EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi: Grau Enginyeria Electrònica Ind. i Automàtica

Títol: Implementació en ROS de l'Arquitectura Software del Robot Mòbil BIGBOT

Document: 1. Memòria

Alumne: Pere Vila Soler

Director/Tutor: Xavier Cufí i Albert Figueras
Departament: ATC / EEEA
Àrea: ATC / ESA

Convocatòria (mes/any): febrer/2014

ÍNDEX

1. INTRODUCCIÓ

El que es planteja és implementar l'arquitectura sofware del ROS en el robot BigBot.

1.1. ANTECEDENTS

El Robotic Operating System (ROS) està esdevenint un estàndard molt reconegut per molts grups de recerca que treballen en la implementació de sistemes robotitzats i més concretament en el desenvolupament de software per a les plataformes robotitzades.

El ROS proporciona serveis generals d'un sistema operatiu com són l'abstracció del hardware, el control de baix nivell de dispositius, la implementació de funcionalitats, el pas de missatges entre processos i la gestió de paquets d'informació.

El paquet ROS permet el treball en moltes àrees d'aplicació directament relacionades amb la robòtica mòbil, la percepció de l'entorn i la navegació de robots.

Aquest fet permet compartir els esforços realitzats pels diferents grups quan es pretén desenvolupar l'arquitectura software d'un robot determinat.

1.2. OBJECTE

L'objectiu és la implementació en ROS de l'arquitectura software del robot mòbil de rescat BIGBOT que ha estat desenvolupat en el laboratori de sistemes intel·ligents de l'EPS.

1.3. ESPECIFICACIONS I ABAST

Concretament es treballarà en la implementació progressiva dels controladors dels sensors (Làser Range Finder URG 04LX d'Hokuyo, una càmera RGB-D, i el sensor inercial 3DM-GX1), dels motors i dels seus codificadors rotatius (anomenats també encóders).

2. ESTRUCTURA HARDWARE DEL BIGBOT

Actualment la flota de robots de rescat de l'Arlab consta de 3 models diferents de robots, però és descriurà només l'estructura del BigBot, el més gran de la flota, i en el qual se li aplicarà la millora que es descriu en aquest document.

Figura 1. BigBot

L'estructura hardware actual del BigBot es basa en un switch ethernet en el que es connecten els diferents sensors. A partir d'aquí, el programa de control de l'ordinador es comunica amb els diferents sensors a través de les IPs que el switch els assigna.

Aquesta estructura implica un seguit d'inconvenients. El primer és que la majoria de sensors i actuadors no es comuniquen via ethernet, per la qual cosa es fa necessària la incorporació d'un conversor port sèrie-ethernet.

Un altre inconvenient és el temps que es perd en la transmissió d'informació entre el robot i l'ordinador. Aquest retard és causat pel nombre de capes de transmissió per les que ha de passar el paquet de dades.

La figura 2 mostra l'estructura esquemàtica de la connexió dels diferents sensors i actuadors del BigBot.

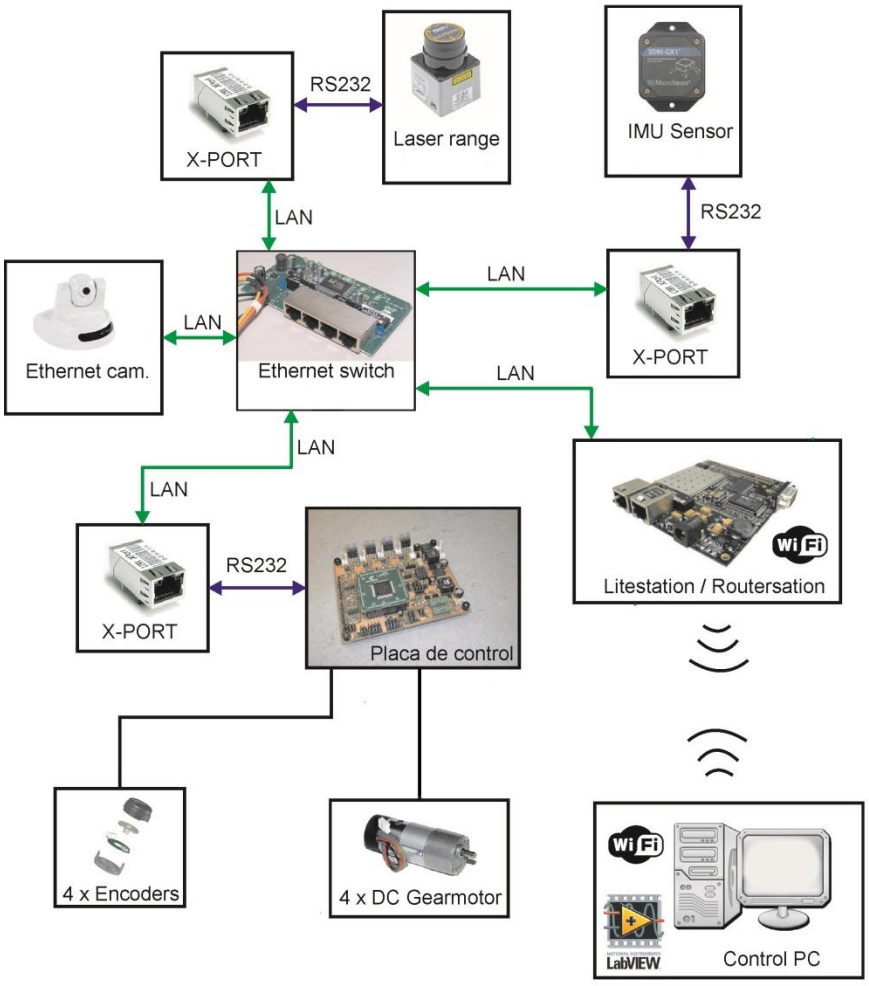

Figura 2. Esquema BigBot original

Com es pot observar a la figura, tots els sensors i actuadors fan una conversió de RS232 a ethernet excepte la càmera ethernet i el router, que s'encarrega d'empaquetar les dades i transmetre-les a l'ordinador.

A continuació es farà una petita descripció dels diferents elements del BigBot, i se'n mostraran algunes especificacions.

2.1. PLACA DE CNTROL

Es tracta d'una placa dissenyada per la UdG per tal de suplir la necessitat de controlar els motors del robot. Aquesta esta basada en un microcoltrolador PIC, el qual se l'hi ha instal·lat

un programa en C que s'encarrega de les tasques de control de més baix nivell. A la taula següent es poden observar les especificacions del PIC.

Model: dsPIC33FJ256MC	
Arquitectura	16 bit – 40MHz
Memòria de programa	256 kB
Memòria RAM	30720 Bytes
Encapsulat	100 pin TQFP (85 i/o pins)
Control d'execució	PBOR, POR, WDT
Canals de memòria DMA	8
Entrades analògiques	24 x 12-bit @ 500 (ksps) 2-A/D
Comunicacions digitals	2 x UART, 2 x SPI, 2 x ECAN, 2 x I2C
Canals per control de motors per PWM	8 (16 bit resolution)
Canals Input-Capture	8
Timers	9 x 16 bit, 4 x 32 bit
Interfície QEI	1

Taula 1. Especificacions PIC

Encara que la principal tasca de la placa és el control dels quatre motors, s'hi desenvolupen altres prestacions com la lectura dels polsos de l'encóder mitjançant comptadors d'alta velocitat. També es fa la lectura dels corrents de cada motor, mitjançant cel·les hall es mesura el corrent consumit per cada motor. I per últim el circuit inclou fonts d'alimentació commutades de 3.3 i 5V per tal d'alimentar els diversos dispositius del robot. És per això que també fa el control de voltatge de la bateria i del circuit d'alimentació.

La comunicació entre el PIC i el PC es fa mitjançant trames de bytes variables a través de protocol TCP/IP, les quals són descrites amb tot detall a l'annex E.

2.2. MOTORS

Com s'ha dit a l'apartat anterior, els motors del robot estan dominats per la placa de control, que disposa de 4 ponts en H per tal de subministrar la potència demanada a cada motor de forma independent.

Degut a les dimensions del BigBot, els motors han de tenir una potència considerable per moure'l. Amb l'objectiu de tenir el parell motor suficient, s'hi ha col·locat un reductor mitjançant engranatges que el que fa és reduir-ne la velocitat de l'ordre de 65.5 a 1 per obtenir un parell nominal de 3.4N m. A la taula següent es mostren alguns altre valors interessants del motor

Model: GM9236S025	
Tensió nominal	12 V
Parell nominal	3.4 N·m
Velocitat en buit	71 rpm
Corrent en buit	330 mA
Corrent màxim (eix bloquejat)	16.9 A
Diàmetre de l'eix	6.4 mm
Factor del reductor	65.5:1
Encóder	
CPR	500
Canals	2 + índex

Taula 2. Especificacions Motors i encóders

2.3. BATERIES

La bateria del robot es conforma per 10 bateries de Níquel - Metall Hidrur de tamany estàndard LR20 de 10.000 mA·h i 1.2V. Així s'obté una bateria de tensió nominal 12V i una capacitat de 10 A·h. La tensió de la bateria es llegeix mitjançant un conversor AD del dsPIC de la placa de control. Durant el funcionament del robot és recomanable comprovar de forma periòdica el nivell de la bateria per tal d'evitar la descàrrega excessiva.

2.4. LASER RANGE FINDER

Aquest és un sensor de distància làser per escombrat, és a dir, és capaç de realitzar un escombrat el seu voltant i de mesurar la distancia dels objectes que es troben dins l'àrea d'abast. Aquest sensor es accessible a través de la xarxa ethernet o wifi creant un socket amb el corresponent X-Port).

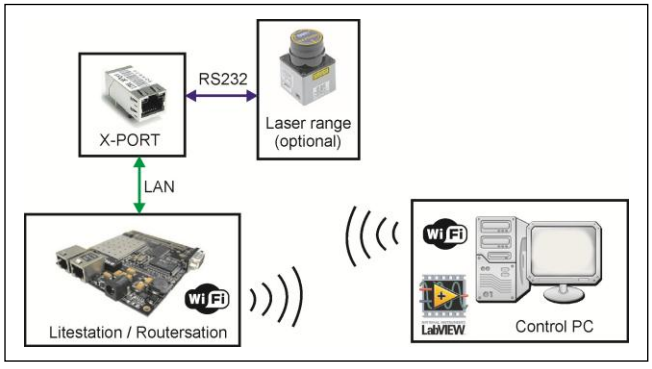

Figura 3. comunicació làser

Un cop creat el socket, les comandes i respostes es poden interpretar pel port sèrie. La taula 2 descriu les principals característiques del sensor. Cal tenir en compte, però, que alguns dels paràmetres s'expressen en valors límit, ja que poden ser modificats per soft.

Model: URG-04LX-UG01	
Font de llum	Semiconductor làser díode (λ=785nm) Classe 1 0.8 mW
Alimentació	5V DC
Consum màxim	500mA
Precisió	0.06-1m: ±30mm, 1-4m: 3% de la distància detectada
Resolució	1 mm
Angle d'escombrat	240º
Resolució angular	360º/1024
Temps d'escombrat	100 ms
Interfície	RS-232C (19.2k, 57.6k, 115.2k, 500k, 750kbps) USB 2.0 (12Mbps)

Taula 3. Especificacions làser range finder

2.5. GIROSCOPI

Mitjançant el giroscopi es pot obtenir amb molta precisió la posició angular del robot respecte al camp magnètic de la terra. Així mateix, el giroscopi permet també obtenir en diversos formats la seva acceleració i velocitat angular en els tres eixos. Aquest sensor es troba, igual que el laser range finder, connectat al robot mitjançant un X-Port. En aquest cas és possible connectar-s'hi de dues formes com mostra la figura 4.

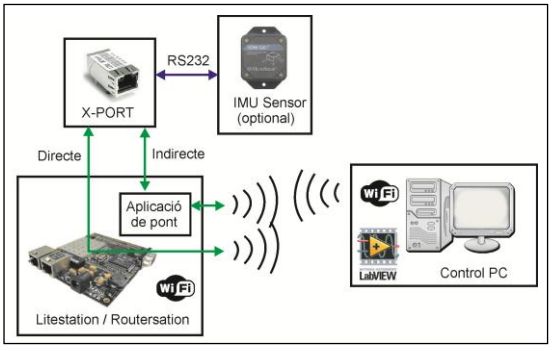

Figura 4. Comunicacions giroscopi

Una és l'opció directa, a través de la qual es crea des del sistema de control un socket directe amb l'X-Port per tal de configurar i llegir valors del giroscopi. La segona és l'opció

indirecta, a través de la qual el socket amb l'X-Port el crea una aplicació de la Litestation per tal que es pugui fer un tractament previ de les dades. Si es vol disposar de les lectures al sistema de control exterior, cal llegir-les per un socket amb el programa ubicat a la Litestation per tal que reenviï les dades.

En la taula següent es detallen les principals prestacions del giroscopi. Alguns dels paràmetres descrits son paràmetres límit i poden dependre de la configuració del sistema.

Model: 3DM-GX1	
Marge de mesura angular (pitch, roll, yaw)	360° tots els eixos (orientation matrix, quaternion) ± 90°, ± 180°,± 180° (Euler angles)
Rang dels girs	± 300º/s
Rang dels acceleròmetres	± 5 g
Rang dels magnetòmetres	± 1.2 Gauss
Rang dels conversors A/D	16 bits
Resolució d'orientació	0.1º
Repetibilitat	0.2º
Precisió	± 0.5º (proves estàtiques) ± 2.0º (proves dinàmiques)
Comunicacions	RS-232 (19.2, 38.4, 115.2 kbaud)
Alimentació	5.2 – 12 V DC
Onsum	65 mA

Taula 4. Especificacions Giroscopi

3. ROS

Sistema Operatiu Robòtic (en anglès Robot Operating System, ROS) és un sistema operatiu pensat per al desenvolupament de programari per a robots que proveeix la funcionalitat d'un sistema operatiu en un clúster heterogeni. ROS va ser un projecte que va començar sota el nom de Switchyard el 2007 al departament d'Intel·ligència Artificial de Stanford per tal de donar suport al projecte STAIR2, (Stanford AI Robot 2, en català Robot d'Intel·ligència Artificial de Stanford). Aquest projecte, actualment i des de el 2008, es desenvolupa a Willow Garage, un institut d'investigació robòtic amb més de vint institucions col·laborant en un model de desenvolupament federat.

Figura 5. Logo ROS

ROS subministra els serveis estàndard d'un sistema operatiu com ara abstracció del maquinari, control de dispositius de baix nivell, implementació de funcionalitat d'ús comú, pas de missatges entre processos i manteniment de paquets. La mentalitat d'aquest sistema operatiu es basa en una arquitectura de grafs on el processament es du a terme en els nodes. Aquests poden rebre, enviar i multiplexar missatges de sensors, control, estats, planificacions i actuadors, entre altres. La comunicació entre nodes es fa a través de Tòpics, aquest són missatges que publiquen els nodes periòdicament i que poden contenir diferents tipus d'informació que processen.

