatiu ocuparà 779 MB.

Si per algun motiu aquestes unitats virtuals ens queden petites existeixen eines que permeten expandir-les fins a un màxim de 4 GB. Una d'aquestes eines s'anomena Toporesize i es programari lliure.

B.2.1.2. INSTAL·LACIÓ DE LA XARXA VIRTUAL

Per tal d'accedir a la imatge del sistema Linux des de Windows es fa a través d'un punt de xarxa virtual. Aquesta xarxa actuarà com a pont entre la imatge del sistema Linux i el nostre sistema Windows. L'instal·larem executant l'instal·lador que subministra el fabricant. La Figura 105 mostra les opcions que haurem de marcar a l'hora d'instal·lar. Només haurem de desmarcar l'opció de descarregar la imatge del sistema donat que farem servir la que ens dona el fabricant

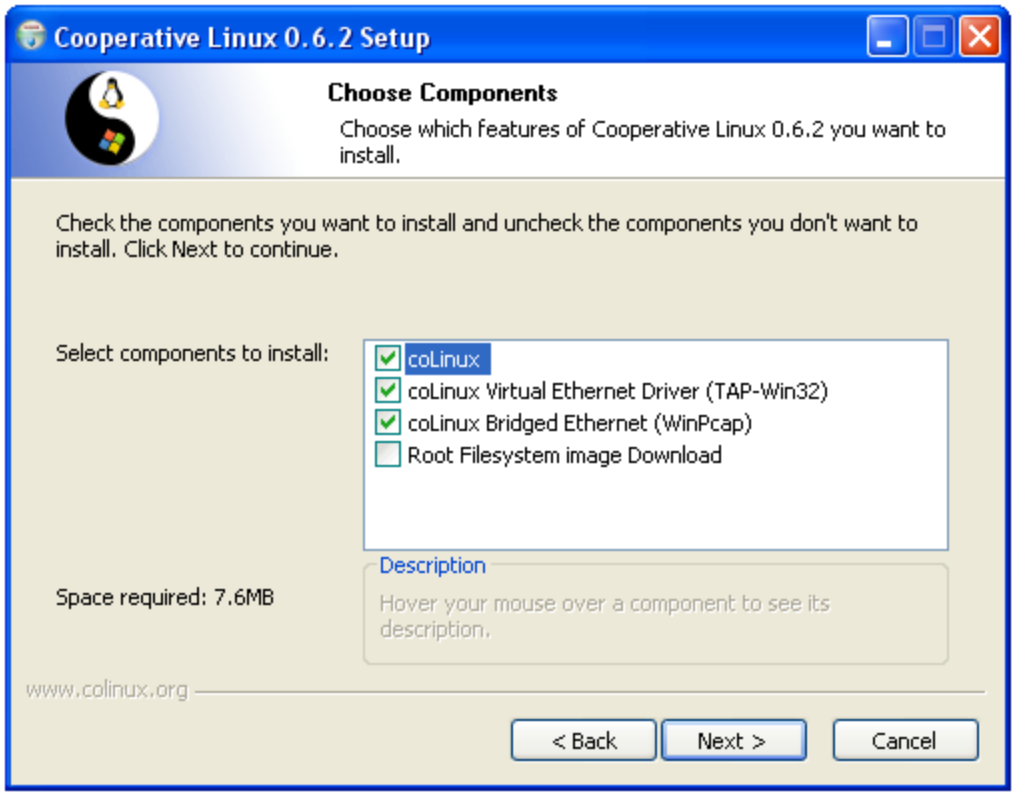

Figura 105. Opcions d'instal·lació

Seguidament ens demanarà on instal·lar el programa, entrarem la ruta de la carpeta "Image" que hem copiat anteriorment al nostre disc dur (C:\CoLinux\image). A partir d'aquí acceptarem tots els passos per finalitzar la instal·lació.

El programa ens crearà un punt de xarxa virtual que podrem veure en la carpeta "Conexiones de red". Aquesta xarxa tindrà un nom assignat per defecte segurament poc identificatiu i per tant el modificarem, en el nostre cas en direm "CoLinuxNet" (Figura 106)

Figura 106. Xarxa virtual

El nom que posem en aquesta xarxa és indiferent sempre i quant concordi amb l'arxiu de configuració.

B.2.1.3. CONFIGURACIÓ

Dins de la carpeta "Image" tenim un document de configuració (gentoo.xml) que haurem d'editar per tal d'introduir les rutes de la nostra instal·lació. La Figura 107 mostra com queda aquest arxiu per al nostre cas.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <colinux>
    <!-- This line needs to point to your root file system.
         For example change "root_fs" to the name of the Debian image.
         Inside coLinux it will be /dev/cobd0

         Block Device Aliasing: You can now handle most dual-boot issues
         by adding an alias="devname" to block_device. i.e. alias="hda",
         alias="hda1" You can do this for SCSI as well as IDE.  You need
         to be aware that if you add an alias, you need to change your
         bootparams root="devname" appropriately (you may need to use
         devfs naming in some situations).    -->
    <block_device index="0" path="\DosDevices\c:\CoLinux\image\Gentoo-colinux-stage3-x86-2004.3" enabled="true" />
    <!-- This line can specify a swap file if you wish, or an additional
         image file, it will /dev/cobd1. Additional block_devices can
         be specified in the same manner by increasing the index   -->
    <block_device index="1" path="\DosDevices\c:\CoLinux\swap_device" enabled="true" />
    <block_device index="3" path="\Device\Cdrom0" enabled="true" />
    <!-- bootparams allows you to pass kernel boot parameters   -->
    <bootparams>root=/dev/cobd0</bootparams>
    <!-- Initial RamDISK (initrd) support   -->
    <initrd path="initrd.gz" />
    <!-- image allows you to specify the kernel to boot   -->
    <image path="vmlinux" />
    <!-- this line allows you to specify the amount of memory available
         to coLinux   -->
    <memory size="64" />
    <!-- This allows you to modify networking parameters, see the README
         or website or wiki for more information   -->
    <network index="0" type="tap" name="CoLinuxNet" />
  </colinux>
```

Figura 107. Arxiu de configuració

B.2.1.4. EXECUCIÓ

Ara ja estan instal·lats i configurats tots els elements. Per iniciar aquest sistema virtual podem executar C:\CoLinux\image\wifibot-gentoo.bat. Se'ns obrirà una finestra com la de la Figura 108 en la qual se'ns demanarà usuari i contrasenya on haurem d'entrar "root" per tots dos casos, recordem que la contrasenya no veurem com es tecleja.

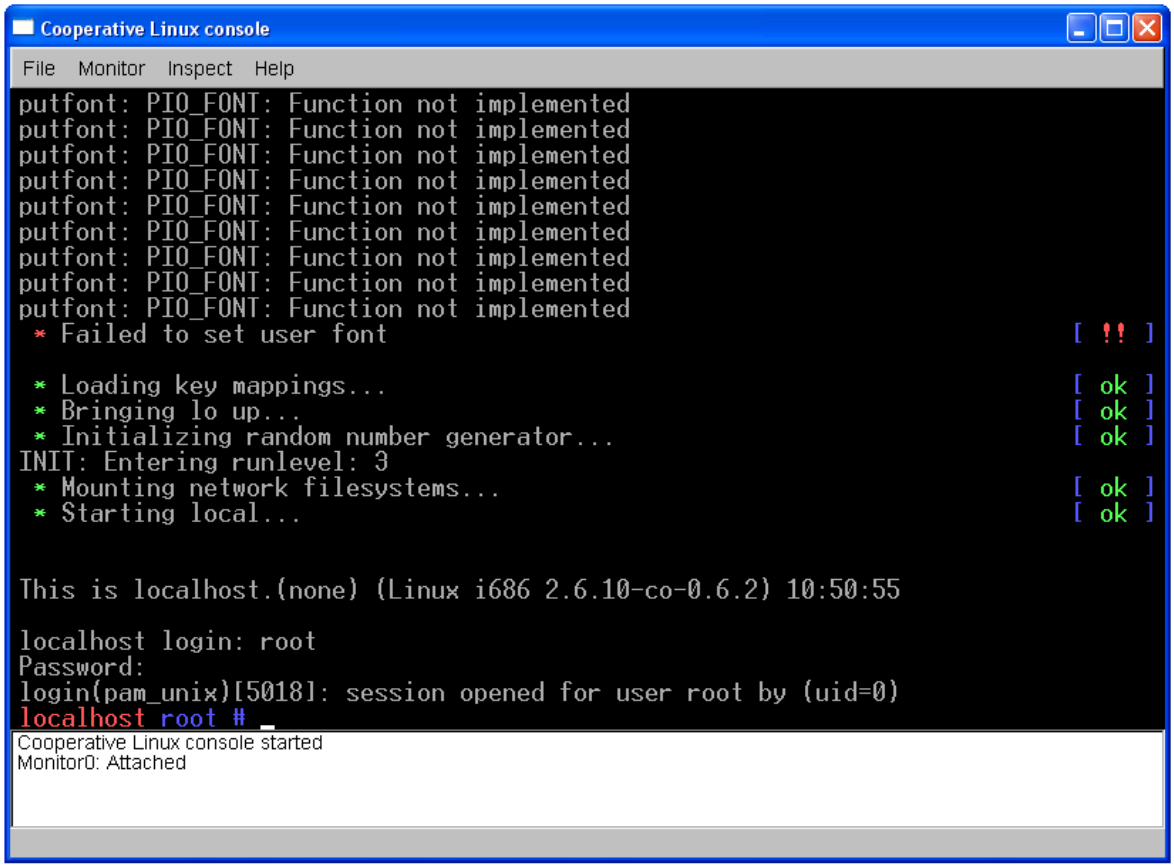

Figura 108. CoLinux

A partir d'aquest punt podem fer servir les comandes típiques de consola Linux per crear les nostres aplicacions. Donat que la imatge no té instal·lat cap motor gràfic haurem de fer servir l'editor de consola "nano" per editar el codi la qual cosa es poc pràctica per als casos en que el codi es llarg. Per tal de compilar el nostre codi per al robot haurem de fer referència al compilador creuat en l'arxiu "makefile" que gestiona la compilació. El compilador creuat el podrem cridar amb la comanda "mipsel-linux-gcc" (no cal afegir la ruta d'accés).

Un cop tinguem compilat el nostre codi el podem enviar al robot connectat al sistema Windows a través de la xarxa virtual amb la comanda "scp"

Per defecte aquesta imatge està configurada per funcionar amb un teclat amb distribució francesa de forma que l'haurem de reconfigurar. Per tal propòsit entrarem en l'arxiu de configuració amb la comanda mostrada a continuació:

```
$ nano /etc/rc.conf
```

En la Taula 23 es detalla on trobar determinats símbols que necessitem per poder escriure la comanda anterior

Caràcter	Tecla que l'ubica
a / A	q / Q
q / Q	a / A
. (punt)	Shift + , (coma)
M / m	Ñ / ñ

Taula 23. Conversió de caràcters.

La propera figura mostra el document que se'ns obrirà. Allà on diu "KEYMAP = "fr" " Haurem d'escriure "KEYMAP = "es" "

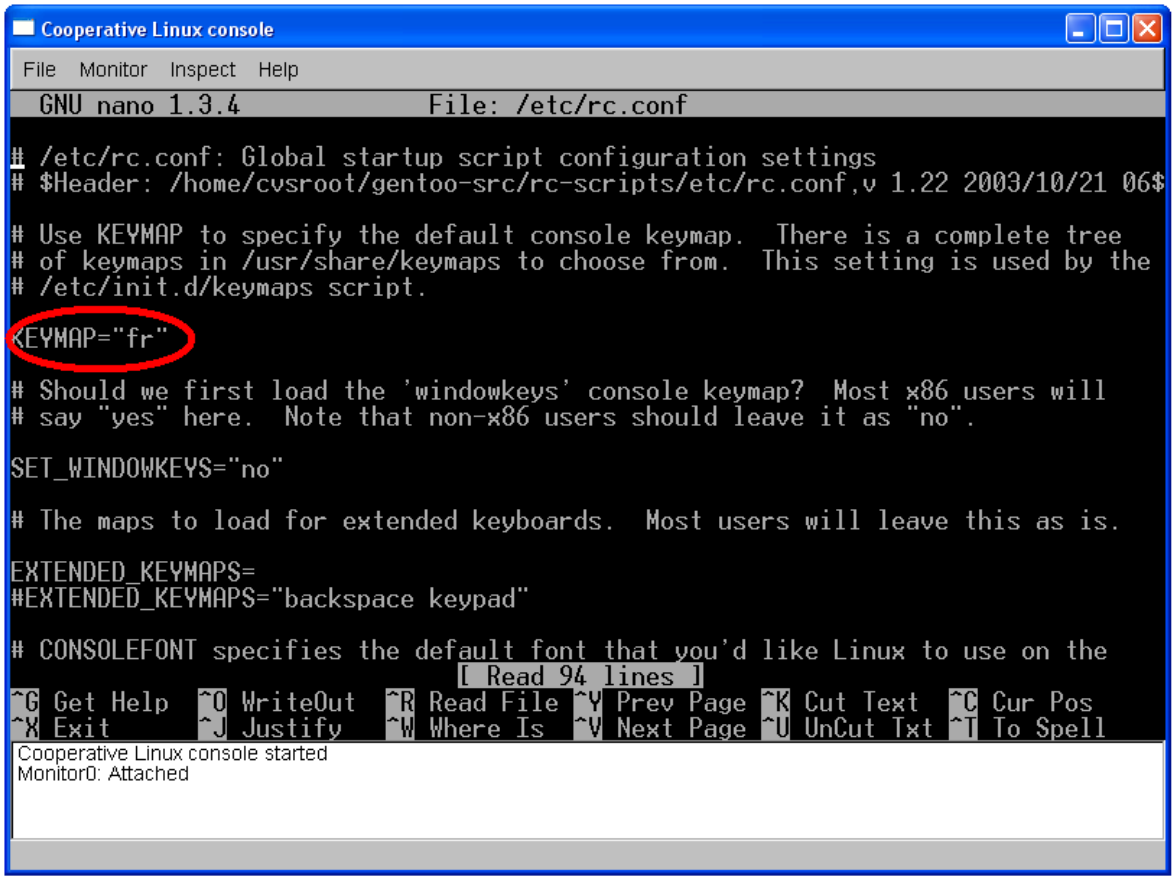

Figura 109. Canvi de configuració de teclat

Aquests canvis no tindran efecte abans no s'hagi reiniciat el sistema

B.2.2. Open embedded

Open embedded és un conjunt d'eines de programació lliures. Aquestes eines permeten construir un compilador creuat. Així ens permet crear un compilador en el nostre PC que compili codi per l'arquitectura del robot. A part per tal que l'executable resultant funcioni correctament en el robot es te en compte la distribució Linux i les versions del kernel.

Aquest manual descriu com instal·lar aquestes eines en un PC sobre un Linux distribució Gentoo, tot i això segurament serien instruccions similars per altres distribucions com ara Ubuntu. Sobre la distribució Suse Linux v.10 s'ha comprovat que no funciona correctament aquesta aplicació. Les proves s'han fet sobre una arquitectura AMD 64 i Intel PIV per altres arquitectures podrien haver petites diferències en les instruccions.

Aquesta via per a la programació del robot s'ha comprovat que a la llarga és la més recomanable donat que ens permet treballar amb un entorn gràfic per a la edició de textos.

B.2.2.1. REQUISITS

El la Taula 24 es detalla una llista de programes que prèviament han d'estar instal·lades en el sistema. Totes elles es poden instal·lar amb la comanda "emerge" nativa de la distribució Gentoo.

Nom aplicació	**Comanda per instal·lar**
Psyco JIT compiler	`$emerge psyco`
Fast compiler cache	`$emerge ccache`
GNU patch	`$emerge patch`
GNU make	`$emerge make`
GNU Sed stream editor	`$emerge sed`
Python	`$emerge dev-lang/python`
GNU m4 macro processor	`$emerge m4`
Bison parser generator	`$emerge bison`
CVS control tools	`$emerge cvs`
Openjade	`$emerge openjade`
Quilt patch manager	`$emerge quilt`
SGML tools	`$emerge sgmltools-lite`
Dockbook XML	`$emerge dockbook-xml-dtd`
XML and Dockbook conversor	`$emerge xmlto`
Dockbook DSSL	`$emerge dockbook-dssl-stylesheets`

Dockbook utilities	$emerge dockbook-sgml-utils
PCRE Headers	$emerge libpcre
Libraries for C++	$emerge boost
Subversion	$emerge subversion
Monotone (mínim versió 0.32)	$emerge monotone

Taula 24. Requeriment per Open Embedded

Seguidament es descriu pas per pas com generar un compilador per al robot Wifibot, tenint en compte les especificacions de la Taula 25 donades per les característiques del robot.

Distribució Linux	Nylon
Arquitectura	MIPS Little enddian
Dispositiu	Mesh Cube
Kernel	Versió 2.4.27
GCC	Versió 3.3.4

Taula 25. Especificacions del compilador

B.2.2.2. CONSTRUCCIÓ DE L'ENTORN

Primerament cal construir l'entorn on es crearà el compilador i on es situaran els arxius auxiliars. En tot el manual es farà referència amb aquests noms de carpeta però evidentment es pot fer servir qualsevol nom. L'únic requisit per aquesta ubicació és que no es trobi per sota d'alguna ubicació amb accessos indirectes.

```
$ mkdir -p /stuff/build/conf
$ cd /stuff/
```

Bitbaque és una aplicació que permet emular l'arquitectura per la qual volem crear el compilador. Aquesta aplicació no cal instal·lar-la, tot i que es pot fer amb la comanda emerge, en aquest manual ens baixarem de la xarxa un executable.

```
$ svn co svn://svn.berlios.de/bitbake/branches/bitbake-1.8/ bitbake
```

Actualitzem Bitbake

```
$ cd /stuff/bitbake; svn info
$ cd /stuff/bitbake; svn update
```

Ara cal descarregar una imatge comprimida d'Open Embedded. Primerament cal tornar al directori arrel.

```
$ cd /stuff/
```

Descarreguem i descomprimim la imatge

```
$ wget http://www.openembedded.org/snapshots/OE.mtn.bz2
$ bunzip2 <OE.mtn.bz2  >OE.mtn
```

En aquest punt cal decidir quina branca d'open embedded es vol fer servir. En el cas del nostre robot triem la branca general openembedded.dev.

```
$ mtn --db=/stuff/OE.mtn pull monotone.openembedded.org org.openembedded.dev
$ mtn --db=/stuff/OE.mtn checkout --branch=org.openembedded.dev
```

Per tal de tenir l'ultima versió de tots els components executem l'actualització.

```
$ mtn --db=/stuff/OE.mtn pull monotone.openembedded.org org.openembedded.dev
$ cd /stuff/org.openembedded.dev && mtn update
$ cd /stuff/org.openembedded.dreambox && mtn update
```

B.2.2.3. CREAR LA CONFIGURACIÓ

En aquest pas es configura l'entorn creat per tal que s'adapti a les especificacions que havíem esmentat en la Taula 25.

Entrem en el directori arrel i fem una copia del exemple de configuració que posteriorment editarem.

```
$ cd /stuff/
$ cp org.openembedded.dev/conf/local.conf.sample build/conf/local.conf
```

Per tal de editar l'arxiu podem fer servir qualsevol editor de textos (en l'exemple nano). Si es prefereix es pot fer des d'una finestra gràfica en comptes de la consola.

```
$ vi build/conf/local.conf
```

A continuació es mostra com hauria de quedar el document, les línies de comentaris no s'han inclòs en l'exemple, no fa falta esborrar-les, hi ha prou amb marcar-les com a comentari amb un '#' davant la línia.

```
#
# OpenEmbedded local configuration file (sample)
#
# Please visit the Wiki at http://openembedded.org/ for more info.

# Use this to specify where BitBake should place the downloaded sources into
DL_DIR = "${HOME}/sources"

# Specify which .bb files to consider for
BBFILES := "/stuff/org.openembedded.dev/packages/*/*.bb"

# Use the BBMASK below to instruct BitBake to _NOT_ consider some .bb files
BBMASK = ""

# Select between multiple alternative providers, if more than one is eligible.
PREFERRED_PROVIDERS = "virtual/qte:qte virtual/libqpe:libqpe-opie"
PREFERRED_PROVIDERS += " virtual/libsdl:libsdl-x11"
PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}gcc-initial:gcc-cross-initial"
PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}gcc:gcc-cross"
PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}g++:gcc-cross"

# Machine to build for. See the conf directory
MACHINE = "mtx-1"

# the MACHINE attribute (see above)
TARGET_ARCH = "mipsel"

# Normally the DISTRO of your choosing will take care of this
TARGET_OS = "linux"

# Select a distribution policy. See the conf directory
DISTRO = "nylon"

# following MACHINE types: poodle, tosa and simpad.
MACHINE_KERNEL_VERSION = "2.4.27"

# Add the required image file system types below.
IMAGE_FSTYPES = "jffs2 tar"

# BitBake to emit the log if a build fails.
BBINCLUDELOGS = "yes"
```

En aquest punt s'instal·larà el nou entorn amb la configuració especificada anteriorment.

```
$ export PATH=/stuff/bitbake/bin:$PATH
$ export BBPATH=/stuff/build:/stuff/org.openembedded.dev
```

B.2.2.4. CONSTRUCCIÓ DEL COMPILADOR

A partir d'aquí ja es pot començar a construir el compilador creuat. Donat que open Embedded es va dissenyar per tal de compilar paquets ja existents donarem l'ordre de compilar un paquet senzill. D'aquesta manera automàticament es crearà el compilador.