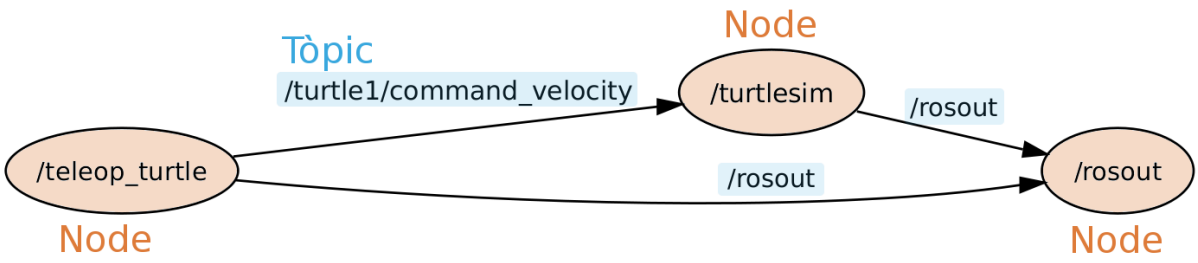

Figura 6. Exemple Nodes i Tòpics

ROS és un sistema operatiu complementat amb el que s'anomena col·loquialment ROS-pkg, que és el cúmul de nodes i aplicacions aportades per la comunitat d'usuaris. S'organitzen en

diferents paquets (també anomenats stacks) que implementen diverses funcionalitats com poden ser la localització i mapatge simultani, la planificació, la percepció, la simulació, etc.

La llibreria està orientada a un sistema UNIX (Ubuntu (Linux)). És el sistema que suporta completament el ROS, encara que actualment també s'està adaptant a altres sistemes operatius com Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows, tot i que de moment es considera en mode de proves. A més, el ROS és programari lliure sota termes de llicència BSD. Aquesta llicència permet un ús comercial i també investigador. Les contribucions dels paquets dels usuaris, d'altra banda, estan sota una gran varietat de llicències diferents.

3.1. ROS FUERTE

En aquest cas, la versió que s'utilitzarà de ROS és Fuerte.

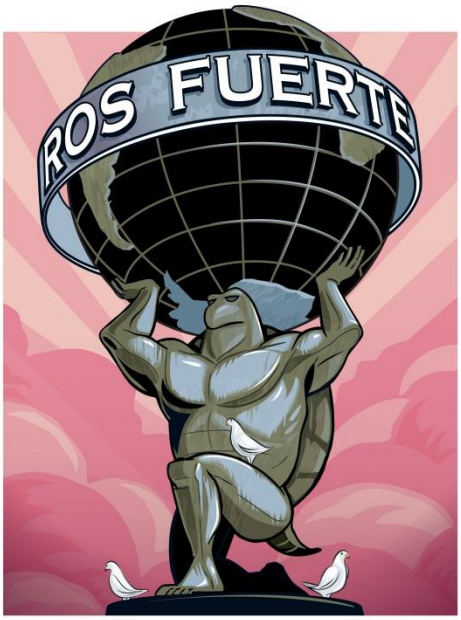

Figura 7. Logo ROS Fuerte

Aquesta és la cinquena versió desenvolupada d'aquest sistema operatiu. Va ser publicada el 23 d'abril de 2012 i bàsicament aquesta versió va presentar importants millores que el feien més fàcil d'integrar a altres softwares o eines de programació. Per exemple, existeix un nou sistema que permet compilar arxius a llenguatges que no són purament ROS.

Bàsicament es va decidir utilitzar ROS Fuerte ja que aquesta és la versió que ja s'ha implementat a altres laboratoris de recerca de la UdG i d'aquesta manera, es podia aprofitar l'experiència prèvia i alhora apropar una mica més aquests dos laboratoris.

4. INSTAL·LACIÓ ROS

Per tal d'instal·lar el ROS sobre un ordinador amb Ubuntu com a sistema operatiu, bàsicament el que s'ha de fer és obrir la Terminal i introduir un seguit de comandes que ja vénen pautades pel mateix fabricant. En aquest cas, se suposa que la versió d'Ubuntu de la qual es disposa, és 12.04 (Precise). Si no fos així, alguna de les comandes canviaria, o seria possible que no fos compatible amb ROS Fuerte.

Per començar s'hauran de configurar les fonts d'Ubuntu per tal que acceptin programari de de fonts "restricted", "universe" i "multiverse". Un cop fet això, es pot començar amb la instal·lació.

Primerament s'haurà d'obrir la Terminal d'Ubuntu i caldrà que s'escrigui la següent línia de comandes que agrega una font més a la llista.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > etc/apt/sources.list.d/ros-latest.list'
```

Tot seguit, s'aconseguiran les claus amb la comanda que es mostra a continuació.

```
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Aleshores s'assegurarà que s'hagi re-indexat el servidor de ROS.org des de el que el programa baixarà la informació que després instal·larà.

```
$ sudo apt-get update
```

Un cop s'hagi arribat aquí, hi haurà diferents tipus d'instal·lacions. A continuació només es mostra la instal·lació completa, però si es volgués, també es podria fer una instal·lació amb menys complements.

```
$ sudo apt-get install ros-fuerte-desktop-full
```

Aquesta última comanda pot arribar a tardar considerables minuts, fins i tot hores. Això dependrà de la velocitat de descàrrega de la línia i de la velocitat de processament de l'ordinador en què s'instal·li.

Un cop s'hagi completat aquesta comanda ja es tindrà el ROS instal·lat, però encara faltarà fer unes quantes configuracions.

S'haurà d'associar l'arxiu setup.bash de fuerte a l'arxiu .bashrc.

```
$ echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
```

```
$ .~/.bashrc
```

Si és té més d'una versió de ROS instal·lada, l'arxiu ~/.Bashrc només pot tenir associat el setup.bash de la versió de ROS que s'utilitzi actualment. Si no és així, es poden produir problemes de funcionament.

També és interesant instal·lar un seguit d'eines que s'utilitzen freqüentment en l'ús o manipulació de paquets, com són el rosinstall i el rosdep.

```
$ sudo apt-get install python-rosinstall python-rosdep
```

Per tal d'estalviar problemes, s'ha d'assegurar que les variables ROS_ROOT i ROS_PACKAGE_PATH estan establertes.

```
$ export | grep ROS
```

Per últim es crearà un Workspace on es podran guardar tots els programes que es desenvolupin. Per fer-ho, el que primer es necessitarà serà crear una carpeta nova amb el nom ros_workspace.

```
$ mkdir ~/ros_workspace
```

Seguidament es vincularà aquesta carpeta a l'arxiu .bashrc per tal que sàpiga que aquesta és la carpeta de treball.

```
$ echo "export ROS_PACKAGE_PATH=~/ros_workspace:$ ROS_PACKAGE_PATH" >>~/.bashrc
```

```
$ echo "export ROS_WORKSPACE=~/ros_workspace" >> ~/.bashrc
```

I per acabar s'afegiran les variables de configuració de la xarxa ROS.

```
$ echo "export ROS_HOSTNAME=localhost" >> ~/.bashrc
```

```
$ echo "export ROS_MASTER_URI=http://localhost:11311" >> ~/.bashrc
```

Un cop fet això ja es tindrà el sistema operatiu ROS completament configurat.

És molt important saber que per executar el ROS i per tal de fer funcionar tots els nodes implementats en ROS, abans de fer qualsevol altra cosa, s'haurà d'obrir una terminal d'Ubuntu i executar la comanda d'inicialització, que és la següent.

```
$ roscore
```

Aquesta terminal no es podrà tancar mentre el sistema operatiu s'estigui executant ni tampoc s'hi podrà escriure res.

A continuació també s'explica com instal·lar una eina molt útil per visualitzar imatges captades des de la càmera o el sensor làser, entre d'altres. Servirà per comprovar que els sensors implementats funcionen correctament. Aquesta eina és el Rviz, un potent programa que permet la subscripció als tòpics que són publicats pels nodes de càmeres, de sensors làser, etc. Rviz els interpreta i els codifica de manera que l'usuari pugui entendre la informació.

Per tal de descarregar aquest programa i fer possible la comprensió de les dades, introduïm aquesta comanda a la terminal.

```
$ sudo apt-get install ros-fuerte-visualization
```

En aquest moment només faltarà crear el paquet.

```
$ rosdep install rviz
```

```
$ rosmake rviz
```

Per tal d'executar-lo, només s'haurà d'introduir la l'ínia de comandes següent.

```
$ rosrun rviz rviz
```

Quan s'hagi fet, s'obrirà un finestra com la que és mostra a continuació.

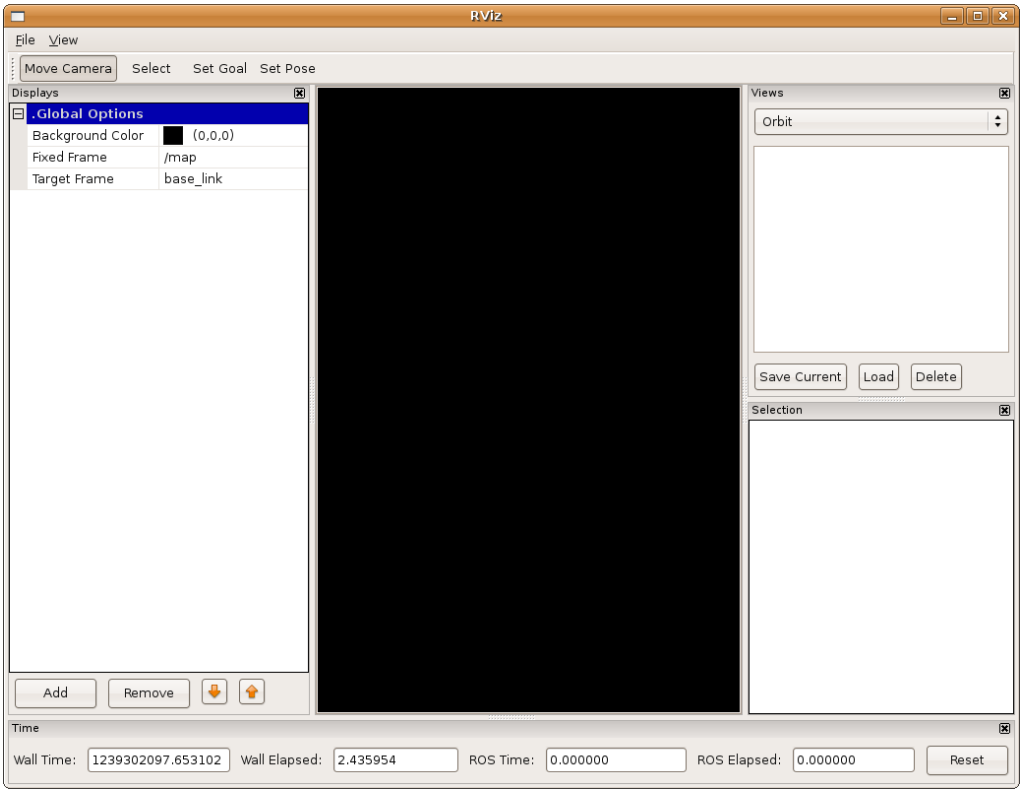

Figura 8. Rviz

Prement el botó Add podrem agregar nodes del tipus desitjat (càmera, laser range finder, imu, etc.), que es subscriuran als tòpics i els interpretaran mostrant-los per pantalla.

5. SOLUCIO APLICADA

Ara que ja s'ha descrit l'estructura de hardware del robot i el ROS, es detallarà la solució aplicada per la implementació d'aquest sistema operatiu al BigBot.

Per començar, el que s'intentarà serà mantenir al màxim l'estructura dels components físics del robot. Tenint en compte això, s'optarà per connectar els diferents sensors via USB a l'ordinador i instal·lar-ne els controladors sempre que estiguin disponibles, tal com es mostra en la imatge següent.

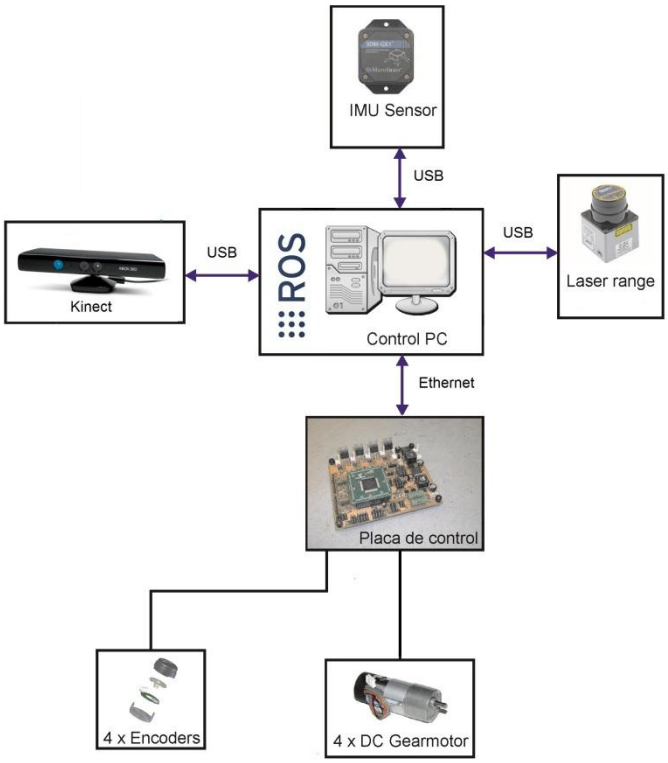

Figura 9. Esquema desitjat

Cal destacar que en aquest moment la part més important del robot serà un ordinador. Abans, en canvi, era la placa wifi qui rebia, tractava i enviava les dades a un ordinador extern per tal que aquest fes els càlculs necessaris i li retornés les ordres adients. Aquesta comunicació entre la placa wifi i l'ordinador extern comportava un retard en la transmissió d'informació. Així doncs, es podrà reduir aquest temps amb el nou disseny i implementació.

Com també es pot veure a la imatge, es vetllarà per tenir el mínim nombre d'intermediaris possible. És per això que es treurà el switch i conseqüentment s'eliminaran els conversors

RS232-ethernet. També s'eliminarà la placa wifi i es connectaran directament per USB tots els sensors. Per altra banda, es connectarà per ethernet la placa de control, cosa que farà que es redueixin considerablement els retards.

L'estructura que en un principi es desitjava és la que es representa a la figura 10. No obstant, aquesta primera idea finalment resulta no ser possible degut a la antiguitat del sensor inercial. Aquest és un model suficientment antic perquè no es trobin controladors per ROS ni per Ubuntu, però la companyia productora d'aquest sensor subministra un codi en C que permet comunicar-s'hi (és el mètode que ja s'utilitzava fins ara). Utilitzar-lo comporta l'inconvenient d'haver de conservar el conversor X-Port de l'IMU i el switch, ja que els ordinadors convencionals nomes tenen un port ethernet.

Després de tenir en compte tots aquests aspectes, l'esquema definitiu quedarà de la manera següent.