```
$ bitbake nano
```

Un cop finalitzat ja s'haurà creat el compilador creuat i podrem compilar les nostres aplicacions per al robot. En aquest cas dins l'arxiu "makefile" que gestiona la compilació del nostre codi haurem de fer referència la compilador creuat amb tota la ruta d'accés tal i com es mostra en els exemples de codi.

B.2.3. Comandes Linux

A continuació es detalla una llista de comandes útils per Linux i el seu significat. La majoria d'aquestes comandes permeten afegir opcions de forma similar a DOS. Les opcions s'afegeixen al final de la comanda i tenen dos possibles formats:

`-x` on x = abreviatura de la opció

`--opció` on opció = forma sense abreviar de la opció

Les diferents opcions no es detallen en aquesta llista. Si es vol obtenir ajuda sobre les opcions d'una comanda podem teclejar:

`$man comanda` on comanda = comanda sobre la qual volem ajuda

Les comandes descrites aquí estan escollides pensant en les necessitats per quan es treballa amb el robot. Alguns exemples son específics per l'ordinador de la universitat i no es poden prendre com a genèrics donat que depenen de la màquina.

`$ls` Llista el contingut de la carpeta on ens trobem
`$ /etc/init.d/nom_servei restart` Reinicia el servei nom_servei
`$ /etc/init.d/nom_servei start` Inicia el servei nom_servei (p. exemple net.rausb0 inicia la xarxa rausb0)
`$ /etc/init.d/nom_servei stop` Para el servei nom_servei
`$ cd` Entra inmediatament en el directori home
`$ cd ./nom_carpeta` Entra en el directori nom_dir (ha d'estar situat en la ruta actual)
`$ cd /nom_dir` Entra en el directori nom_dir (ha d'estar situada en l'arrel)
`$ halt` Apaga el sistema
`$ ifconfig` Retorna les configuracions de xarxa del nostre sistema
`$ iwconfig` Igual funcionament que "ifconfig" però per xarxes wifi

`$ mkdir /ruta/nom_dir` Crea una carpeta en la ruta amb el nom nom_dir
`$ mount /ruta/` Munta un dispositiu de memòria definit en fstab (p. Exemple /mnt/usb per un pendrive en l'ordinador de la universitat)
`$ nano /ruta/nom_fitxer` Edita el fitxer de text nom_fitxer (si no existeix es crea automàticament)
`$ ps` Mostra tots els processos actualment en execució
`$ reboot` Reinicia el sistema
`$ rm /ruta/nom_fitxer` Borra el fitxer nom_fitxer (si nom_fitxer és una carpeta també la borra)
`$ scp /ruta/nom_fitxer usuari@xxx.xxx.xxx.xxx /ruta/nou_nom_fitxer` Copia el fitxer nom_fitxer del nostre sistema a un altre (en xarxa) definit pel usuari i la IP (xxx.xxx.xxx.xxx) amb el nou nom.
`$ ssh usuari@xxx.xxx.xxx.xxx` Entra per ssh en un altre sistema de la xarxa
`$ startx` Inicia el motor gràfic (si existeix algun)

`$ su`
Si es treballa com a usuari entra en mode superusuari. Per tornar a sortir teclegem:
`$ exit`

`$ uname –a` Retorna la versió del kernel del nostre sistema

Taula 26. Comandes per consola Linux

B.3. Manual ICD2

A continuació es descriuen les principals prestacions del programador i depurador ICD2. Les explicacions es realitzen prenent com a exemple el programa elaborat per als dsPIC's.

B.3.1. Configuració del port sèrie

L'ICD2 requereix una determinada configuració del port sèrie que fem servir. Si treballem amb Windows aquesta configuració la trobarem a l'administrador de hardware del sistema. Respecte a la configuració per defecte hem de modificar dues opcions. Per una banda s'ha d'activar el control de flux per hardware i per altra banda s'ha de desactivar el buffer FIFO. La Figura 110 i la Figura 111 mostren les finestres de configuració del port.

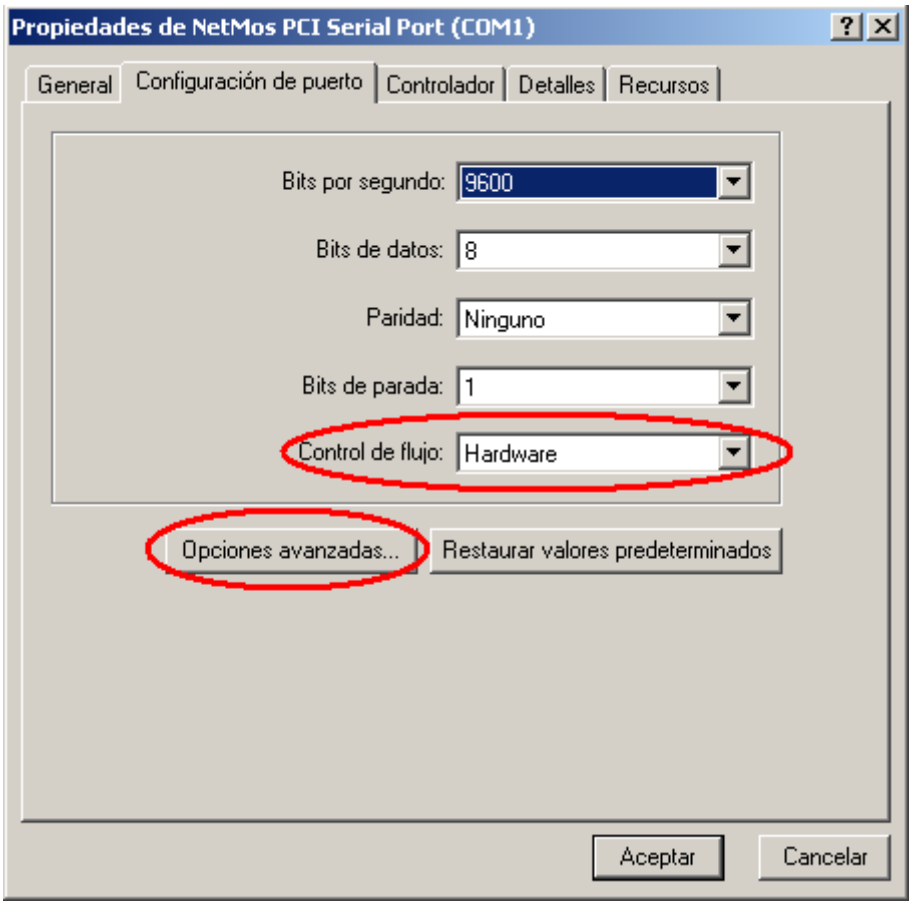

Figura 110. Activació del control de flux per hardware

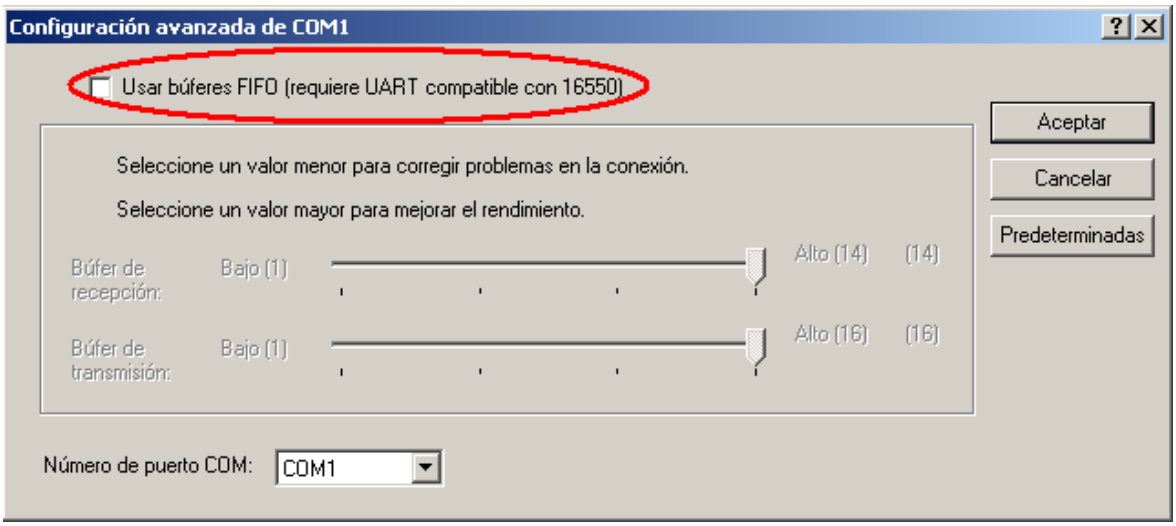

Figura 111. Desactivació del buffer FIFO

B.3.2. L'entorn MPLAB

MPLAB és l'entorn de programació de Microchip per als seus PIC's i dsPIC's. Dins l'entorn MPLAB es poden fer servir compiladors de diferents distribuïdors, per aquest projecte s'ha decidit fer servir el propi compilador de Microchip (C30). Tot i que altres compiladors com ara el PICC30 de HI-TECH també presten compatibilitat amb el ICD2 s'ha observat un millor funcionament amb el C30.

Donat que per al robot es treballa amb dos programes diferents a l'hora (un per cada dsPIC) hem d'activar primerament l'opció de treballar simultàniament amb dos projectes. Per això cal anar al menú "> Configure > Settings" i la pestanya "Projects", aquí s'ha de desmarcar l'opció "Use one to one project-workspace model". La Figura 112 mostra la finestra on hem de desmarcar la casella de verificació.

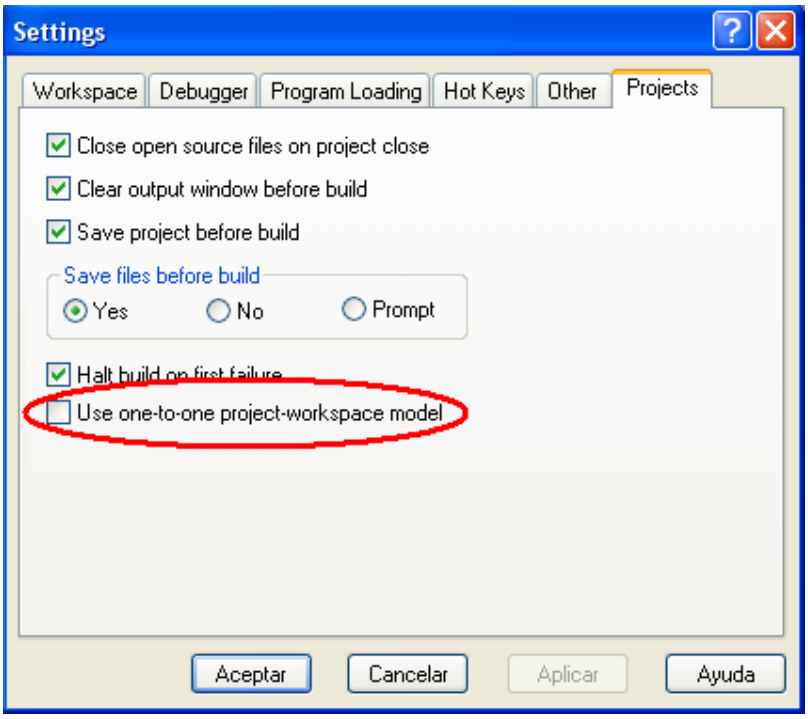

Figura 112. Configuració de MPLAB per múltiples projectes

Si no realitzem aquest pas previ no podrem obrir l'espai de treball creat per la programació dels dsPIC's.

Ara ja es pot obrir l'espai de treball "dsPIC" que conté el projecte per al dsPIC de davant i el projecte per al dsPIC de darrera. La Figura 113 mostra la distribució de les finestres que s'ha fet servir per l'elaboració dels programes. Aquesta distribució permet veure a primera vista tota la informació rellevant de cada programa.

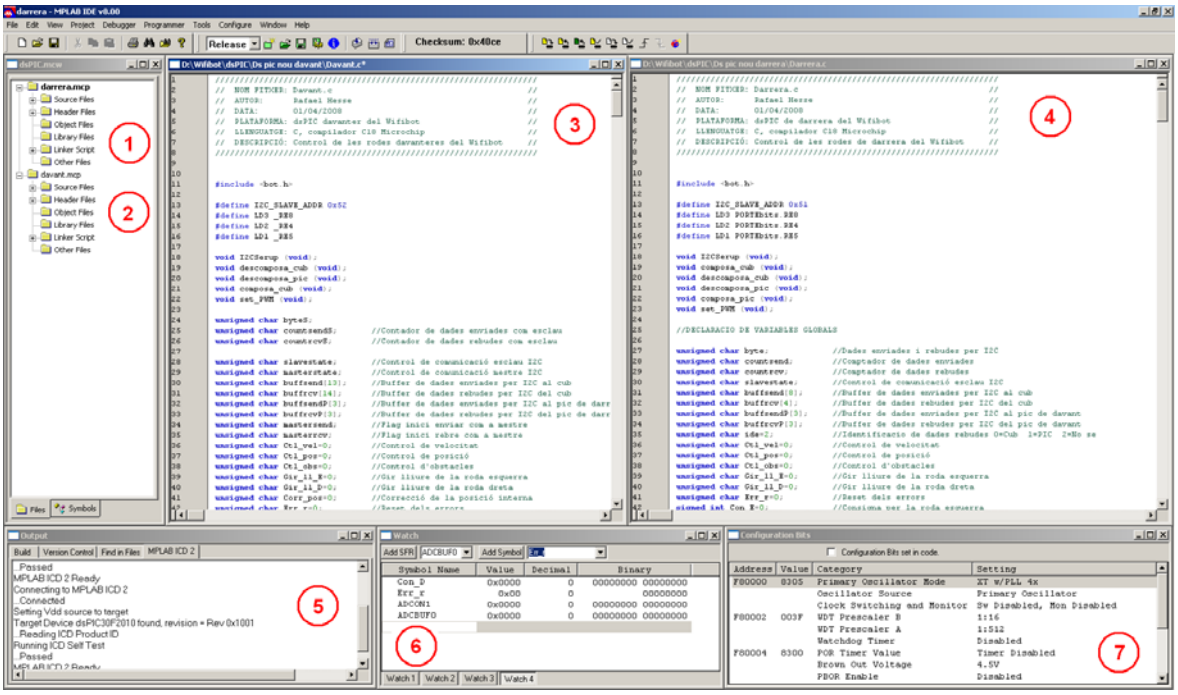

Figura 113. Espai de treball MPLAB

La Taula 27 descriu cada una de les finestres numerades de la figura x. En tot cas aquesta és una distribució i visualització proposada que en tot cas es pot modificar segons les necessitats. Les finestres pròpies de MPLAB les podem visualitzar des del menú "> View > nom_finestra". Les finestres que mostren el codi creat les podem obrir des de l'arbre de projecte de l'esquerra.

1	Arbre del projecte del dsPIC de darrera. Conté tots els arxius de codi i auxiliars de compilació per al projecte. La lletra negreta marca el projecte actualment actiu.
2	Arbre del projecte del dsPIC de davant. Conté tots els arxius de codi i auxiliars de compilació per al projecte. La lletra negreta marca el projecte actualment actiu.
3	Codi del programa principal del dsPIC de davant.
4	Codi del programa principal del dsPIC de darrera
5	Output: Mostra el resultat de les accions de compilar, programar i depurar. Aquí es mostren els errors en cas de produir-se
6	Watch: En mode de depuració ens permet visualitzar l'estat de determinades variables del programa
7	Bits de configuració: Si els bits de configuració no es programen en el codi es poden programar de forma contextual en aquesta finestra.

Taula 27. Definicions respecte la Figura 113

B.3.3. ICD2 com a programador

Quan es fa servir l'ICD2 com a programador aquest grava el programa en el dsPIC sense les funcions de depuració, un cop programat el programa s'executarà un cop es faci un reset. A continuació es descriu pas per pas el procediment de programació.

Primer s'ha de seleccionar el programador. En el menú "> Programer > select programer > MPLAB ICD2". Aquesta acció ens farà aparèixer una sèrie de botons en la barra d'eines per controlar el programador (Figura 114. Barra d'eines del programador).

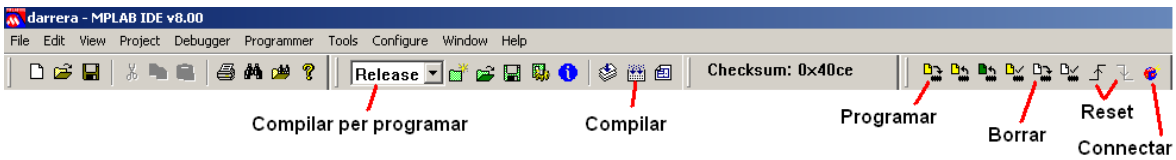

Figura 114. Barra d'eines del programador

Segons el dsPIC que volem programar s'ha de seleccionar com a actiu el corresponent projecte (botó dret del ratolí sobre el projecte i seleccionar "Set active") En la interfície de programació s'ha de situar el selector rotatiu en la posició del corresponent dsPIC. El selector de depuració es deixa en la posició central.

Per compilar el codi del projecte seleccionat donarem al botó de compilar. Si el codi conté errors aquests es visualitzaran en la finestra output, altrament acabarà amb el missatge "build succeed" tal com mostra la Figura 115.

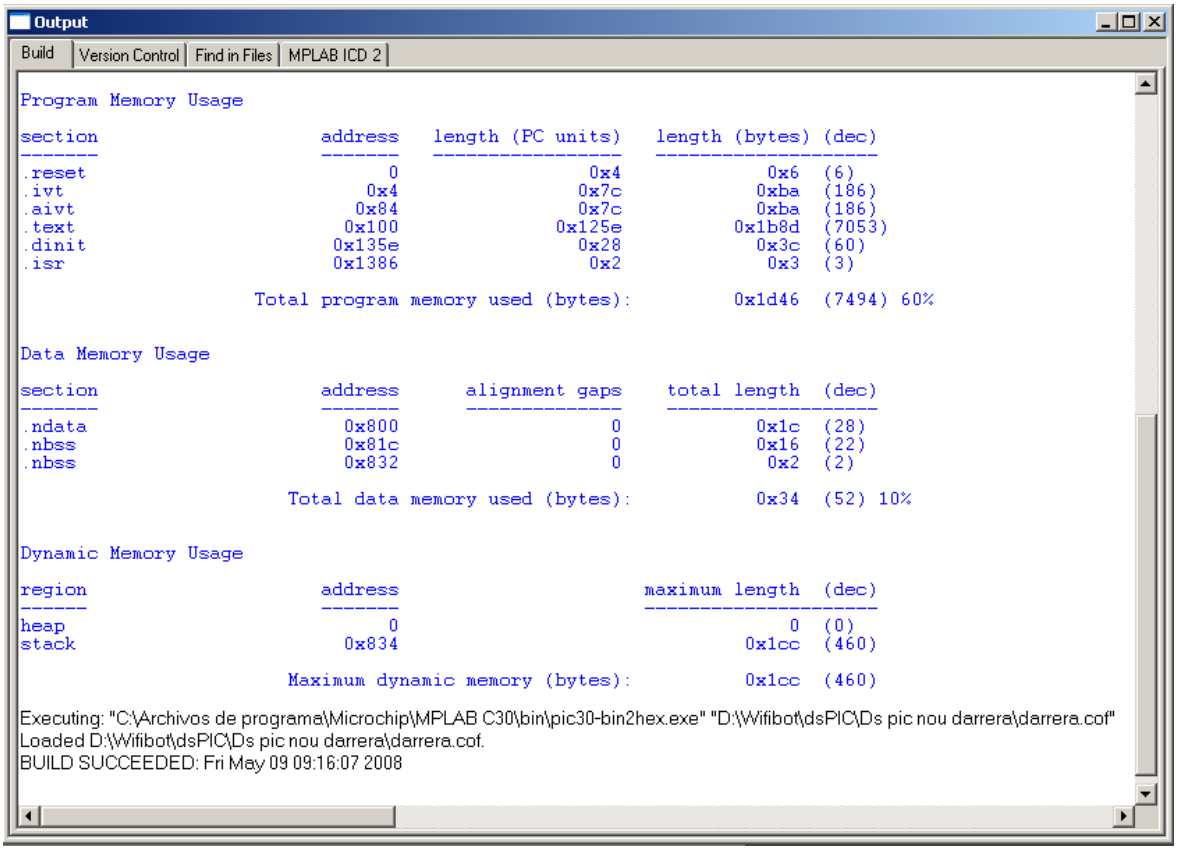

Figura 115. Finestra Output després d'una compilació correcta.

En aquesta finestra també podem veure la quantitat de memòria que ocuparà el programa en el dsPIC. Aquesta quantitat de memòria pot variar segons el mode de compilació. El compilador C30 ens permet compilar el codi en funció de si es desitja obtimitzar per velocitat d'execució o per memòria ocupada. Un programa optimitzat per velocitat d'execució usualment ocuparà més memòria mentre que un programa optimitzat per memòria ocupada usualment trigarà més en executar-se. Aquesta configuració la podem modificar en el menú "> Project > Build options... > Project" La Figura 116 mostra la finestra de configuració, podem seleccionar el mode de compilació sobre el mateix gràfic.

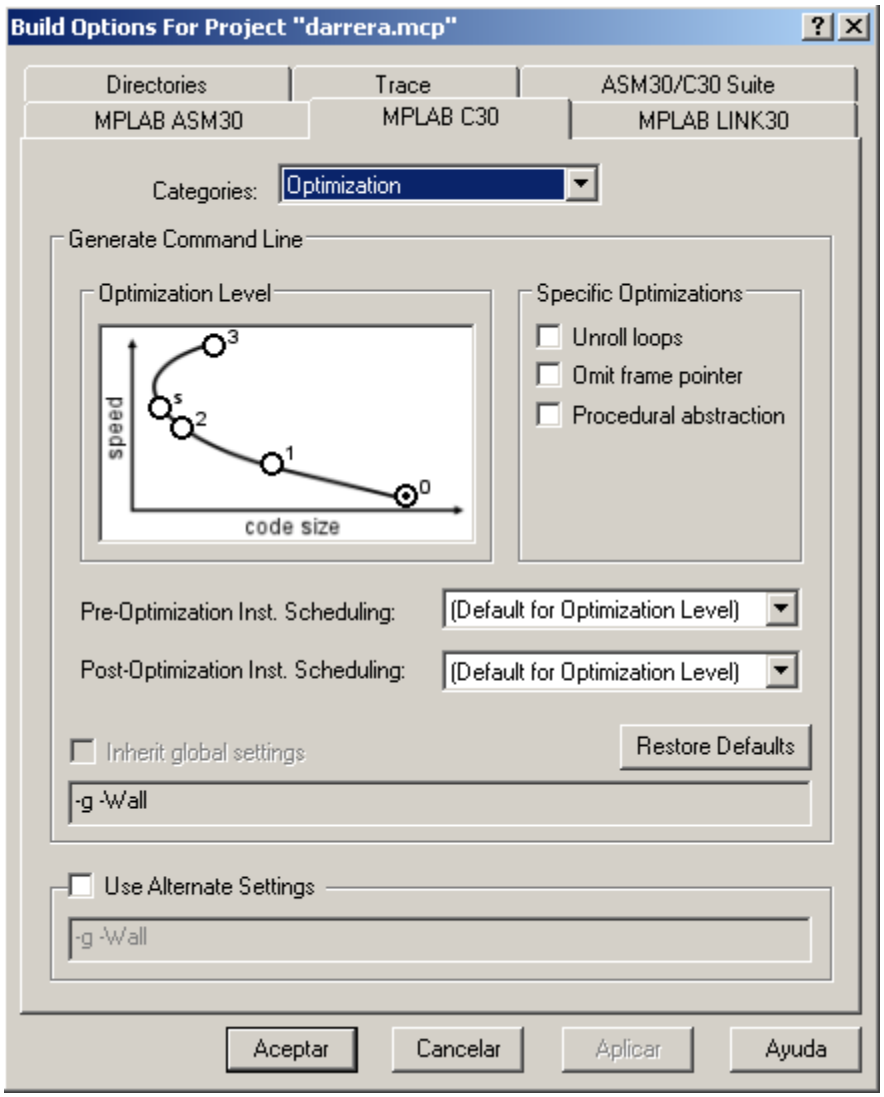

Figura 116. Optimització del compilador

Per tal d'iniciar la comunicació amb el programador basta amb clicar el botó "connect". En aquest punt l'entorn MPLAB es connectarà amb L'ICD2 i comprovarà si conté el sistema operatiu adient. Si aquest no fos el cas es descarregarà de forma automàtica, aquest procediment triga una mica i durant la descàrrega pot donar la impressió de que l'ICD2 hagi quedat bloquejat.

Si la connexió s'ha establert correctament i s'ha detectat un dsPIC connectat al ICD2 en la finestra "Output" s'observarà un missatge com el de la Figura 117

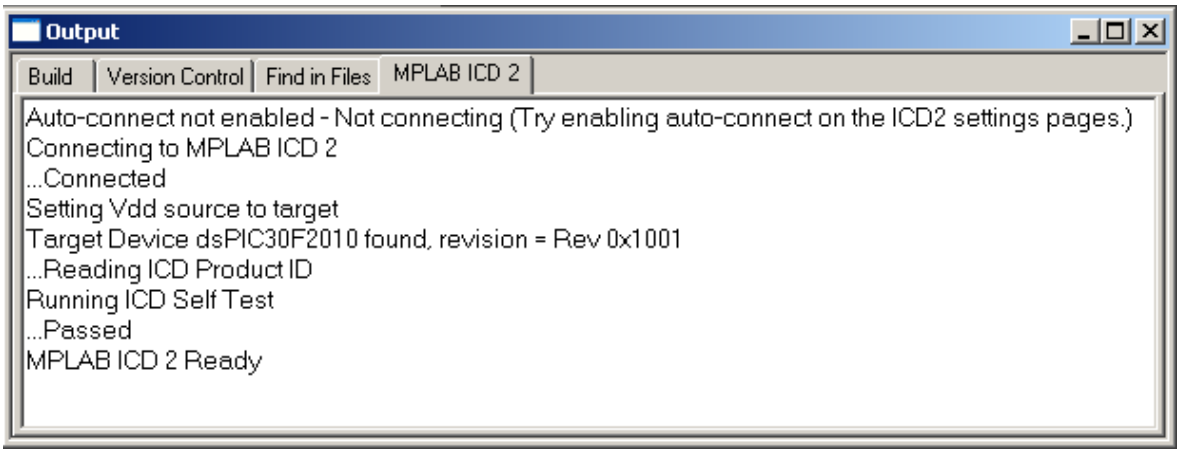

Figura 117. Connexió establerta amb l'ICD2

A partir d'aquest punt només queda transmetre el programa al dsPIC. Farem clic sobre el botó "Pogram" i esperarem a que acabi el procés. Aquest pot durar mes o menys temps en funció del tamany del programa. Si la programació resulta exitosa la finestra "Output mostrarà un missatge similar al de la Figura 118.