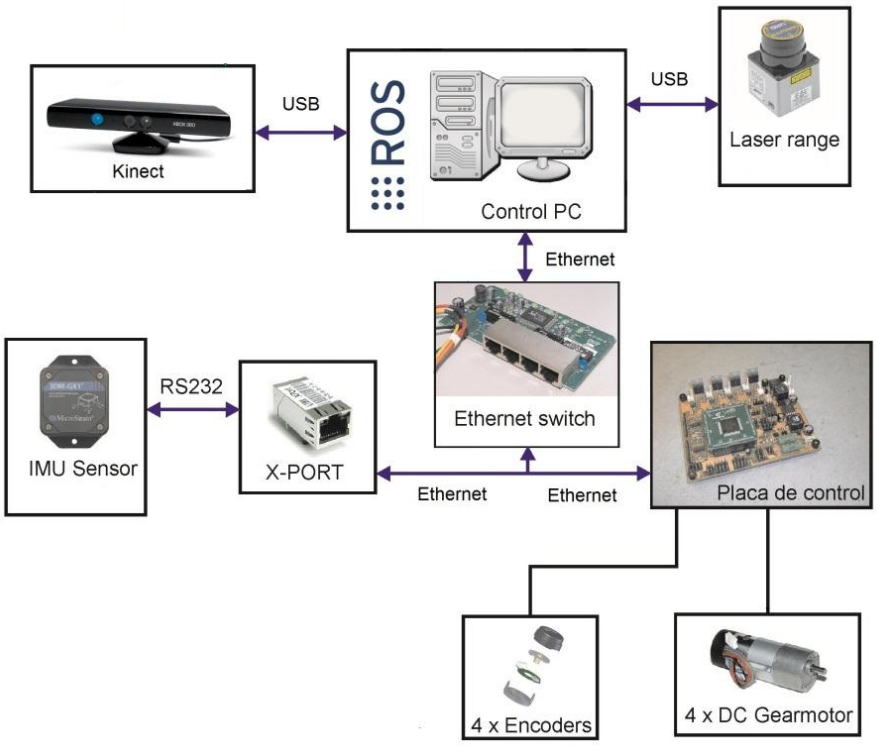

Figura 10. Esquema final

5.1. IMPLEMENTACIÓ DEL LASER RANGE FINDER

Un telèmetre és un dispositiu que permet mesurar distàncies des del sensor fins a un objectiu físic. El més utilitzat és el làser, a causa del seu relatiu baix cost, la seva precisió, i la facilitat d'ús. El seu funcionament es basa en la mesura del temps que triga un puls làser en arribar a un objectiu i tornar. Això permet mesurar de manera molt precisa des de distàncies que no siguin excessivament llargues (d'alguns centímetres), fins a 20 metres. Això dependrà del model que s'utilitzi en cada cas. El seu principal inconvenient és que si es troba en situacions de molta llum, no detecta correctament el feix laser reflectit per l'objectiu.

El telèmetre làser que s'implementarà en aquest cas és el model URG 04LX d'Hokuyo. Al ser un model comercial molt implementat en robòtica, ja n'existeixen controladors per la plataforma ROS.

Figura 11. Hokuyo URG 04LX

A continuació s'expliquen els passos necessaris per instal·lar els controladors d'aquest sensor. Per començar es descarregaran.

```
$ sudo apt-get install ros-fuerte-laser-drivers
```

Un cop haguem fet això, es connectarà a la font d'alimentació i també la sortida USB a l'ordinador. Llavors s'encendrà una llum verda del sensor que pampallugarà. Serà necessari esperar a que la llum del sensor sigui constant. Això significarà que el sensor està operatiu i es podrà seguir amb la instal·lació. En cas que la llum no s'encengui o no deixi de fer

pampallugues, caldrà revisar la font d'alimentació, ja que aquests senyals indiquen que s'hi produeix algun problema.

S'ha d'assegurar que el node de Hokuyo tingui accés al làser. Per fer-ho s'introduirà la comanda següent.

```
$ ls -l /dev/ttyACM0
```

Amb l'execució de la comanda anterior, s'obtindrà una resposta com aquesta.

```
$ crw-rw-XX- 1 root dialout 166, 0 2009-10-27 14:18 /dev/ttyACM0
```

En cas que XX sigui rw, el làser esta ben configurat. Per altre banda, si XX fos - - , el làser no estarà ben configurat i s'haurà d'introduir la següent comanda, la qual dóna permisos especials al port.

```
$ sudo chmod a+rw /dev/ttyACM0
```

Un cop fet això i després d'inicialitzar el sistema operatiu ROS amb la comanda roscore, s'introduiran els paràmetres de configuració del temps de calibrat i del port d'entrada del node.

```
$ rosparam set hokuyo_node/calibrate_time false
```

```
$ rosparam set hokuyo_node/port /dev/ttyACM0
```

Aleshores ja es podrà executar el node.

```
$ rosrun hokuyo_node hokuyo_node
```

Si tot el procés s'ha fet correctament, s'obtindrà una resposta com la següent.

```
[ INFO] 1256687975.743438000: Connected to device with ID:H0807344
```

El gràfic següent mostra com està funcionant internament tot el conjunt. El node de Hokuyo amb els seus paràmetres publica diferents tòpics, a un dels quals esta subscrit el programa de visualització Rviz.

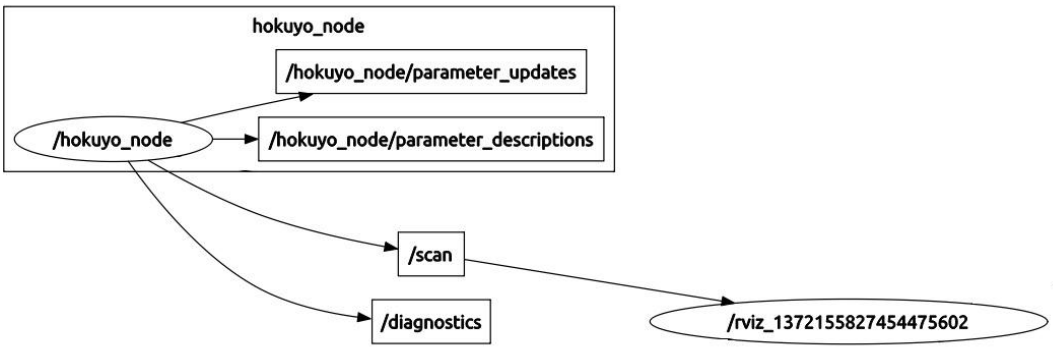

Figura 12. Grafcet node Hokuyo

Si s'executa el programa Rviz i s'inicialitza un node que escolti el tòpic scan que publica el node de Hokuyo, es mostrarà el que està veient el sensor. El resultat serà el que es mostra a continuació.

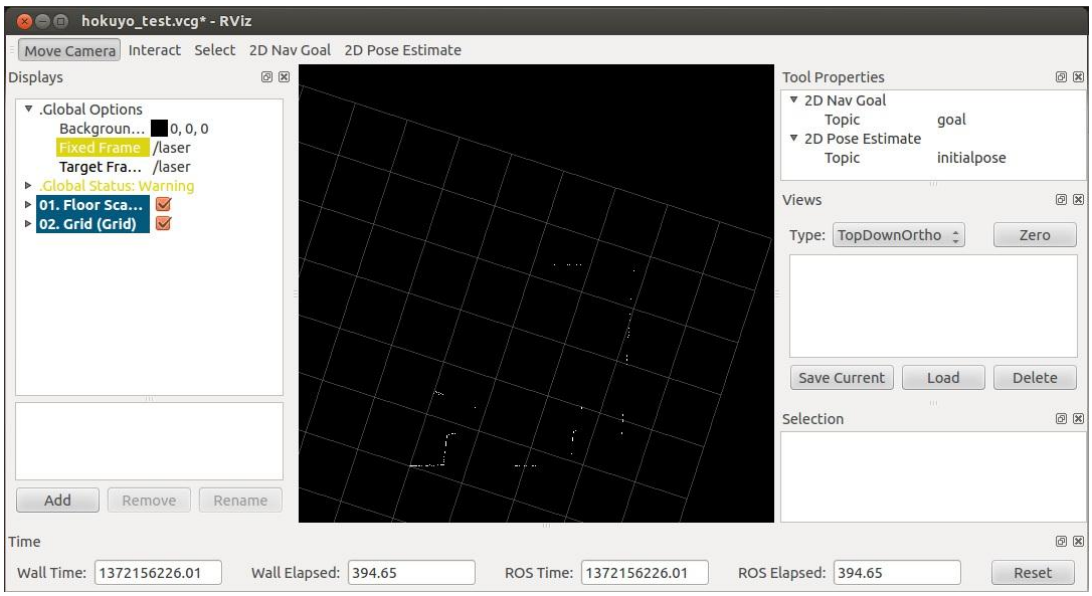

Figura 13. Resultat làser

5.2. IMPLEMENTACIÓ DE LA CÀMERA RGB-D

Una càmera RGB-D és un aparell que, a més de captar imatges, també proporciona informació sobre profunditat, permetent així poder crear una visió en 3D. Això ho aconsegueix combinant una càmera RGB amb un sensor làser que fa un escombrat de la

zona que té al davant. Aquest aparell normalment permet mesurar distàncies des de 80 cm. fins a 4 m. Per tant, la imatge 3D que es pot generar és de la zona immediata.

La càmera seleccionada que s'implementarà amb el nou sistema operatiu no és la que tenia anteriorment el BigBot. En aquest cas, s'optarà per canviar una càmera ethernet per una càmera Kinec de Microsoft. Aquesta, a banda d'oferir la captació d'imatges, també capta profunditat amb un sensor làser d'infrarojos que porta incorporat.

Figura 14. Kinect

Al ser un model de càmera molt utilitzat en el món de la robòtica mòbil, no és difícil trobar un controlador per ROS.

```
$ sudo apt-get install ros-fuerte-openni-camera ros-fuerte-openni-launch
```

Un cop instal·lat el controlador, s'assegurarà que la càmera estigui alimentada i que el cable USB estigui endollat a l'ordinador. S'executarà la comanda d'inicialització del programa de control de la càmera (cal recordar que la comanda roscore ha d'estar prèviament executada.)

```
$ roslaunch openni_launch openni.launch
```

Quan s'hagi fet això, es podran visualitzar els resultats a l'eina de visualització Rviz.

```
$ rosrun rviz rviz
```

Havent arribat en aquest punt, caldrà afegir una nova capa i seleccionar el o els tòpics desitjats. La càmera en publica un gran nombre, entre els quals n'hi ha de profunditat o d'imatge, com es pot observar a les imatges següents.

El tòpic de profunditat pot ser interessant alhora de programar el robot en el tema navegació en l'espai per tal que sigui conscient de la distància que guarda amb l'entorn, ja que fa una representació dels objectes que té al voltant a escala de diferents colors.

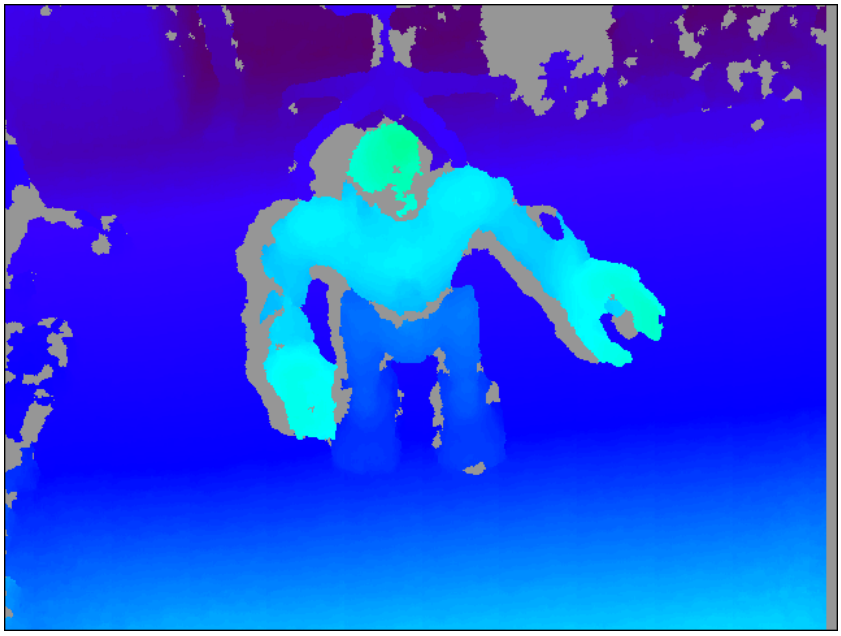

Figura 15. Imatge de profunditat

La imatge en rgb pot ser necessària per programes de reconeixement d'imatges, com és el cas d'un dels altres projectes que ja existeix del BigBot, com és el seguiment d'un gos a través de la càmera.

Figura 16. Imatge en rgb

Pot ser interessant combinar les dues imatges anteriors. D'aquesta manera el programa podrà fer imatges 3D amb percepció de la profunditat. Això serà molt útil per l'ull humà a l'hora d'analitzar les imatges que capta el robot.

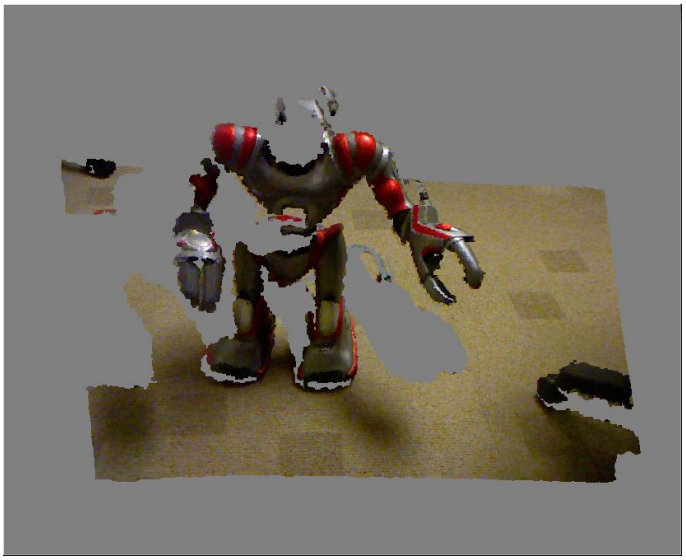

Figura 17. Imatge en 3D

També es pot mostrar la imatge en 3D amb escala de colors per saber amb més precisió la profunditat del que capta la càmera.

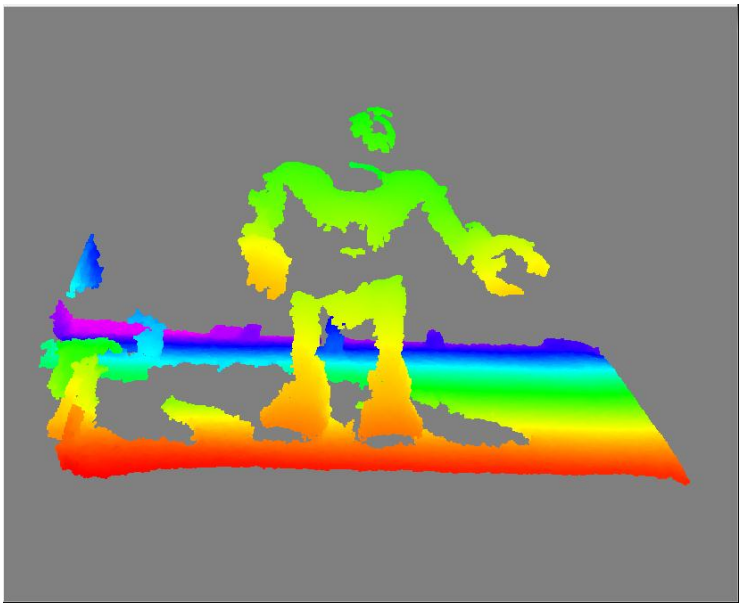

Figura 18.imatge 3D amb distancia

5.3. IMPLEMENTACIÓ DEL SENSOR INERCIAL

Un sensor inercial és un aparell que calcula posició, acceleració i velocitat angular. S'utilitza per captura i anàlisis de moviments. Està compost per acceleròmetres, giroscopis i magnetòmetres. Els acceleròmetres calculen l'acceleració lineal amb què és mou el sensor. Els giroscopis, al seu torn, la velocitat angular. Per últim, els magnetòmetres donen informació sobre on està ubicat el nord magnètic respecte el sensor, podent conèixer així la seva orientació en els tres eixos. Amb aquests tres sensors es possible estudiar el moviment del sensor inercial complet en l'espai.