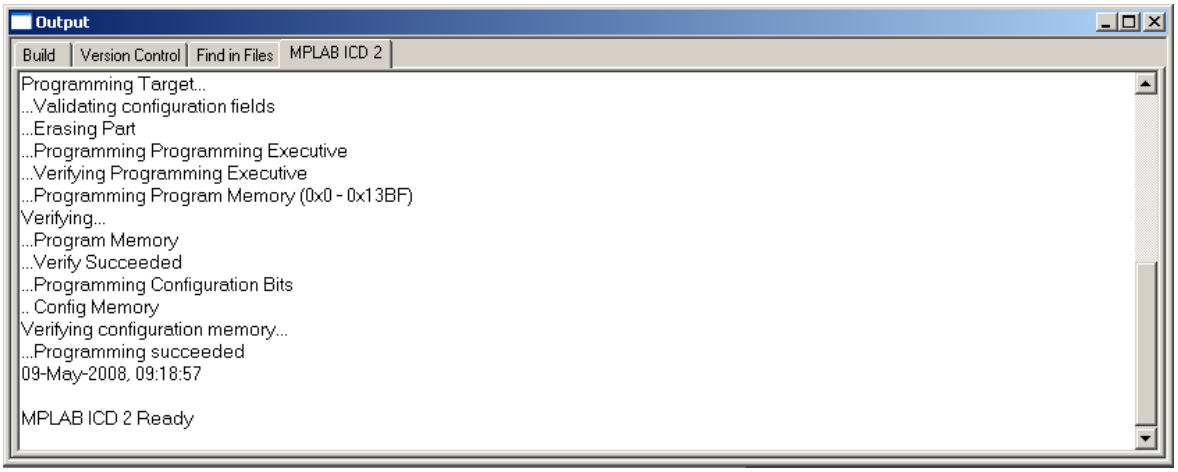

Figura 118. Programació correcta del dsPIC

Per tal que s'executi el programa hi ha dues opcions, o es desconnecta el programador o es posa remotament el reset en estat alt mitjançant el corresponent botó. Només queda recordar que si es precisen funcions I^2C s'ha de girar el selector de la interfície en la posició I^2C o desconnectar-la i connectar l'adaptador.

B.3.4. ICD2 com a depurador

En aquest mode de funcionament l'ICD2 ens servirà per programar el dsPIC i a demés per depurar el programa o trobar falles. En general el mode de funcionament és molt similar al de programador però amb diverses funcions afegides. A continuació es descriu pas per pas com carregar un programa en mode de depuració en els dsPIC's.

Primer s'ha de seleccionar el depurador. En el menú "> Debugger > Select tool > MPLAB ICD2". Amb aquesta acció tornarà a aparèixer una nova barra d'eines que ens permeten controlar el funcionament de l'ICD2 com a depurador. La Figura 119 mostra la funció de les principals eines.

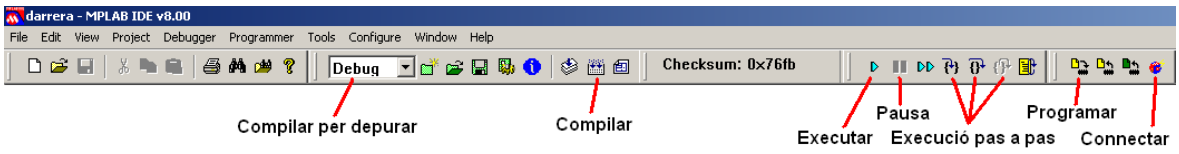

Figura 119. Barra d'eines del depurador

En mode de depuració l'ICD2 fa servir un port per programar que és el mateix que en el cas de programador i un altre port que es fa servir per l'actualització de l'estat de les variables. En el cas del robot aquest port es pot seleccionar entre PGC/PGD o EMUC2/EMUD2. Si fem servir el primer no podrem fer servir el bus I2C durant la depuració (donat que es troba en les mateixes potes) per aquest motiu es dona per suposat que es faran servir les segones. Aquestes s'han de seleccionar entre els bits de configuració (Figura 120).

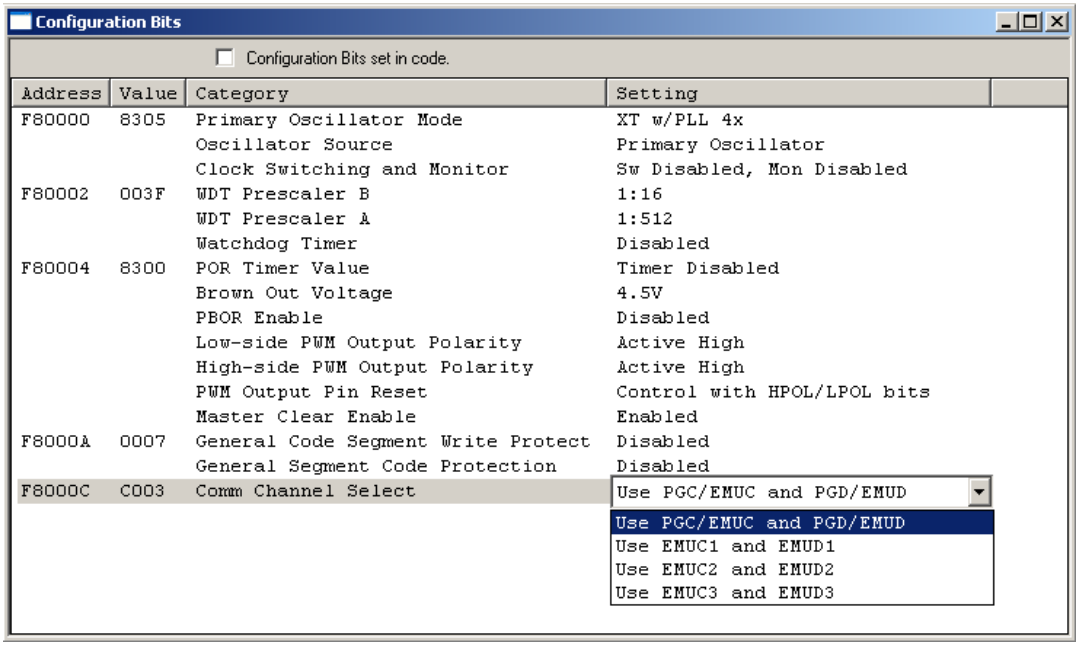

Figura 120. Selecció dels ports per depurar

Segons el dsPIC que volem programar s'ha de seleccionar com a actiu el corresponent projecte (botó dret del ratolí sobre el projecte i seleccionar "Set active").

En la interfície de programació s'ha de situar el selector rotatiu en la posició del corresponent dsPIC. El selector de depuració es deixa en la posició central en cas de fer

servir els ports PGC/PGD o seleccionant el corresponent dsPIC en cas de fer servir els ports EMUC2/EMUD2.

Per compilar el codi del projecte seleccionat donarem al botó de compilar. Si el codi conté errors aquests es visualitzaran en la finestra output, altrament acabarà amb el missatge "build succeed" tal com mostra la Figura 121.

En aquest cas es pot observar que l'extensió del codi augmenta per la qual cosa es pot donar el cas de que un dsPIC es pugui programar però no depurar per excessiu tamany del codi.

En aquest mode de funcionament determinades configuracions no son possibles, la més destacat es el Watch dog timer. En mode depuració aquest no el podem configurar com a actiu.

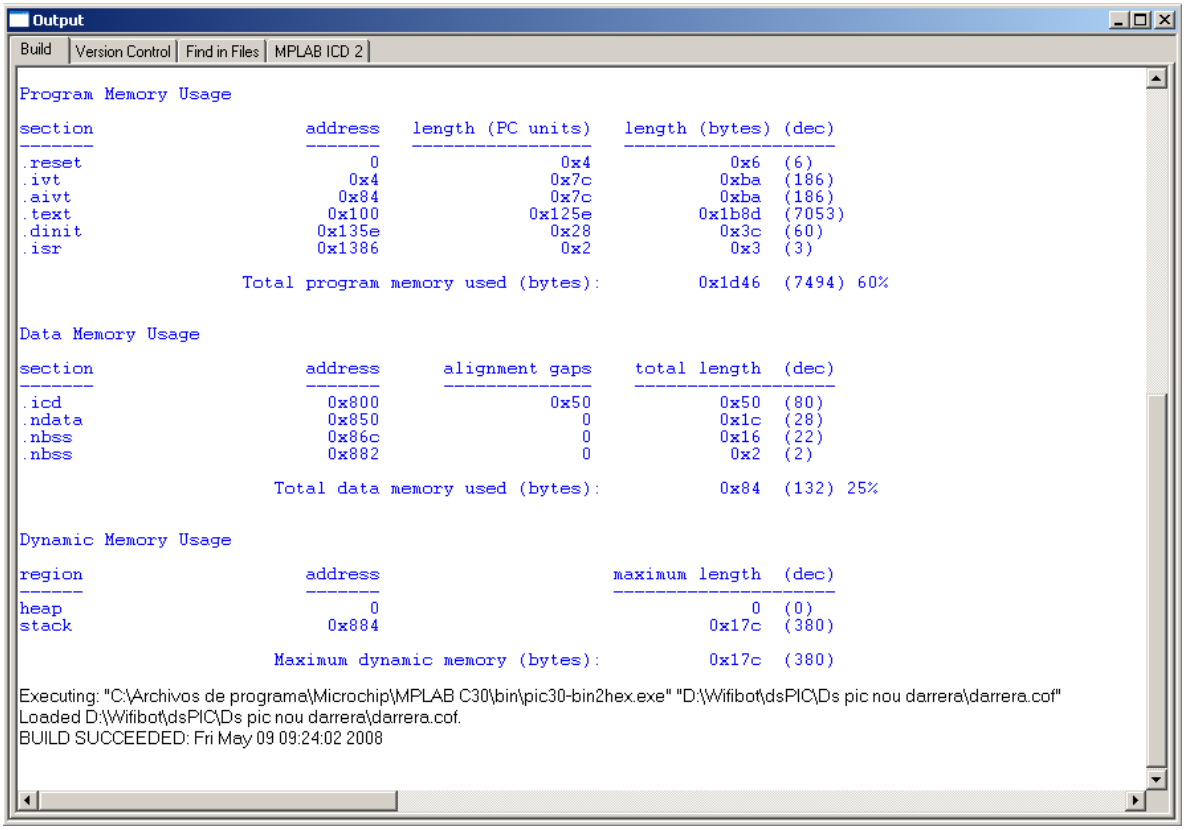

Figura 121. Finestra Output després d'una compilació correcta en mode depuració.

Per tal d'iniciar la comunicació amb el programador basta amb clicar el botó "connect". En aquest punt l'entorn MPLAB es connectarà amb L'ICD2 i comprovarà si conté el sistema operatiu adient. Si aquest no fos el cas es descarregarà de forma automàtica, aquest procediment triga una mica i durant la descàrrega pot donar la impressió de que l'ICD2 hagi quedat bloquejat.

Si la connexió s'ha establert correctament i s'ha detectat un dsPIC connectat al ICD2 en la finestra "Output" s'observarà un missatge com el de la Figura 122

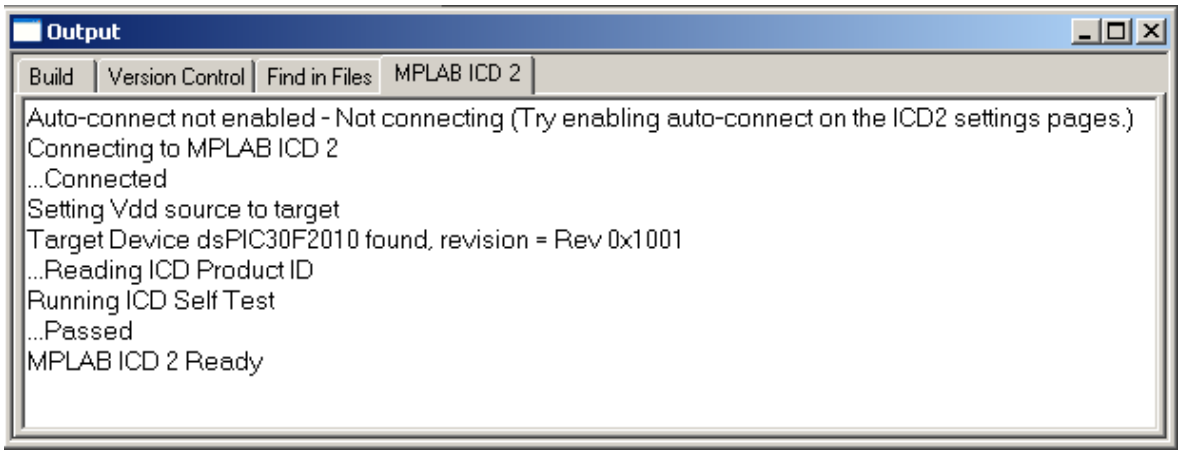

Figura 122. Connexió establerta amb l'ICD2

Si anteriorment ja havíem carregat un programa en el dsPIC en mode depuració es pot donar un error en aquest punt donat que el dsPIC segueix executant la rutina de depuració. En tal cas s'ha de provocar un reset del dsPIC i iniciar la comunicació novament.

Un cop establerta correctament la comunicació es pot carregar el programa en el dsPIC donant al botó de programació. Si aquesta s'efectua exitosament obtindrem una finestra de sortida similar a la Figura 123.

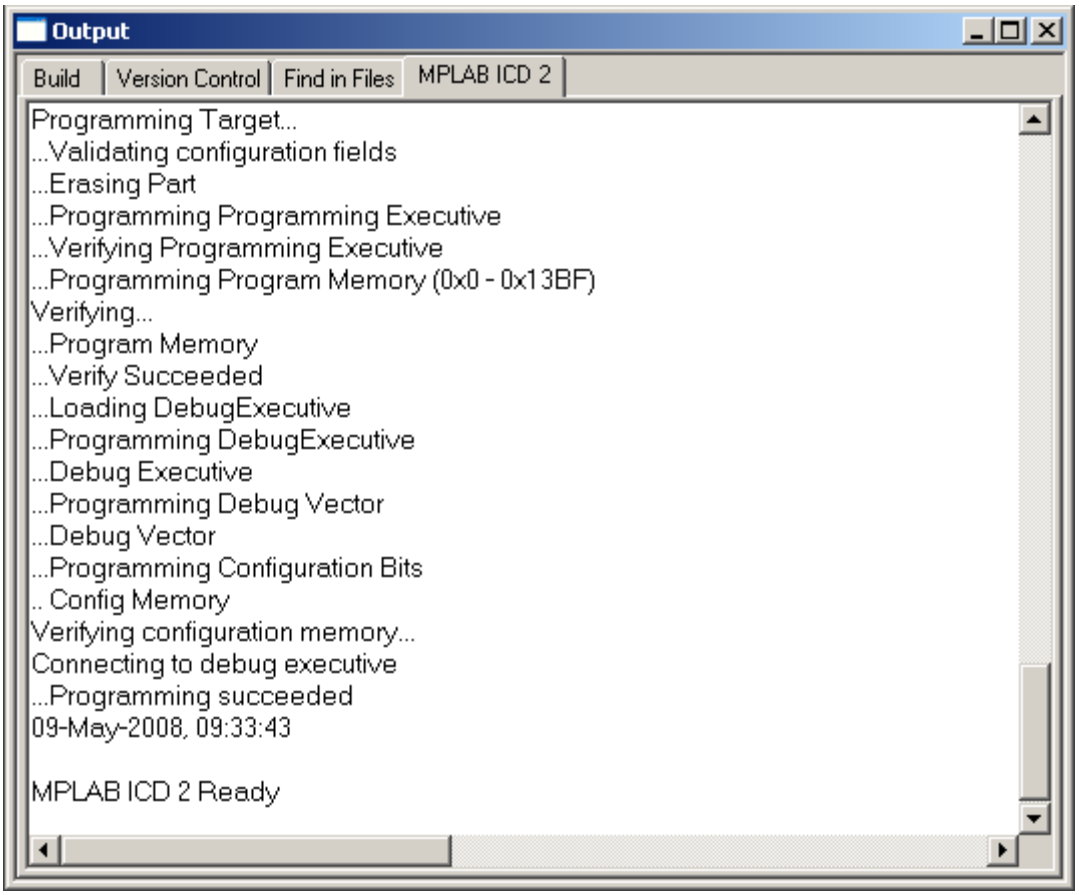

Figura 123. Programació correcta en mode depuració

Un cop programat el dsPIC correctament es pot executar de diverses formes el programa. La més senzilla és l'execució contínua (F9). El programa s'executarà fins que finalitzi o fins que es posi en pausa manualment (F5). Si parem l'execució l'entorn MPLAB mostrarà amb una fletxa verda el punt exacte on s'ha parat.

Un cop el programa es troba parat podem executar un sol pas (F7). En aquest mode de funcionament es recomana tenir en compte quina serà la propera instrucció. Donat que en mode pas a pas s'executa cada instrucció (en llenguatge màquina) una a una controlat des del PC es pot donar el cas que una línia de programa en C equivalgui a nombroses instruccions i per tant pot trigar molt. Per altre banda s'ha d'evitar entrar en un bucle infinit buit en mode pas a pas, en aquest cas quedaria bloquejat tot el sistema.

Cada cop que es para l'execució del programa en el dsPIC s'actualitza l'estat de les variables en el PC. A través de la finestra Watch podem observar l'estat de les variables globals del nostre programa i dels registres interns del dsPIC. La Figura 124 mostra un exemple per aquesta finestra. Podem afegir a la llista tants registres i variables com vulguem.

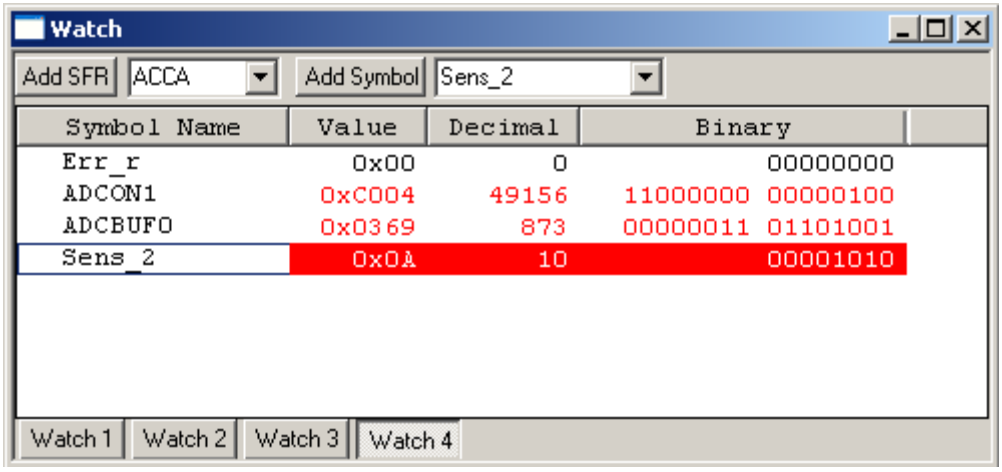

Figura 124. Finestra Watch actualitzada

Les variables marcades en vermell son aquelles que han canviat d'estat des de l'ultima actualització.

Mentre el programa es troba parat podem forçar l'estat de qualsevol de les variables. Si seleccionem alguna de les variables o registres podem modificar el valor (en qualsevol format). Els canvis tindran efecte tan bon punt es premi enter. En la Figura 125 es mostra a mode d'exemple com modificar els dos últims bits de la variable Err_r.

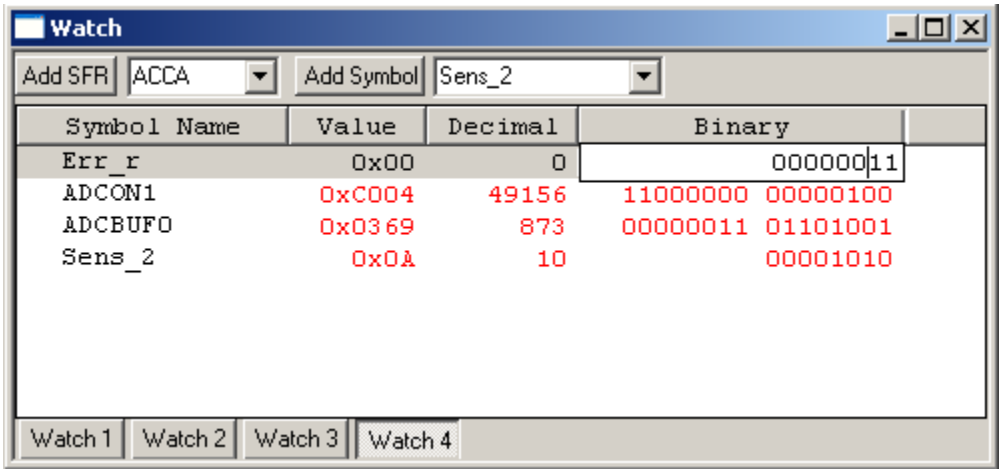

Figura 125. Forçar l'estat de variables i registres

Un breakpoint (punt d'interrupció) permet parar l'execució del programa en un punt determinat de forma automàtica. Si fem doble clic sobre una línia del codi del programa s'inserirà un breakpoint (punt vermell de la Figura 126). Si ara s'executa el programa aquest es tornarà a parar en el moment l'execució passi per aquesta instrucció.

Aquest model de dsPIC només permet inserir un sol breakpoint actiu a la vegada. Si volem treballar amb més breakpoints a la vegada hem de canviar l'actiu cada cop que es pari l'execució del programa. Els punts actius es marquen amb un punt vermell mentre

que els inactius es marquen amb un cercle. Amb la tecla F2 apareix un menú que ens permet seleccionar quin breakpoint es actiu.

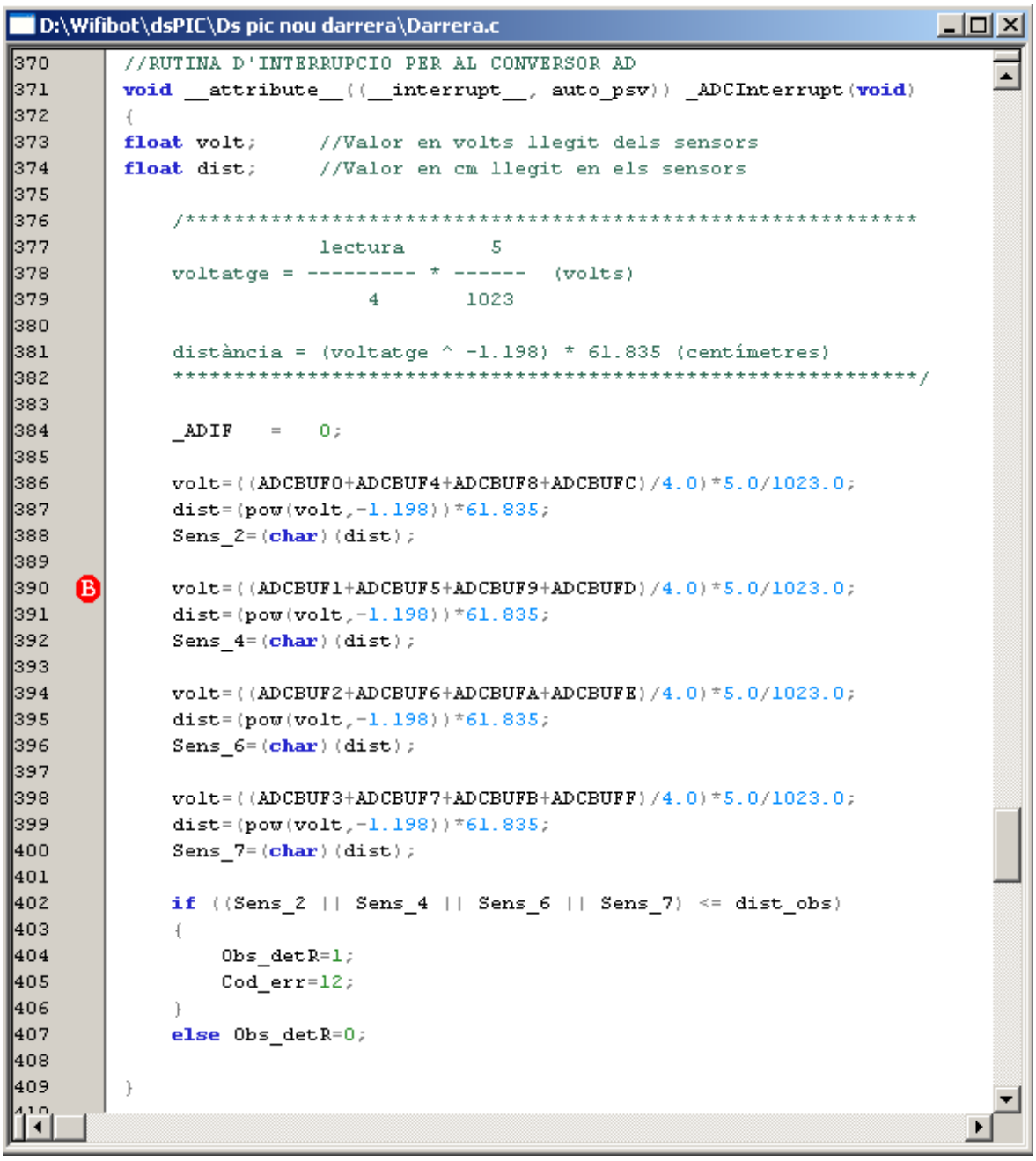

Figura 126. Breakpoint

ANNEX C. COMUNICACIONS I CÀLCULS

En aquest apartat es resumeixen les dades que circulen pel bus I^2C entre els dsPIC's i el cub i les dades que circulen per wifi entre el programa pont del cub i una aplicació externa. L'objectiu es servir com a guia ràpida a l'hora de crear una nova aplicació per al cub o una aplicació externa.

En aquest punt no es descriu el significat dels diversos bits o bytes. Per una informació més detallada s'ha de consultar el corresponent apartat de la memòria.

C.1. Comunicacions I^2C

Les dades transmeses en aquest nivell es separen en diferents parts segons quin element es l'emissor i quin es el receptor..

C.1.1. Dades enviades des del cub al dsPIC de darrera

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	M_enc	Gir_ll	Err_r	Ctl_obs	Ctl_pos	Ctl_vel	Ident.

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_E	Con_E						

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_D	Con_D						

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

C.1.2. Dades retornades des del dsPIC de darrera al cub

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_des	Err_cons	Err_com	Cod_err				

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
signe	Enc_E						

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_D							

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_1							

byte 4	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_2							

byte 5	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_3							

byte 6	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_4							

byte 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

C.1.3. Dades enviades des del cub al dsPIC de davant

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Corr_pos	Ctl_Tf	Gir_ll	Err_r	Ctl_obs	Ctl_pos	Ctl_vel	M_enc

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_E	Con_E						

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_D	Con_D						

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Con_x_cm						

byte 4	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Con_y_cm						

byte 5	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_x_m							

byte 6	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_y_m							

byte 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_T							

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Pos_x_cm						