El sensor inercial que s'utilitzarà és el 3DM-GX1 de MicroStrain. Aquest combina tres giroscopis angulars, tres acceleròmetres ortogonals DC, tres magnetòmetres ortogonals, multiplexors, conversor A/D de 16 bits i el microcontrolador integrat, per donar sortida a la seva orientació en entorns dinàmics i estàtics.

Figura 19. MicroStrain 3DM GX1

A l'intentar implementar aquest sensor no es va trobar un controlador per al ROS. Al connectar-lo a l'ordinador, l'Ubuntu no el reconeixia i per aquest motiu es va haver d'optar per una solució diferent a les anteriors, més complexa, però igualment vàlida i funcional. Encara que no hi hagi un controlador, al ser un programari lliure i obert, se'n pot crear un de propi, sempre i quan es tingui els coneixements suficients de C++. Aquest és un dels avantatges del ROS.

La solució per la que s'ha optat, ha estat aprofitar part de l'estructura original del BigBot, no eliminar el switch i seguir comunicant aquest sensor via ethernet. Per aconseguir això, ha estat necessari crear un node independent amb un programa que, a través d'un socket, es comunicarà amb el sensor inercial a la corresponent IP i tractarà les dades.

Per Fer això és va haver de crear un Pakage de ROS de la següent manera.

Per començar, s'obrirà una nova terminal i s'introduirà el codi següent per indicar la ubicació on volem el paquet.

```
$ cd ~/ros_workspace
```

A continuació es crearà el paquet.

```
$ roscreate-pkg Imu std_msgs rospy roscpp
```

Això crearà una carpeta amb diferents arxius i subcarpetes a la ubicació que s'ha assenyalat anteriorment.

Dins aquesta carpeta s'ha de modificar l'arxiu CMakeLists.txt i afegir-hi la línia següent.

```
rosbuild_add_executable(Imu src/Imu.cpp)
```

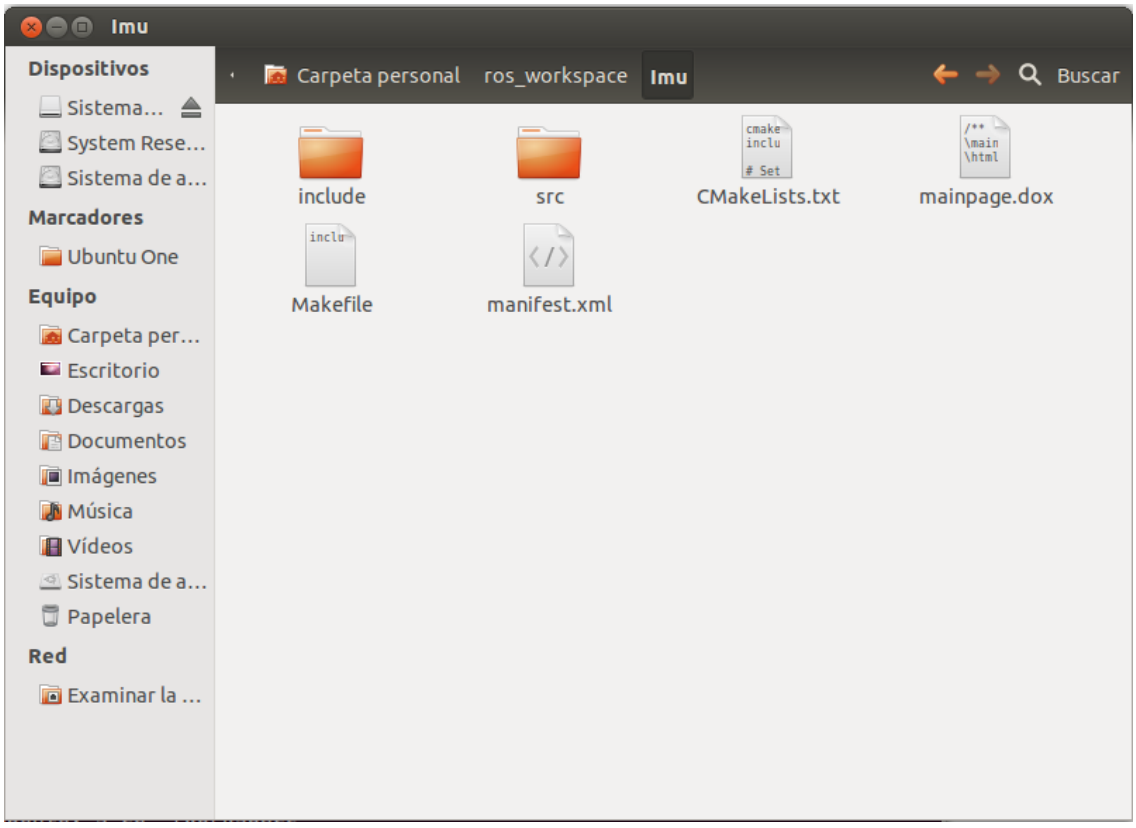

Figura 20. Carpeta Imu

Ara cal crear dins la subcarpeta /src un arxiu .cpp amb el codi que realitzarà el control i la comunicació amb el sensor (adjuntat a l'annex). L'organigrama del qual es mostra a continuació.

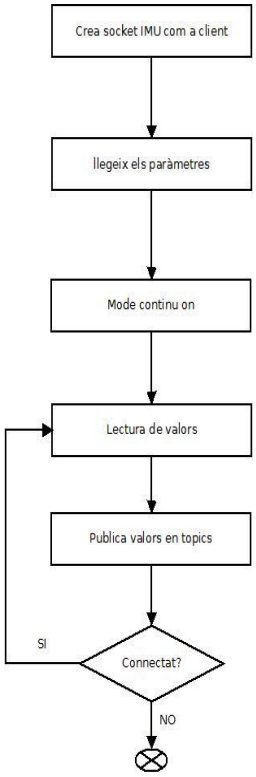

Figura 21. Organigrama Imu

Per últim s'haurà de compilar el programa. En cas que no compilés, ens informaria de l'error a la mateix finestra de la terminal. D'altra banda, si compila, estarà preparat per executar-se.

```
$ cd ~/ros_workspace/Imu
```

```
$ make
```

Al gràfcet següent es mostra l'estructura del node del sensor inercial. Com es pot veure, aquest publica 3 missatges, Roll, Pitch i Yaw.

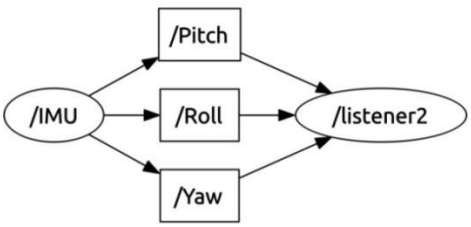

Figura 22. Gràfcet Imu

Aquests tres tòpics diferents que es mostren a la imatge anterior corresponen als tres angles que es necessiten per conèixer l'orientació exacta del robot.

Com es veu en el gràfcet, també s'ha creat un node anomenat listener2, el qual està subscrit als tòpics que publica l'IMU per poder-ne veure el resultat.

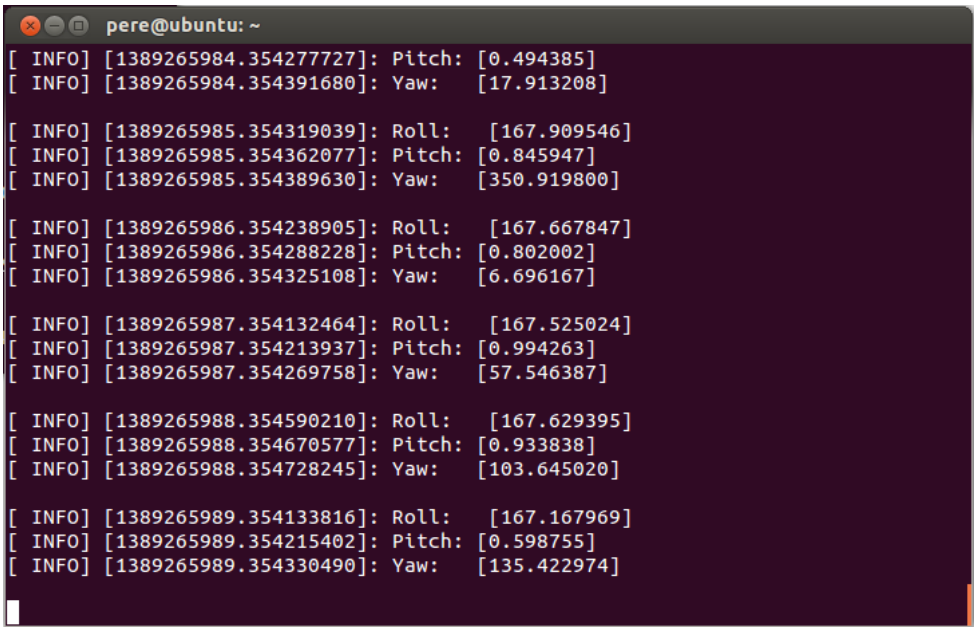

Figura 23. Resultat Node IMU

5.4. IMPLEMENTACIÓ DE LA PLACA DE CONTROL

La placa de control va ser fabricada per la Udg. Disposa d'un DS-Pic per tal de controlar els motors i encóders del robot i alhora processa algunes dades que es retornen a l'ordinador per tal d'alleugerir el nombre de càlculs del processador de l'ordinador.

En aquest cas, igual que en el cas del sensor inercial, al ser una placa creada per la UdG tampoc es poden trobar controladors per ROS. Per tant, també s'ha optat per la solució d'aprofitar l'estructura antiga del robot.

A l'haver de conservar el switch per comunicar-se amb el sensor inercial, també s'obtenen les dades del PIC a través del protocol TCP/IP. És per això que també s'ha creat un paquet com en el cas anterior amb un arxiu C++ en el que es crea un socket per rebre totes les dades que transmeti el DS-PIC.

Aquest programa s'anomena Interlocutor i segueix l'organigrama que es mostra a la figura 24.

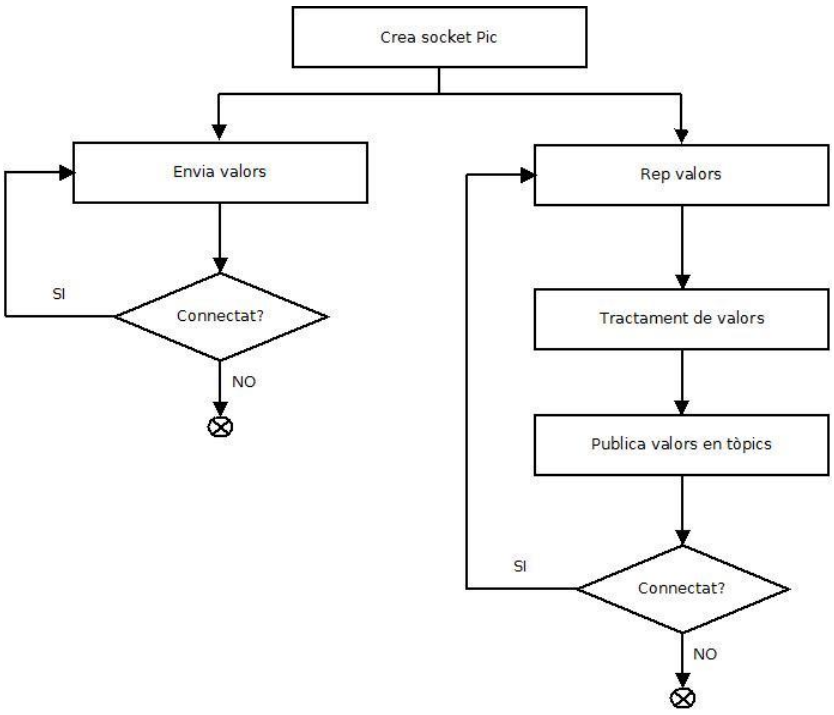

Figura 24. Diagrama Pic

El principal inconvenient que presenta aquest sistema és que la placa només és capaç de transmetre trames de byte. És per això que s'ha de saber en cada posició quina dada hi ha i tractar-la adequadament per tal de treballar amb valors lògics.

En el següent gràfcet es mostren els diferents tòpics que publica l'interlocutor.

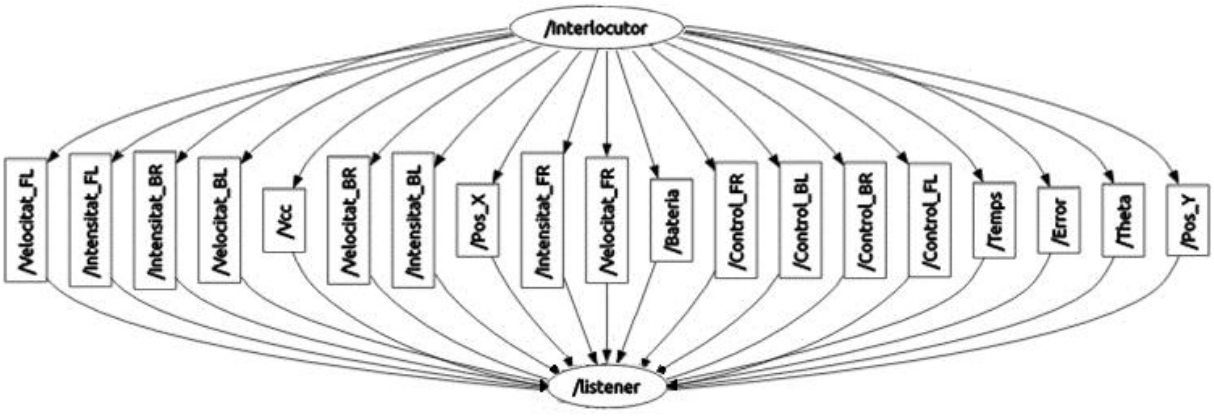

Figura 25. Gràfcet Interlocutor

Com es pot observar a la figura anterior, aquest node publica moltes més dades que les que s'especificaven a l'objectiu. Això és així perquè es va creure interessant mostrar dades com l'estat de la bateria, les intensitats de les rodes, etc. encara que no estiguessin contemplades als objectius.

Per acabar s'ha de tenir en compte que tant el node interlocutor com l'Imu s'han creat per comunicar-se només amb la placa de control i l'Imu de BigBot 3. Al conservar l'enrutador s'ha utilitzat protocol TCP/IP i per tant els nodes parlen amb la IP específica de cada component. Si aquests nodes es vulguessin implementar en altres robots de la flota, s'hauria de canviar el programa perquè parlessin amb les IPs específiques de les plaques de control i dels Imus dels altre robots.