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Pos _y_cm						

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _x_m							

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _y_m							

byte 12	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _T							

byte 14	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

C.1.4. Dades retornades des del dsPIC de davant al cub

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_des	Err_cons	Err_com	Cod_err				

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_E							

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
sentit	Enc_D						

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_1							

byte 4	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_2							

byte 5	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_3							

byte 6	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_4							

byte 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Pos_x_cm						

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Pos _y_cm						

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _x_m							

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _y_m							

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _T							

byte 12	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8	RE_5	RE_4	TRISE_8	TRISE_5	TRISE_4

C.1.5. Dades enviades des del dsPIC de davant al de darrera

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
1	x	x	x	Sentit_E	Sentit_D	Obs_det	Ident

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_E							

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_D							

C.1.6. Dades retornades pel dsPIC de darrera al de davant

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
1	x	x	x	x	x	Obs_det	x

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
signe	Enc_E						

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_D							

C.2. Comunicacions wifi

En aquest apartat es resumeixen les dades que podem enviar al robot des de l'exterior i la corresponent resposta que donarà el robot. Igualment no s'especifica el significat de cada un dels bits. Per informació més detallada es pot consultar la memòria.

C.2.1. Dades que podem enviar al robot des de l'exterior

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	Gir_ll_R	Gir_ll_F	x	x	Ctl_obs	Ctl_pos	Ctl_vel

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	x	M_env	Ctl_Tf	Err_r_R	Err_r_F	Corr_pos

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_EF	Con_EF						

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_DF	Con_DF						

byte 4	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_ER	Con_ER						

byte 5	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sentit_DR	Con_DR						

byte 6	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Con_x_cm						

byte 7	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Con_y_cm						

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_x_m							

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_y_m							

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Con_T							

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Pos_x_cm						

byte 12	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Pos _y_cm						

byte 13	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _x_m							

byte 14	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _y_m							

byte 15	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _T							

byte 16	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8F	RE_5F	RE_4F	TRISE_8F	TRISE_5F	TRISE_4F

byte 17	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8R	RE_5R	RE_4R	TRISE_8R	TRISE_5R	TRISE_4R

C.2.2. Dades retornades pel robot cap a l'exterior

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Lectura conversor AD (0-255)							

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_desF	Err_consF	Err_comF	Cod_errF				

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Err_desR	Err_consR	Err_comR	Cod_errR				

byte 3:6	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_EF – Enc_ER – Enc_DF – Enc_DR							

byte 7:14	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_1 : Sens_8							

byte 15	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_x	Pos_x_cm						

byte 16	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Signe_y	Pos _y_cm						

byte 17	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _x_m							

byte 18	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _y_m							

byte 19	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos _T							

byte 20	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
x	x	RE_8R	RE_5R	RE_4R	RE_8F	RE_5F	RE_4F

C.3. Càlculs

Aquest annex recull els càlculs I les equacions que s'han fet servir per les diverses aplicacions d'aquest projecte

C.3.1. Odometria.

Els càlculs de posició del robot es deriven de les equacions bàsiques de l'odometria (equacions 3-6)

$$\Delta O = \frac{(\omega_d + \omega_e)\cdot D}{2}\cdot \Delta t$$

(Eq. 3)

$$\Delta X = \frac{(\omega_d + \omega_e)\cdot D}{2}\cdot \Delta t \cdot \cos(\sigma)$$

(Eq. 4)

$$\Delta Y = \frac{(\omega_d + \omega_e)\cdot D}{2}\cdot \Delta t \cdot \sin(\sigma)$$

(Eq. 5)

$$\Delta \sigma = \frac{\omega_d - \omega_e}{B}\cdot \Delta t$$

(Eq. 6)

S'entén que aquestes equacions expressen increments de posició del centre de masses del robot (ΔO), increment sobre els eixos (ΔX i ΔY) i l'increment de l'angle ($\Delta\sigma$). El valor "D" equival al diàmetre de les rodes i el valor "B" donat que es tracta d'un vehicle de quatre rodes es pot aproximar a la diagonal del robot. La Figura 127 mostra el significat de les demés variables de les equacions.

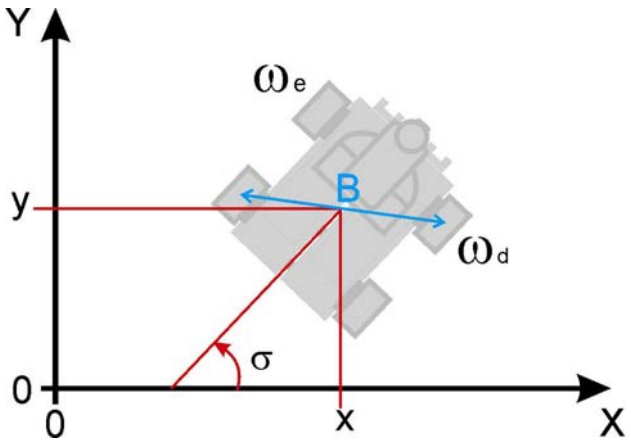

Figura 127. Paràmetres de la odometria

Donat que les lectures dels encoders ens donen un valor proporcional al desplaçament i no a la velocitat es poden adaptar les equacions anteriors per tal de calcular amb valors de desplaçament. Tal com es mostra en les properes equacions aquest fet fa desaparèixer de les equacions el terme del temps.

$$\Delta O = \frac{(U_d + U_e) \cdot D}{2}$$

(Eq. 7)

$$\Delta X = \frac{(U_d + U_e) \cdot D}{2} \cdot \cos(\sigma)$$

(Eq. 8)

$$\Delta Y = \frac{(U_d + U_e) \cdot D}{2} \cdot \sin(\sigma)$$

(Eq. 9)

$$\Delta \sigma = \frac{U_d - U_e}{B}$$

(Eq. 10)

En aquestes últimes equacions es troben dues unitats que son proporcionals entre elles, per una banda tenim les distàncies i mesures en metres i per altra banda tenim les lectures dels encoders que es troben en “tics”. Per tal que al llarg dels càlculs apareguin el mínim nombre possible de decimals s’ha decidit utilitzar el “tic” com a unitat de mesura de distància. A partir d’aquí es plantegen les següents definicions:

Donat que un encoder dona 6000 tics per volta de la roda i que el diàmetre de la roda “D” és de 0.127m tenim:

$$K = \frac{D \cdot \pi \cdot 1000}{6}$$

(Eq. 11)

Aquest valor K serveix com a constant per passar de metres a “tics” i a l’invers. Així podem expressar “B” en “tics”:

$$B = \frac{K}{0.33}$$

(Eq. 12)

Així es poden expressar les equacions de la odometria finalment donant el resultat en “tics” on s’entén que ΔN equival a l’increment de polsos de l’encoder des de l’instant n-1 a l’instant n.

$$\Delta O = \frac{(N_d + N_e)}{2} \left[\text{tics}\right]$$

(Eq. 13)

$$\sigma_n = \sigma_{n-1} + \frac{N_d - N_e}{B} \left[\text{rad}\right]$$

(Eq. 14)

$$X_n = X_{n-1} + \Delta O \cdot \cos(\sigma) \left[\text{tics}\right]$$

(Eq. 15)

$$Y_n = Y_{n-1} + \Delta O \cdot \sin(\sigma) \left[\text{tics}\right]$$

(Eq. 16)

A partir d'aquestes equacions podem obtenir fàcilment la posició del robot en metres traduint els valors obtinguts en "tics". Guardant els històrics dels valors de posició en "tics" permet mantenir molta més precisió davant l'opció de guardar-los en metres.

C.3.2. Lectura de l'estat de la bateria

L'estat de la bateria es dedueix a partir del resultat que dona la lectura del conversor analògic digital. Aquesta lectura dona la tensió que hi ha en borns del divisor de tensió que hi ha a l'entrada del conversor. Donat que es tracta d'un conversor de 8 bits (256 valors) i que la referència es troba entre 0 i 5 volts la tensió de la bateria es dedueix a partir de:

$$V_{bat} = V_{Ain} \cdot 2.5 = 2.5 \cdot \text{lectura} \cdot \frac{5}{255} = \frac{\text{lectura} \cdot 25}{51} \left[\text{volts}\right]$$

(Eq. 17)

C.3.3. Lectura i linealització dels sensors de distància.

Els sensors de distància infrarojos es llegeixen en els dsPIC's a través d'entrades analògiques que permeten llegir un valor de tensió comprès entre 0 i 5 volts. Donat que es tracta d'un conversor de 10 bits aquest ens retorna un valor entre 0 i 1023 proporcional a la tensió aplicada. D'aquí tenim que la tensió en borns al sensor equival a:

$$V_{sens} = \text{lectura} \cdot \frac{5}{1023} \left[\text{volts}\right]$$

(Eq. 18)

Aquesta tensió llegida equival a una certa distància, aquesta es pot calcular aproximant la resposta del sensor a una equació exponencial (Figura 128). Així la distància es calcula com:

$$D = V_{sens}^{-1.198} \cdot 61.835$$

(Eq. 19)

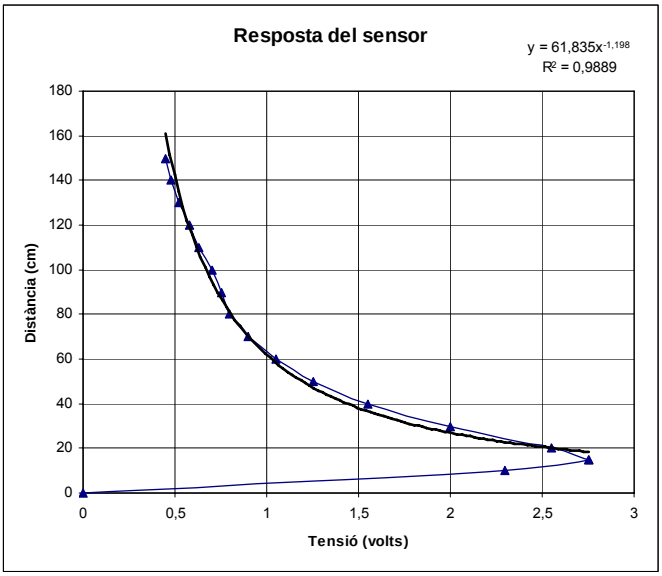

Figura 128. Aproximació de la resposta del sensor