En aquest cas també s'ha creat un node, anomenat listener, per tal de comprovar el bon funcionament del programa interlocutor. El resultat del qual és el següent.

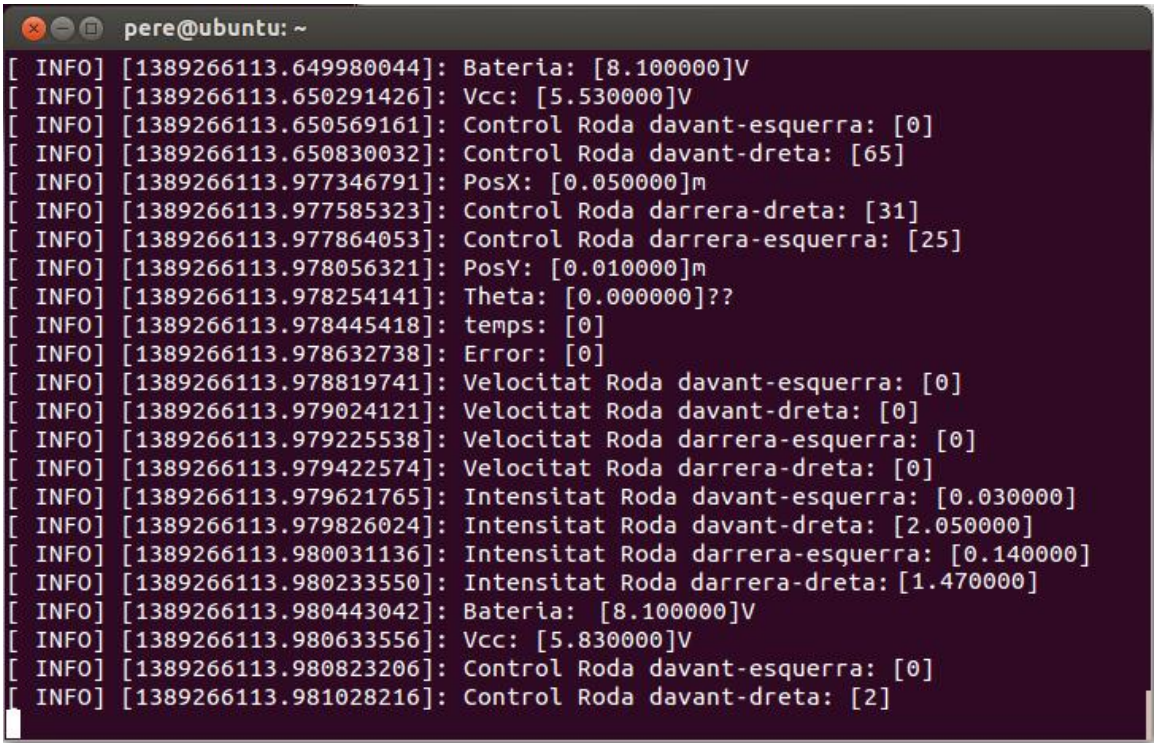

Figura 26. Resultat Node Interlocutor

6. GUIA D'UTILITZACIÓ

Una vegada tots els controladors estan instal·lats, ja es podrà iniciar el sistema. Es partirà del punt en què el ROS no s'està executant i per tant tampoc hi haurà cap node operatiu.

Per començar s'iniciarà el ROS. Per fer-ho serà necessari que l'ordinador estigui encès. S'obrirà una nova terminal i s'executarà la comanda següent.

```
$ roscore
```

Aquesta comanda retornarà un seguit de dades i deixarà la terminal inactiva. En qualsevol moment, prement Control+C es pararà l'execució del ROS.

En aquest punt serà necessari activar tots els nodes. Per començar s'activarà la comunicació amb la placa de control, obrint una nova terminal i introduint la comanda següent.

```
$ rosrun Pic interlocutor
```

Això farà que el node comenci a publicar dades sobre l'estat dels motors, encóders, bateries i altres dades provinents de la placa de control, que alhora es mostraran per la terminal.

En una nova terminal, per tal de fer funcionar el sensor inercial, s'introduirà la comanda següent.

```
$ rosrun IMU IMU
```

Com en el cas de l'interlocutor també es publicaran dades i es mostraran per la terminal, però en aquest cas, les dades seran provinents de l'Imu.

Per inicialitzar la Kinect, en una nova terminal, com en tots els casos, s'introduirà la comanda següent perquè el node s'inicialitzi.

```
$ roslaunch openni_launch openni.launch
```

Per últim, s'executarà el node del telèmetre. Primer de tot, en una nova terminal, s'haurà d'indicar el port USB on està conectat el sensor. Tot seguit caldrà configurar uns paràmetres d'inicialització introduint les línies següents.

```
$ ls - /dev/ttyACM0

$ sudo chmod a+rw /dev/ttyACM0

$ rosparam set hokuyo_node/calibrate_time false

$ rosparam set hokuyo_node/port/dev/ttyACM0
```

Un cop fet això ja es podrà executar el node.

```
$ rosrun hokuyo_node hokuyo_node
```

Havent executat totes aquestes línies, ja serà operatiu tot el sistema implementat en ROS.

Cal tenir en compte que es podria penjar algun dels nodes o fins i tot podria ser necessari aturar-ne algun. En aquest cas, a la mateixa terminal s'haurien de prémer les tecles Control+C. Si això no funcionés, s'hauria de parar el ROS des de la primera terminal oberta, utilitzant també les tecles Control+C i després tornant a començar el procés seguint els passos d'aquesta guia.

7. RESUM DEL PRESSUPOST

El cost d'aquest projecte, tenint en compte el material i la mà d'obra, serà de sis mil dos-cents setanta-un amb setanta-cinc cèntims, preu sense IVA.

8. CONCLUSIONS

Un cop realitzada aquesta implementació, es pot observar que les especificacions s'han complert en la seva totalitat.

Per començar, s'ha implementat el sistema operatiu ROS en un ordinador que es connecta al BigBot, que era l'objectiu primordial. A part, també s'han implementat els sensors Laser Range Finder (Hokuyo URG 04LX), la càmera RGB-D (Microsoft Kinect) i el sensor inercial (MicroStrain 3DM-GX1) que s'especificaven als objectius.

També s'han controlat tant els motors a través de la placa de control fabricada per la UdG, com els encóders, tot i que queda pendent tractar-ne les dades per obtenir valors útils, com poden ser distància, posició, etc.

A més, s'ha aconseguit llegir un seguit d'informacions a través de la placa de control com el nivell de bateria, el corrent dels motors, etc. que no sortia a les especificacions inicials.

Cal notar també les dificultats que s'han trobat. Primer és necessari destacar que per molt que el ROS estigui pensat perquè els usuaris comparteixin els seus coneixements i que hi hagi tutorials que expliquin com instal·lar els nodes, sovint apareixen problemes amb el hardware per diferents motius. A vegades pot ser per l'alimentació, perquè el tutorial no especifica com obtenir les dades pel port desitjat, o simplement perquè l'ordinador no reconeix el sensor. També cal remarcar que aquest document és de temàtica informàtica i que això ha fet que, abans de la realització, s'hagués de fer una recerca i aprofundiment en temes que no s'havien treballat durant la carrera. Per últim, el ROS, al ser un projecte considerablement nou i al no ser conegut a la universitat, s'ha hagut de contactar amb persones que treballen amb aquest sistema operatiu, fora de la universitat, per tal d'aconseguir tenir unes primeres nocions.

Resta assenyalar el treball futur. La implementació del ROS només és el principi, el següent pas seria trobar una plataforma que el pogués suportar i fos mòbil, per tal d'incorporar-lo dins l'estructura del robot. Una altra feina que s'està duent a terme és la implementació dels sensors secundaris i la comunicació del robot a través de wifi amb una altre terminal de ROS instal·lada en un altre ordinador, per tal de poder visualitzar les dades i enviar senyals de control. Més endavant també seria interessant desenvolupar un software que dotés el robot de total autonomia i d'aquesta manera no necessités control humà.

Un altre aspecte que es podria considerar és la actualització de la versió de ROS. Al ser un sistema operatiu robòtic i la robòtica, al seu torn, ser una tecnologia puntera, aquest sistema evoluciona molt ràpidament. A data de gener de 2014 ja existeixen dos versions més recents, Groovy Galapagos i Hydro Medusa, i ja s'està treballant en una nova versió, Indigo Igloo, el llançament de la qual està previst per l'abril de 2014. Cal recordar que es va escollir la versió Fuerte perquè a un altre laboratori de recerca de la UdG ja es treballava amb aquesta plataforma. Amb la intenció d'aprofitar la seva experiència es va decidir començar per aquesta versió, però ara que ja hi ha experiència en ROS a la UdG, és podria decidir anar més enllà i actualitzar el sistema operatiu a una versió més moderna.

La conclusió final és que tot i que s'han complert les especificacions, es poden realitzar moltes millores pel que fa al robot i al seu control.

Pere Vila Soler
Graduat en Enginyeria Electrònica Industrial i Automàtica.

Girona, 23 de juliol de 2013

9. RELACIÓ DE DOCUMENTS

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

10. BIBLIOGRAFIA

3DM-GX1, LORD MicroStrain, (http://www.microstrain.com/inertial/3DM-GX1, 17 de juliol de 2013)

Hokuyo URG-04LX, Acroname, (http://www.acroname.com/products/R283-HOKUYO-LASER1.html, 14 de gener de 2014)

Learning ROS for Robotics Programming, Las Palmas, Espanya, 2013

McGraw-Hill, TCP/IP: Arquitectura, Protocolos e Implementacion, Madrid, Espanya, 20004

OpenNI, OpenNI Org., (http://www.openni.org/, 15 d'agost de 2013)

ROS, Ros, (http://www.ros.org/wiki/, 23 de juliol de 2013)

ROS INSTALLATION, Nootrix, (http://nootrix.com/2012/05/ros-installation/, 23 de juliol de 2013)

UdG, Flota De Robots De Rescat, Girona, Espanya, 2006

UdG, Resum de Hardware, Girona, Espanya, 2006

11. GLOSARI

IMU (Inertial Measurement Unit, en català, unitat de mesura inercial)

LAN (Local Area Network, en català, xarxa d'area local)

PC (Personal Computer, en català, ordinador personal)

PIC (Peripheral Interface Controller, en català, controlador d'interfaç periféric)

RGB (Red Gren Blue, en català, vermell verd blau)

ROS (Robotic Operational System, en català, sistema operatiu robotic)

TCP/IP (Transmision Control Protocol/ Internet Protocol, en català, protocol de control de transmissió/ Protocol d'internet)

USB (Universal Serial Bus, en català, bus universal en sèrie)

Wifi (Wireless Fidelity, en català, fidelitat inalambrica)

A. CODI INTERLOCUTOR

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Int8.h"
#include "std_msgs/Int16.h"
#include "std_msgs/Float32.h"
#include "std_msgs/Float64.h"
//#include "std_msgs/Uint8.h"
#include "ihm.h"

//INCLUDES DE TCPSOCK.C, UDPSOCK.C I SOCKBACE.C
#include "SockBase.h"
#include "UDPSock.h"
#include <stdio.h>
#include "SockBase.h"

//INCLUDE DE ERRNO
#include <errno.h>
#include <sstream>

/** AQUEST PROGRAMA FARA DE PONT ENTRE ELS SENSORS I ACTUADORS,
* i ELS PROGRAMES PER CONTROLAR EL ROBOT
*/


//Declaracio de Variables globals

int errno;
int hola = 10;

char * addrRobot;
char * addrPIC;
char * addrPc;
char * addrIMU;

int PC_send_period=1000; //Must be a multiple of 5ms
int rcv_count=0; //Counter for number of paquets recieved from the dsPIC (5ms
counter)
int send_PIC=0; //Flag to start transmision to the dsPIC
int sendTimer=0; //Flag per controlar que el thread de sendPIC no envii massa rapid
unsigned int real_time=0; //Acumulated robot time
int nBytesInt = 0; //Variable per controlar els bytes a enviar cap al PIC

int PC_send_bytes=0; //Number of bytes to be send to the PC
int Buf_PC_send_bytes[5];
int indexRebre = 0;
unsigned char bufModeCom[3];

//Definim configuraci socket per PC
static int servSock=-1; /* Socket descriptor for server */
static int clntSock=-1; /* Socket descriptor for client */
static struct sockaddr_in echoServAddr; /* Local address */
static struct sockaddr_in echoClntAddr; /* Client address */
static unsigned int clntLen; /* Length of client address data structure */
static unsigned short echoServPort=15000; //Server port
#define MAXPENDING 5

int sockPIC;
int sockIMU;
int start_send=0;

// Threads
void * recvPIC(void *num);
void * recvPC(void *num);
void * sendPC(void *num);
void * sendPIC(void *num);
```

```
void * recvIMU(void *num);

void desglosa_PIC(void);
int composa_PC(int index);
void startSocketPIC(void);
void stopSocketPIC(void);
void startSocketIMU(void);
void gestio_int (int numero);
int calculaPaquet_PIC(void);
void calculaVarIMU(void);

//Variables utilitzades per convertir els valors dels bufUp's a un format que es
pugui publicar
//Aixo es degut a que el ROS no es capas de publicar unsigned char*

float Vtemsp = 0;
float Vcodi_err = 0;

int VVel_FL= 0;
int VVel_FR= 0;
int VVel_BL= 0;
int VVel_BR= 0;
int VInt_FL = 0;
int VInt_FR = 0;
int VInt_BL = 0;
int VInt_BR = 0;

char VBat = 0;

char VVcc = 0;

int Scont_FL= 0;
int Scont_FR= 0;
int Scont_BL= 0;
int Scont_BR= 0;
int VPos_X = 0;
int VPos_Y = 0;
int VTheta =0;
double dVVel_FL= 0;
double dVVel_FR= 0;
double dVVel_BL= 0;
double dVVel_BR= 0;
double dVInt_FL = 0;
double dVInt_FR = 0;
double dVInt_BL = 0;
double dVInt_BR = 0;

double dVBat = 0;

double dVVcc = 0;

int dScont_FL= 0;
int dScont_FR= 0;
int dScont_BL= 0;
int dScont_BR= 0;

double dPos_X = 0;
double dPos_Y = 0;
double dTheta = 0;




int main(int argc, char **argv)
{
```

```
/**
The ros::init() function needs to see argc and argv so that it can perform
* any ROS arguments and name remapping that were provided at the command line. For
programmatic
* remappings you can use a different version of init() which takes remappings
* directly, but for most command-line programs, passing argc and argv is the
easiest
* way to do it. The third argument to init() is the name of the node.
*
* You must call one of the versions of ros::init() before using any other
* part of the ROS system.
*/
ros::init(argc, argv, "Interlocutor");

/**
* NodeHandle is the main access point to communications with the ROS system.
* The first NodeHandle constructed will fully initialize this node, and the last
* NodeHandle destructed will close down the node.
*/
ros::NodeHandle n;

//DEFINIM ELS MISATGES QUE ES PUBLICARAN

ros::Publisher chattor_pub = n.advertise<std_msgs::String>("chattor", 1000); //Es
van utilitzar per fer proves
//ros::Publisher chatter_pub = n.advertise<std_msgs::Int8>("hola",1000); //aquesta
tambe
ros::Publisher temps_pub = n.advertise<std_msgs::Int8>("Temps",1000);
ros::Publisher codi_error_pub = n.advertise<std_msgs::Int8>("Error",1000);

/**
*Topics de Velocitats de les rodes FX significa roda davantera (front)
*BX significa roda trasera (Back), i XL i XR signifiquen roda left i
*right respectivament
*/

ros::Publisher Vel_FL_pub = n.advertise<std_msgs::Float32>("Velocitat_FL",1000);
ros::Publisher Vel_FR_pub = n.advertise<std_msgs::Float32>("Velocitat_FR",1000);
ros::Publisher Vel_BL_pub = n.advertise<std_msgs::Float32>("Velocitat_BL",1000);
ros::Publisher Vel_BR_pub = n.advertise<std_msgs::Float32>("Velocitat_BR",1000);

/**
*Topics de Bateria I Vcc
*/

ros::Publisher Bat_pub = n.advertise<std_msgs::Float64>("Bateria",1000);
ros::Publisher Vcc_pub = n.advertise<std_msgs::Float64>("Vcc",1000);

/**
*Topics de Intensitats de les rodes, les abreviacions funcionen igual
*que les consignes de Velocitat
*/

ros::Publisher Int_FL_pub = n.advertise<std_msgs::Float64>("Intensitat_FL",1000);
ros::Publisher Int_FR_pub = n.advertise<std_msgs::Float64>("Intensitat_FR",1000);
ros::Publisher Int_BL_pub = n.advertise<std_msgs::Float64>("Intensitat_BL",1000);
ros::Publisher Int_BR_pub = n.advertise<std_msgs::Float64>("Intensitat_BR",1000);

/**
*Topics de tensio de control
*/

ros::Publisher Cont_FL_pub = n.advertise<std_msgs::Int8>("Control_FL",1000);
ros::Publisher Cont_FR_pub = n.advertise<std_msgs::Int8>("Control_FR",1000);
ros::Publisher Cont_BL_pub = n.advertise<std_msgs::Int8>("Control_BL",1000);
ros::Publisher Cont_BR_pub = n.advertise<std_msgs::Int8>("Control_BR",1000);

/**
```

```
*Topics de posicio
*/

ros::Publisher PosX_pub = n.advertise<std_msgs::Float32>("Pos_Y",1000);
ros::Publisher PosY_pub = n.advertise<std_msgs::Float32>("Pos_X",1000);
ros::Publisher Theta_pub = n.advertise<std_msgs::Float32>("Theta",1000);



ros::Rate loop_rate(3);

/**
* Definicio de variables que agafaran valors
* Totes les variables portaran a davant una X davant per saber que son variables i
no confondreles amb els std_msgs que aniran asociats
*/
int hola = 10;
int count=0;



while (ros::ok())
{



/**
* This are a message object. You stuff it with data, and then publish it.
*
*Definim els objectes misatge
*
*/
std_msgs::String msg;
std_msgs::Float32 temps;
std_msgs::Float32 codi_err;
std_msgs::Float32 Vel_FL;
std_msgs::Float32 Vel_FR;
std_msgs::Float32 Vel_BL;
std_msgs::Float32 Vel_BR;

std_msgs::Float64 Int_FL;
std_msgs::Float64 Int_FR;
std_msgs::Float64 Int_BL;
std_msgs::Float64 Int_BR;

std_msgs::Float64 Bat;
std_msgs::Float64 Vcc;

std_msgs::Int8 Cont_FL;
std_msgs::Int8 Cont_FR;
std_msgs::Int8 Cont_BL;
std_msgs::Int8 Cont_BR;

std_msgs::Float32 Pos_X;
std_msgs::Float32 Pos_Y;
std_msgs::Float32 Theta;


/**
* $$$$$ !!!!EL PROGRAMA ANIRA AQUI!!! $$$$$$
*
*/

int resultatRecv=1;

struct itimerval val; // Variable per la configuraci? del timer
struct sigaction interrupcio; // Variable per la configuraci? d'interrupcions
```

```
struct timeval timeoutSock; //Variable per controlar el temps d'espera dels
sockets.

//0x88 = Possem a 1 el bit de més pes per indicar que es un paquet amb el Yaw,
//i l'altre 1 indica que es correció d'angle
PIC_sendBuf_minim[0] = 0x88;

if ((1)){
addrPIC = "192.168.1.18";
printf("\nX\n");
printf("IP PIC: %s\n",addrPIC);
//iniciem el buffer down
int k=0;
int h=0;
for (k=0; k<=39; k++){
bufDown[k]=0;
}
for (h=0; h<=4; h++){
for (k=0; k<=39; k++){
bufUp[h][k]=0;
}
}
}
else{
printf("\n**ERROR**\n\nComanda: ./comunicacio IP_dspic periode\n\n");
exit(0);
}


startSocketPIC();
if(sockPIC == -1){
printf("\nError al crear el socket del PIC\n");
}else{ //Netejem les possibles dades que hi puguin haver al dsPIC
timeoutSock.tv_sec=0;
timeoutSock.tv_usec=750000;

setsockopt(sockPIC, SOL_SOCKET, SO_RCVTIMEO, &timeoutSock, sizeof(timeoutSock));

printf("\nNetejant socket dsPIC...");
while(recv(sockPIC, &bufUp[0][0], 1, 0)>=0){
//printf("rebent %d\n", bufUp[0][0]);
}
//printf(" Ok !\n");

}

// CREACIO DELS THREADS
if (pthread_create (&recvPICTh, NULL, recvPIC, (void *) 1) != 0){
printf("\nError al crear el thread de recvPIC\n");
}
if (pthread_create (&sendPICTh, NULL, sendPIC, (void *) 1) != 0){
printf("\nError al crear el thread de sendPIC\n");
}

pthread_mutex_init (&mutex_nBufEnviar, NULL);

val.it_interval.tv_sec=0;
val.it_interval.tv_usec=15000;
val.it_value.tv_sec=0;
val.it_value.tv_usec=15000;
setitimer(ITIMER_REAL, &val, NULL);
//printf ("period=%d\n",PC_send_period);

//Interrupci timer
interrupcio.sa_handler = gestio_int; // El manejador de de les senyals capturades
es gestio_int
sigemptyset (& (interrupcio . sa_mask)); // Especifiquem que no es bloquejar cap
senyal
```

```
interrupcio.sa_flags = 0; // Tots els flags desactivats
sigaction(SIGALRM, &interrupcio, NULL);
sigaction(SIGINT, &interrupcio, NULL);

while(1){

bufDown[0]= 0x00;
bufDown[1]= 0xFF;
bufDown[2]= 0x07;
bufDown[3]= 0x03;
bufDown[4]= 0x64;
bufDown[5]= 0x64;
bufDown[6]= 0x05;
bufDown[7]= 0x7f;
bufDown[8]= 0x00;
bufDown[9]= 0x00;
bufDown[10]= 0x00;
bufDown[11]= 0x00;
bufDown[12]= 0x00;
bufDown[13]= 0x00;
bufDown[14]= 0x00;
bufDown[15]= 0x00;
bufDown[16]= 0x00;
bufDown[17]= 0x00;
bufDown[18]= 0x00;
bufDown[19]= 0x00;
bufDown[20]= 0xFF;
bufDown[21]= 0xFF;
bufDown[22]= 0x00;
bufDown[23]= 0x00;
bufDown[24]= 0x00;
bufDown[25]= 0x00;
bufDown[26]= 0x00;
bufDown[27]= 0x00;


//Definicio del misatge exemple
//msg2.data = hola ;


std::stringstream ss;

ss << "hello world " << count;
msg.data = ss.str();


ROS_INFO("%s", msg.data.c_str());
chattor_pub.publish(msg);
//Aqui s'acaba el misatge exemple

//Assignem les variables als misatges
temps.data= Vtemsp;
codi_err.data = Vcodi_err;


Bat.data=dVBat;
Vcc.data=dVVcc;
Vel_FL.data=VVel_FL;
Vel_FL.data=VVel_FR;
Vel_FL.data=VVel_BL;
Vel_FL.data=VVel_BR;

Int_FL.data=dVInt_FL;
Int_FR.data=dVInt_FR;
Int_BL.data=dVInt_BL;
Int_BR.data=dVInt_BR;

Cont_FL.data=dScont_FL;
```

```
Cont_FR.data=dScont_FR;
Cont_BL.data=dScont_BL;
Cont_BR.data=dScont_BR;

Pos_X.data=dPos_X;
Pos_Y.data=dPos_Y;
Theta.data=dTheta;



//avisem al master que publicarem als topics
send_PIC=1;
ROS_INFO("%d", temps.data);
ROS_INFO("%d", codi_err.data);
ROS_INFO("%d", Vel_FL.data);
ROS_INFO("%d", Vel_FR.data);
ROS_INFO("%d", Vel_BL.data);
ROS_INFO("%d", Vel_BR.data);
ROS_INFO("%f", Int_FL.data);
ROS_INFO("%f", Int_FR.data);
ROS_INFO("%f", Int_BL.data);
ROS_INFO("%f", Int_BR.data);
ROS_INFO("%f", Bat.data);
ROS_INFO("%f", Vcc.data);
ROS_INFO("%f", Cont_FL.data);
ROS_INFO("%f", Cont_FR.data);
ROS_INFO("%f", Cont_BL.data);
ROS_INFO("%f", Cont_BR.data);
ROS_INFO("%f", Pos_X.data);
ROS_INFO("%f", Pos_Y.data);
ROS_INFO("%f", Theta.data);


//Publiquem els misatges


temps_pub.publish(temps);
codi_error_pub.publish(codi_err);
Vel_FL_pub.publish(Vel_FL);
Vel_FR_pub.publish(Vel_FR);
Vel_BL_pub.publish(Vel_BL);
Vel_BR_pub.publish(Vel_BR);
Int_FL_pub.publish(Int_FL);
Int_FR_pub.publish(Int_FR);
Int_BL_pub.publish(Int_BL);
Int_BR_pub.publish(Int_BR);
Bat_pub.publish(Bat);
Vcc_pub.publish(Vcc);
Cont_FL_pub.publish(Cont_FL);
Cont_FR_pub.publish(Cont_FR);
Cont_BL_pub.publish(Cont_BL);
Cont_BR_pub.publish(Cont_BR);
PosX_pub.publish(Pos_X);
PosY_pub.publish(Pos_Y);
Theta_pub.publish(Theta);



/**
* The publish() function is how you send messages. The parametr
* is the message object. The type of this object must agree with the type
* given as a template parameter to the advertise<>() call, as was done
* in the constructor above.
*/

// chatter_pub.publish(msg2);

ros::spinOnce();
```

```
loop_rate.sleep();
++count;
}

return 0;
}


}



// THREAD REBRE DEL dsPIC
void * recvPIC (void * num){
//Definim la variable que anira movent el punter en les diferents posicions de
bufUp[index][]
int index=0;

unsigned short aux1;
unsigned short aux2;
unsigned short aux3;
unsigned short aux4;

while(1){
//Obtenim la part del paquet que es fixe!
recv(sockPIC, &bufUp[index][0], 5, MSG_WAITALL);
//printf("ADEUPERE%x,                                %x,                                %x,
%x\n",bufUp[index][0],bufUp[index][1],bufUp[index][2],bufUp[index][3]);
//Obtenim pas per pas la part variable
if(fVelocitat)
{ (sockPIC, &bufUp[index][5], 4, MSG_WAITALL);
aux1 = bufUp[index][5];
aux2 = bufUp[index][6];
aux3 = bufUp[index][7];
aux4 = bufUp[index][8];

VVel_FL= aux1;
VVel_FR= aux2;
VVel_BL= aux3;
VVel_BR= aux4;
//printf("\n\nVelocitats: %f, %f, %f, %f\n",VVel_FL,VVel_FR,VVel_BL,VVel_BR);
//printf("ADEU2%d, %x, %s, %s\n",velocitatFL,velocitatFR,velocitatRL,velocitatRR);

}
if(fBattery)
{
recv(sockPIC, &bufUp[index][9], 2, MSG_WAITALL);
aux1 = bufUp[index][9];
aux2 = bufUp[index][10];

if(aux1!=0)
{dVBat=(double)aux1/10;}
if(aux2!=0)
{dVVcc=((double)aux2/100)+3.5;}
//printf("Bat: %f, %f,\n",dVBat,dVVcc);
//printf("Bat: %i, %i,\n",bufUp[index][9],bufUp[index][10]);
}
if(fPosition)
{ recv(sockPIC, &bufUp[index][11], 5, MSG_WAITALL);


aux1 = bufUp[index][13];
aux1 = aux1 < 8;
aux2 = bufUp[index][11];
VPos_X = aux1|aux2;
dPos_X= (double)VPos_X/100;
```

```
aux1 = bufUp[index][14];
aux1 = aux1 < 8;
aux2 = bufUp[index][12];
VPos_Y = aux1|aux2;
dPos_Y= (double)VPos_Y/100;

aux1 = bufUp[index][15];
dTheta = (aux1*360)/255;

//printf("Pos:                    %f,                    %f,                    %d,
%d,%d\n",bufUp[index][11],bufUp[index][12],bufUp[index][13],bufUp[index][14],bufUp[
index][15]);
}
if(fCurrent)
{ recv(sockPIC, &bufUp[index][16], 8, MSG_WAITALL);
aux1 = l_Int_FL;
aux1 = aux1 < 8;
aux2 = h_Int_FL;
VInt_FL = aux1|aux2;
dVInt_FL = (double)VInt_FL/100;

aux1 = l_Int_FR;
aux1 = aux1 < 8;
aux2 = h_Int_FR;
VInt_FR = aux1|aux2;
dVInt_FR = (double)VInt_FR/100;

aux1 = l_Int_RL;
aux1 = aux1 < 8;
aux2 = h_Int_RL;
VInt_BL = aux1|aux2;
dVInt_BL = (double)VInt_BL/100;

aux1 = l_Int_RR;
aux1 = aux1 < 8;
aux2 = h_Int_RR;
VInt_BR = aux1|aux2;
dVInt_BR = (double)VInt_BR/100;

//printf("Corrents: %f, %f, %i, %f,\n",VInt_FL, VInt_FR, VInt_BL, VInt_BR);
}
if(fCtlSig)
{ recv(sockPIC, &bufUp[index][24], 4, MSG_WAITALL);

aux1 = bufUp[index][24];
aux2 = bufUp[index][25];
aux3 = bufUp[index][26];
aux4 = bufUp[index][27];

Scont_FL= aux1;
Scont_FR= aux2;
Scont_BL= aux3;
Scont_BR= aux4;

//printf("Senyals  de  Control:  %d,  %d,  %d,  %d\n",Scont_FL,  Scont_FR,  Scont_BL,
Scont_BR);
}
if(fDiffTick) recv(sockPIC, &bufUp[index][28], 2, MSG_WAITALL);
if(fSensors) recv(sockPIC, &bufUp[index][30], 4, MSG_WAITALL);
if(fEncoders) recv(sockPIC, &bufUp[index][34], 4, MSG_WAITALL);

//printf("HOLA!:)\n%d\n",Flags_PICtoPC);
//printf("HOLA2!:)\n%d\n",peticioFlags);
rcv_count++;
if (rcv_count>=PC_send_period && clntSock != -1){
pthread_mutex_lock(&mutex_nBufEnviar);
nBufEnviar++;
```

```
pthread_mutex_unlock(&mutex_nBufEnviar);
index++;
if(index>4) index = 0;
//Preparem la variable indexRebre, que apunta a la penultima posició més actual
indexRebre = index-1;
rcv_count = 0;
}
}
}

void * sendPIC (void * num){
//while(!start_send);
while(1){sendTimer = 1;
if(sendTimer == 1){
if (send_PIC==1){
if (yaw_to_PIC){
Flags_PCtoPIC = Flags_PCtoPIC|0x08;
h_corr_t = h_yaw;
l_corr_t = l_yaw;
}
//Enviem al PIC la part fixa del paquet
send(sockPIC, &bufDown[0], 8, 0);
//Enviem psd per pas la part variable
if (fPosOrder) send(sockPIC, &bufDown[8],4,0);
if (fTOrder) send(sockPIC, &bufDown[12],1,0);
if (fCorrPos) send(sockPIC, &bufDown[13],4,0);
if (fCorrT) send(sockPIC, &bufDown[17],2,0);
if (fErrAd) send(sockPIC, &bufDown[19],1,0);
if (fConfPID) send(sockPIC, &bufDown[20],6,0);
//printf("Jo estic enviant!\n%d\n",fPosOrder);
send_PIC=0;
}
else if(yaw_to_PIC == 1){
//Enviem la capçalera del paquet "mini"
send(sockPIC, &PIC_sendBuf_minim[0], 1, 0);
h_corr_t = h_yaw;
l_corr_t = l_yaw;
send(sockPIC, &bufDown[17], 2, 0);
}
sendTimer = 0;
}
}

}


void startSocketPIC(){
sockPIC = CreateTCPClient(addrPIC,10001);
printf("\n entra al startsocketPIC %d %s", sockPIC, addrPIC );
}

void stopSocketPIC(){
CloseSock(sockPIC);
}

// Gesti□ de les senyals capturades
void gestio_int (int numero)
{
int resultatRecv = 0;
int i;
char final1=0, final2=0, final3=0;

switch (numero)
{
case SIGALRM :
//Flag que controla quan enviar al PIC per conservar l'ample de banda
sendTimer = 1;
```

```
break;

case SIGINT:
/*******************************************
Codi executat per la interrupció de Ctrl+C. A part, està configurat per que quan
Labview es desconecti
salti aquesta interrupció.

Aquesta part ens assegura que la transmissió de l'IMU es "tanca" correctament i que
no queda res
al buffer.
*******************************************/
printf("\nS'ha pulsat Ctrl+C!\n");

printf("\nFinalitzant comunicacio amb el PIC...");
/*//Configurem el buffer per deixar de transmetre en continuu
bufModeCom[0] = 0x10;
bufModeCom[1] = 0x00;
bufModeCom[2] = 0x00;
//Enviem els 3 bytes
resultatRecv = send(sockIMU, bufModeCom, 3, 0);

//Com que podem rebre encara dades de l'IMU, anem rebent fins que llegim la
seqüencia que ens indica
//que s'ha deixat de transmetre en continuu: 0x10 , 0x00 i 0x00. Per aixo, fem
servir les variables finalX
//per cada valor a llegir.
i = 0;
while((final1==0)||(final2 == 0)||(final3 == 0)){
recv(sockIMU, IMU_getBuf, 1, 0);
//printf("\nLectura[%d]: 0x%x",i, IMU_getBuf[0]);i++;
if(IMU_getBuf[0] == 0x10){
final1 = 1;//printf("\nfinal1");
}else if((IMU_getBuf[0] == 0x00)&&(final1 == 1)&&(final2 == 0)){
final2 = 1;//printf("\nfinal2");
}else if((IMU_getBuf[0] == 0x00)&&(final1 == 1)&&(final2 == 1)){
final3 = 1;//printf("\nfinal3");
}else{final1 = 0; final2 = 0; final3 = 0;}
}
//Rebem els 4 bytes que falten per llegir (al ficar en Polled mode es reben 7 (els
3 ja trobats i els 4 d'acontinuació)
recv(sockIMU, IMU_getBuf, 4, 0);
printf(" Ok !\n");
CloseSock(sockIMU);*/
CloseSock(sockPIC);

printf("\nSortint ...\n\n");
exit(1);
break;
}
}



//FUNCIONS DE TCPSOCK



int CreateTCPServer(int Port)
{

int s;

s = CreateSock(SOCK_STREAM);

if ( s == -1 )
return -1;
```

```
if (BindSock(s,Port) == -1 )
return -1;

if (listen(s, SOMAXCONN) < 0)
return -1;

return s;
}



int AcceptConn(int Sock,char * RemoteAddr)
{
struct sockaddr_in addr;
int addrSize = sizeof(struct sockaddr_in);
int s;

s = accept(Sock,(struct sockaddr *)&addr,(socklen_t *)&addrSize);

strcpy(RemoteAddr,inet_ntoa(addr.sin_addr));

return s;
}

int CreateTCPClient2(struct sockaddr_in dest_addr)
{
int s;
//struct sockaddr_in dest_addr;

s = CreateSock(SOCK_STREAM);

if ( s == -1 )
return -1;

dest_addr.sin_family = AF_INET;
//dest_addr.sin_port = htons(Port);
//HostNameToAddr(HostName,&dest_addr);

if ( connect(s, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) == -1 )
return -1;

return s;
}

int CreateTCPClient(char * HostName,int Port)
{
printf("\n entra al create tcpclient");int s;
struct sockaddr_in dest_addr;

s = CreateSock(SOCK_STREAM);

if ( s == -1 )
return -1;

dest_addr.sin_family = AF_INET;
dest_addr.sin_port = htons(Port);
HostNameToAddr(HostName,&dest_addr);

if ( connect(s, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) == -1 )
{ printf("\n PETA AQUI MERDA! %d \n", errno);
return -1;
}
return s;
}

int TCPSend(int Sock,char * Data,int Len)
{
return send(Sock, Data, Len, 0);
```

```
}

int TCPRecv(int Sock,char * Buf,int Len)
{

return recv(Sock, Buf, Len, 0);
}

//FUNCIONS DE UDPSOCK


int CreateUDPServer(int Port)
{
int s;

s = CreateSock(SOCK_DGRAM);

if ( s == -1 )
return -1;

if (BindSock(s,Port) == -1 )
return -1;


return s;
}




int CreateUDPClient(char * HostName,int Port,int Type,struct sockaddr_in * Serv)
{
int s;

printf("%s %d %d\n",HostName,Port,Type);
s = CreateSock(SOCK_DGRAM);

if ( s == -1 )
{ printf("CreateSock \n");
return -1;
}

Serv->sin_family = AF_INET;
Serv->sin_port = htons(Port);
if (Type == 0)
{
if ( HostNameToAddr(HostName,Serv) == -1)
return -1;
}
else /* HostName = IP address */
{
if ( inet_aton(HostName,&Serv->sin_addr) == 0)
return -1;
}

return s;
}


int UDPRecv(int Sock,char * Buf,int Len)
{

struct sockaddr_in client;
int client_addr_len;

client_addr_len=sizeof(client);
```

```
return                    recvfrom(Sock,Buf,Len,0,(struct                    sockaddr
*)&client,(socklen_t*)&client_addr_len);

}

int UDPSend(int Sock,char * Buf, int Len,struct sockaddr_in * Dest)
{

return sendto(Sock,Buf,Len,0,(struct sockaddr *)Dest,sizeof(struct sockaddr_in));
}



//FUNCIONS DE SOCKBASE




int CreateSock(int SockType)
{
printf("\n entra al createsock" );
return socket(AF_INET,SockType,0);

}

int BindSock(int Sock,int Port)
{
struct sockaddr_in serv;

serv.sin_family=AF_INET;
serv.sin_addr.s_addr=htonl(INADDR_ANY);
serv.sin_port=htons(Port);

return bind(Sock,(struct sockaddr *)&serv,sizeof(serv)) ;

}


int HostNameToAddr(char * HostName,struct sockaddr_in * Addr)
{
struct hostent *hp;

hp=gethostbyname(HostName);
if(hp==(struct hostent *)0)
return -1;

memcpy((char *) &Addr->sin_addr,(char *)hp->h_addr,hp->h_length);


return 0;
}



void CloseSock(int Sock)
{
if ( close(Sock) == -1 )
perror("CloseSock");
```

B. CODI IMU

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Int8.h"
#include "std_msgs/Int16.h"
#include "std_msgs/Int32.h"
#include "std_msgs/Int64.h"
#include "ihm.h"

//INCLUDES DE TCPSOCK.C, UDPSOCK.C I SOCKBACE.C
#include "SockBase.h"
#include "UDPSock.h"
#include <stdio.h>
#include "SockBase.h"

#include <sstream>
//Declaracio de Variables globals
char * addrRobot;
char * addrPIC;
char * addrPc;
char * addrIMU;

int Buf_PC_send_bytes[5];
int indexRebre = 0;
unsigned char bufModeCom[3];

//Definim configuraci socket per PC
static int servSock=-1; /* Socket descriptor for server */
static int clntSock=-1; /* Socket descriptor for client */
static struct sockaddr_in echoServAddr; /* Local address */
static struct sockaddr_in echoClntAddr; /* Client address */
static unsigned int clntLen; /* Length of client address data structure */
static unsigned short echoServPort=15000; //Server port
#define MAXPENDING 5


int sockIMU;
int start_send=0;

// Threads

void * recvIMU(void *num);


void startSocketIMU(void);
void gestio_int (int numero);
int calculaPaquet_PIC(void);
void calculaVarIMU(void);

int main(int argc, char **argv)
{


ros::init(argc, argv, "IMU");


ros::NodeHandle n;

while (ros::ok())
{

/**
* $$$$$ !!!!EL PROGRAMA ANIRA AQUI!!! $$$$$$
*
*/
```

```
int resultatRecv=1;
struct itimerval val; // Variable per la configuraci� del timer
struct sigaction interrupcio; // Variable per la configuraci� d'interrupcions
struct timeval timeoutSock; //Variable per controlar el temps d'espera dels
sockets.

//0x88 = Possem a 1 el bit de més pes per indicar que es un paquet amb el Yaw,
//i l'altre 1 indica que es correció d'angle
PIC_sendBuf_minim[0] = 0x88;

if (1){
addrIMU = "192.168.1.38";
printf("\nX\n");
printf("IP IMU: %s\n",addrIMU);
//iniciem el buffer down
int k=0;
int h=0;
for (k=0; k<=39; k++)
{
bufDown[k]=0;
}
for (h=0; h<=4; h++)
{
for (k=0; k<=39; k++)
{
bufUp[h][k]=0;
}
}


}

// CREACIO DEL THREAD
if (pthread_create (&recvIMUTh, NULL, recvIMU, (void *) 1) != 0){
//printf("\nError al crear el thread de recvPC\n");
}

pthread_mutex_init (&mutex_nBufEnviar, NULL);

val.it_interval.tv_sec=0;
val.it_interval.tv_usec=15000;
val.it_value.tv_sec=0;
val.it_value.tv_usec=15000;
setitimer(ITIMER_REAL, &val, NULL);
//printf ("period=%d\n",PC_send_period);

//Interrupci� timer
interrupcio.sa_handler = gestio_int; // El manejador de de les senyals capturades
es gestio_int
sigemptyset (& (interrupcio . sa_mask)); // Especifiquem que no es bloquejar� cap
senyal
interrupcio.sa_flags = 0; // Tots els flags desactivats
sigaction(SIGALRM, &interrupcio, NULL);
sigaction(SIGINT, &interrupcio, NULL);

while(1){
}


}




}
```

```
void startSocketIMU(void){
sockIMU = CreateTCPClient(addrIMU,10003);
}


// THREAD LECTURA DE IMU
void * recvIMU (void * num){

int bytesRebuts=0, autoritzacio=1;// Autoritzem les peticions de comunicaci
entrants
unsigned char bufReadEprom[2];
signed short aux1,aux2;
bufModeCom[0] = 0x10;
bufModeCom[1] = 0x00;
//Posem aquest byte a 0x31 per demanar que s'envii el vector de angles i
acceleracions (instruccio 0x31)
bufModeCom[2] = 0x31;

bufReadEprom[0] = 0x08;

while(1){
startSocketIMU();
while(bytesRebuts != -1){
bytesRebuts = recv(sockIMU, IMU_getBuf, 1, MSG_OOB);
}
bufReadEprom[1] = 238;
send(sockIMU, bufReadEprom, sizeof(bufReadEprom), 0);
recv(sockIMU, IMU_getBuf, 2, MSG_WAITALL);
aux1 = IMU_getBuf[0];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[1];
calculationCycle1 = aux1|aux2;
printf("calculationCycle1: %d\n",calculationCycle1);


bufReadEprom[1] = 240;
send(sockIMU, bufReadEprom, sizeof(bufReadEprom), 0);
recv(sockIMU, IMU_getBuf, 2, MSG_WAITALL);
aux1 = IMU_getBuf[0];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[1];
calculationCycle2 = aux1|aux2;
printf("calculationCycle2: %d\n",calculationCycle2);

bufReadEprom[1] = 242;
send(sockIMU, bufReadEprom, sizeof(bufReadEprom), 0);
recv(sockIMU, IMU_getBuf, 2, MSG_WAITALL);
aux1 = IMU_getBuf[0];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[1];
calculationCycle3 = aux1|aux2;
printf("calculationCycle3: %d\n",calculationCycle3);
bufReadEprom[1] = 246;
send(sockIMU, bufReadEprom, sizeof(bufReadEprom), 0);
recv(sockIMU, IMU_getBuf, 2, MSG_WAITALL);
aux1 = IMU_getBuf[0];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[1];
calculationCycle4 = aux1|aux2;
printf("calculationCycle4: %d\n",calculationCycle4);

bufReadEprom[1] = 230;
send(sockIMU, bufReadEprom, sizeof(bufReadEprom), 0);
recv(sockIMU, IMU_getBuf, 2, MSG_WAITALL);
aux1 = IMU_getBuf[0];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[1];
```

```
AccelGainScale = aux1|aux2;

printf("AccelGainScale: %d\n",AccelGainScale);

bufReadEprom[1] = 130;
send(sockIMU, bufReadEprom, sizeof(bufReadEprom), 0);
recv(sockIMU, IMU_getBuf, 2, MSG_WAITALL);
aux1 = IMU_getBuf[0];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[1];
GyroGainScale = aux1|aux2;
printf("GyroGainScale: %d\n",GyroGainScale);
//Fiquem mode continu
autoritzacio = send(sockIMU, bufModeCom, sizeof(bufModeCom), 0);
printf("enviat bufModeCom(Mode continu): %d\n",autoritzacio);

printf("bufMode[0]:                0x%x\nbufMode[1]:                0x%x\nbufMode[2]:
0x%x\n",bufModeCom[0],bufModeCom[1],bufModeCom[2]);
//Agafar be el paquet
int i = 0;
autoritzacio = recv(sockIMU, IMU_getBuf, 7, MSG_WAITALL);
for(i=0; i<3; i++){
printf("IMU_getBuf[%d] = 0x%x\n", i, IMU_getBuf[i]);
}

while(1){
//Rebem les dades que ens interessa i les enviem cap al PC
recv(sockIMU,IMU_getBuf,23, MSG_WAITALL);
//recv(sockIMU, &bufUp[);
h_yaw = IMU_getBuf[5];
l_yaw = IMU_getBuf[6];
//calculaVarIMU();
}

}
}

//Desglosem els paquets que ha enviat l'IMU
void calculaVarIMU(void){
//a IMU_getBuf tenim els 20 bytes que ens interessen
unsigned short aux1;
unsigned short aux2;

h_yaw = IMU_getBuf[5];
l_yaw = IMU_getBuf[6];


aux1 = IMU_getBuf[1];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[2];
roll = aux1|aux2;
//printf("\nroll: %f\n",roll);
roll = roll*(360.0/65536.0);

aux1 = IMU_getBuf[3];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[4];
pitch = aux1|aux2;
pitch = pitch*(360.0/65536.0);


aux1 = IMU_getBuf[5];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[6];
yaw = aux1|aux2;
yaw = yaw*(360.0/65536.0);

printf("Roll: %f\tPitch: %f\tYaw: %f\n",roll, pitch, yaw);
```

```
aux1 = IMU_getBuf[7];
aux1 = aux1 < 8;
aux2 = IMU_getBuf[8];
accel_X = aux1|aux2;
accel_X = accel_X/(32768000/AccelGainScale);

aux1 = IMU_getBuf[9];
aux1 = aux1 < 8;
aux2 = IMU_getBuf[10];
accel_Y = aux1|aux2;
accel_Y = accel_Y/(32768000/AccelGainScale);

aux1 = IMU_getBuf[11];
aux1 = aux1 < 8;
aux2 = IMU_getBuf[12];
accel_Z = aux1|aux2;
accel_Z = accel_Z/(32768000/AccelGainScale);

printf("accel_X: %d\taccel_Y: %d\taccel_Z: %d\n",accel_X,accel_Y,accel_Z);

aux1 = IMU_getBuf[13];
aux1 = aux1 < 8;
aux2 = IMU_getBuf[14];
compAngRate_X = aux1|aux2;
compAngRate_X = compAngRate_X/(32768000/GyroGainScale);

aux1 = IMU_getBuf[15];
aux1 = aux1 < 8;
aux2 = IMU_getBuf[16];
compAngRate_Y = aux1|aux2;
compAngRate_Y = compAngRate_Y/(32768000/GyroGainScale);

aux1 = IMU_getBuf[17];
aux1 = aux1 < 8;
aux2 = IMU_getBuf[18];
compAngRate_Z = aux1|aux2;
compAngRate_Z = compAngRate_Z/(32768000/GyroGainScale);

printf("compAngRate_X:             %d\tcompAngRate_Y:             %d\tcompAngRate_Z:
%d\n",compAngRate_X,compAngRate_Y,compAngRate_Z);

/*aux1 = IMU_getBuf[19];
aux1 = aux1 << 8;
aux2 = IMU_getBuf[20];
timerTicks_IMU = aux1|aux2;
timerTicks_IMU = timerTicks_IMU*0.0065536;*/
//printf("timerTicks: %d\n",timerTicks_IMU);


}

// Gesti□ de les senyals capturades
void gestio_int (int numero)
{
int resultatRecv = 0;
int i;
char final1=0, final2=0, final3=0;

switch (numero)
{
case SIGALRM :
//Flag que controla quan enviar al PIC per conservar l'ample de banda
sendTimer = 1;

break;

case SIGINT:
/*******************************************
```

```
Codi executat per la interrupció de Ctrl+C. A part, està configurat per que quan
Labview es desconecti
salti aquesta interrupció.

Aquesta part ens assegura que la transmissió de l'IMU es "tanca" correctament i que
no queda res
al buffer.
*******************************************/
printf("\nS'ha pulsat Ctrl+C!\n");

printf("\nFinalitzant comunicacio amb IMU...");
//Configurem el buffer per deixar de transmetre en continuu
bufModeCom[0] = 0x10;
bufModeCom[1] = 0x00;
bufModeCom[2] = 0x00;
//Enviem els 3 bytes
resultatRecv = send(sockIMU, bufModeCom, 3, 0);

//Com que podem rebre encara dades de l'IMU, anem rebent fins que llegim la
seqüencia que ens indica
//que s'ha deixat de transmetre en continuu: 0x10 , 0x00 i 0x00. Per aixo, fem
servir les variables finalX
//per cada valor a llegir.
i = 0;
while((final1==0)||(final2 == 0)||(final3 == 0)){
recv(sockIMU, IMU_getBuf, 1, 0);
//printf("\nLectura[%d]: 0x%x",i, IMU_getBuf[0]);i++;
if(IMU_getBuf[0] == 0x10){
final1 = 1;//printf("\nfinal1");
}else if((IMU_getBuf[0] == 0x00)&&(final1 == 1)&&(final2 == 0)){
final2 = 1;//printf("\nfinal2");
}else if((IMU_getBuf[0] == 0x00)&&(final1 == 1)&&(final2 == 1)){
final3 = 1;//printf("\nfinal3");
}else{final1 = 0; final2 = 0; final3 = 0;}
}
//Rebem els 4 bytes que falten per llegir (al ficar en Polled mode es reben 7 (els
3 ja trobats i els 4 d'acontinuació)
recv(sockIMU, IMU_getBuf, 4, 0);
printf(" Ok !\n");
CloseSock(sockIMU);

printf("\nSortint ...\n\n");
exit(1);
break;
}
}

//FUNCIONS TCP


int CreateTCPServer(int Port)
{
int s;

s = CreateSock(SOCK_STREAM);

if ( s == -1 )
return -1;

if (BindSock(s,Port) == -1 )
return -1;

if (listen(s, SOMAXCONN) < 0)
return -1;

return s;
}
```

```
int AcceptConn(int Sock,char * RemoteAddr)
{
struct sockaddr_in addr;
int addrSize = sizeof(struct sockaddr_in);
int s;

s = accept(Sock,(struct sockaddr *)&addr,(socklen_t *)&addrSize);

strcpy(RemoteAddr,inet_ntoa(addr.sin_addr));

return s;
}

int CreateTCPClient2(struct sockaddr_in dest_addr)
{
int s;
//struct sockaddr_in dest_addr;

s = CreateSock(SOCK_STREAM);

if ( s == -1 )
return -1;

dest_addr.sin_family = AF_INET;
//dest_addr.sin_port = htons(Port);
//HostNameToAddr(HostName,&dest_addr);

if ( connect(s, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) == -1 )
return -1;

return s;
}

int CreateTCPClient(char * HostName,int Port)
{
int s;
struct sockaddr_in dest_addr;

s = CreateSock(SOCK_STREAM);

if ( s == -1 )
return -1;

dest_addr.sin_family = AF_INET;
dest_addr.sin_port = htons(Port);
HostNameToAddr(HostName,&dest_addr);

if ( connect(s, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) == -1 )
return -1;

return s;
}

int TCPSend(int Sock,char * Data,int Len)
{
return send(Sock, Data, Len, 0);
}

int TCPRecv(int Sock,char * Buf,int Len)
{

return recv(Sock, Buf, Len, 0);
}

//FUNCIONS SOCKBASE
```

```
int CreateSock(int SockType)
{
return socket(AF_INET,SockType,0);

}

int BindSock(int Sock,int Port)
{
struct sockaddr_in serv;

serv.sin_family=AF_INET;
serv.sin_addr.s_addr=htonl(INADDR_ANY);
serv.sin_port=htons(Port);

return bind(Sock,(struct sockaddr *)&serv,sizeof(serv)) ;

}


int HostNameToAddr(char * HostName,struct sockaddr_in * Addr)
{
struct hostent *hp;

hp=gethostbyname(HostName);
if(hp==(struct hostent *)0)
return -1;

memcpy((char *) &Addr->sin_addr,(char *)hp->h_addr,hp->h_length);


return 0;
}



void CloseSock(int Sock)
{
if ( close(Sock) == -1 )
perror("CloseSock");
}
```

C. CODI LISTENER2

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Int8.h"

/**
* This tutorial demonstrates simple receipt of messages over the ROS system.
*/
void chatterCallback(const std_msgs::Int8::ConstPtr& msg)
{
ROS_INFO("I heard: [%d]", msg->data);
}

int main(int argc, char **argv)
{
/**
* The ros::init() function needs to see argc and argv so that it can perform
* any ROS arguments and name remapping that were provided at the command line. For
programmatic
* remappings you can use a different version of init() which takes remappings
* directly, but for most command-line programs, passing argc and argv is the
easiest
* way to do it. The third argument to init() is the name of the node.
*
* You must call one of the versions of ros::init() before using any other
* part of the ROS system.
*/
ros::init(argc, argv, "listener2");

/**
* NodeHandle is the main access point to communications with the ROS system.
* The first NodeHandle constructed will fully initialize this node, and the last
* NodeHandle destructed will close down the node.
*/
ros::NodeHandle n;

/**
* The subscribe() call is how you tell ROS that you want to receive messages
* on a given topic. This invokes a call to the ROS
* master node, which keeps a registry of who is publishing and who
* is subscribing. Messages are passed to a callback function, here
* called chatterCallback. subscribe() returns a Subscriber object that you
* must hold on to until you want to unsubscribe. When all copies of the Subscriber
* object go out of scope, this callback will automatically be unsubscribed from
* this topic.
*
* The second parameter to the subscribe() function is the size of the message
* queue. If messages are arriving faster than they are being processed, this
* is the number of messages that will be buffered up before beginning to throw
* away the oldest ones.
*/
ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
/**
* ros::spin() will enter a loop, pumping callbacks. With this version, all
* callbacks will be called from within this thread (the main one). ros::spin()
* will exit when Ctrl-C is pressed, or the node is shutdown by the master.
*/
ros::spin();

return 0;
}
```

D. CODI LISTENER

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Int8.h"
#include "std_msgs/Int16.h"
#include "std_msgs/Float64.h"
#include "std_msgs/Float32.h"

/**
* !CALLBACKS!
*/
void tempsCallback(const std_msgs::Int8::ConstPtr& msg2)
{
ROS_INFO("temps: [%d]", msg2->data);
}

void ErrorCallback(const std_msgs::Int8::ConstPtr& msg2)
{
ROS_INFO("Error: [%d]", msg2->data);
}
//Velocitats

void Vel_FLCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("Velocitat Roda davant-esquerra: [%f]", msg2->data);
}
void Vel_FRCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("Velocitat Roda davant-dreta: [%f]", msg2->data);
}

void Vel_BLCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("Velocitat Roda darrera-esquerra: [%f]", msg2->data);
}

void Vel_BRCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("Velocitat Roda darrera-dreta: [%f]", msg2->data);
}
//Senyals de control

void Cont_FLCallback(const std_msgs::Int8::ConstPtr& msg2)
{
ROS_INFO("Control Roda davant-esquerra: [%d]", msg2->data);
}
void Cont_FRCallback(const std_msgs::Int8::ConstPtr& msg2)
{
ROS_INFO("Control Roda davant-dreta: [%d]", msg2->data);
}

void Cont_BLCallback(const std_msgs::Int8::ConstPtr& msg2)
{
ROS_INFO("Control Roda darrera-esquerra: [%d]", msg2->data);
}

void Cont_BRCallback(const std_msgs::Int8::ConstPtr& msg2)
{
ROS_INFO("Control Roda darrera-dreta: [%d]", msg2->data);
}
//Intensitats
void Int_FLCallback(const std_msgs::Float64::ConstPtr& msg2)
{
ROS_INFO("Intensitat Roda davant-esquerra: [%f]", msg2->data);
}
```

```
void Int_FRCallback(const std_msgs::Float64::ConstPtr& msg2)
{
ROS_INFO("Intensitat Roda davant-dreta: [%f]", msg2->data);
}

void Int_BLCallback(const std_msgs::Float64::ConstPtr& msg2)
{
ROS_INFO("Intensitat Roda darrera-esquerra: [%f]", msg2->data);
}

void Int_BRCallback(const std_msgs::Float64::ConstPtr& msg2)
{
ROS_INFO("Velocitat Roda darrera-esquerra: [%f]", msg2->data);
}

//Bateria

void BatCallback(const std_msgs::Float64::ConstPtr& msg2)
{
ROS_INFO("Bateria: [%f]V", msg2->data);
}
void VccCallback(const std_msgs::Float64::ConstPtr& msg2)
{
ROS_INFO("Vcc: [%f]V", msg2->data);
}

//Posicio

void PosXCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("PosX: [%f]m", msg2->data);
}

void PosYCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("PosY: [%f]m", msg2->data);
}
void ThetaCallback(const std_msgs::Float32::ConstPtr& msg2)
{
ROS_INFO("Theta: [%f]°", msg2->data);
}


int main(int argc, char **argv)
{

/**
* The ros::init() function needs to see argc and argv so that it can perform
* any ROS arguments and name remapping that were provided at the command line. For
programmatic
* remappings you can use a different version of init() which takes remappings
*  directly, but for most command-line programs, passing argc and argv is the
easiest
* way to do it. The third argument to init() is the name of the node.
*
* You must call one of the versions of ros::init() before using any other
* part of the ROS system.
*/
ros::init(argc, argv, "listener");

/**
* NodeHandle is the main access point to communications with the ROS system.
* The first NodeHandle constructed will fully initialize this node, and the last
* NodeHandle destructed will close down the node.
*/
ros::NodeHandle n;

/**
```

```
* The subscribe() call is how you tell ROS that you want to receive messages
* on a given topic. This invokes a call to the ROS
* master node, which keeps a registry of who is publishing and who
* is subscribing. Messages are passed to a callback function, here
* called chatterCallback. subscribe() returns a Subscriber object that you
* must hold on to until you want to unsubscribe. When all copies of the Subscriber
* object go out of scope, this callback will automatically be unsubscribed from
* this topic.
*
* The second parameter to the subscribe() function is the size of the message
* queue. If messages are arriving faster than they are being processed, this
* is the number of messages that will be buffered up before beginning to throw
* away the oldest ones.
*/
//ros::Subscriber sub1 = n.subscribe("chattor", 1000, chattorCallback);
ros::Subscriber sub2 = n.subscribe("Temps", 1000, tempsCallback);
ros::Subscriber sub3 = n.subscribe("Error", 1000, ErrorCallback);
ros::Subscriber sub4 = n.subscribe("Velocitat_FL", 1000, Vel_FLCallback);
ros::Subscriber sub5 = n.subscribe("Velocitat_FR", 1000, Vel_FRCallback);
ros::Subscriber sub6 = n.subscribe("Velocitat_BL", 1000, Vel_BLCallback);
ros::Subscriber sub7 = n.subscribe("Velocitat_BR", 1000, Vel_BRCallback);
ros::Subscriber sub8 = n.subscribe("Intensitat_FL", 1000, Int_FLCallback);
ros::Subscriber sub9 = n.subscribe("Intensitat_FR", 1000, Int_FRCallback);
ros::Subscriber sub10 = n.subscribe("Intensitat_BL", 1000, Int_BLCallback);
ros::Subscriber sub11 = n.subscribe("Intensitat_BR", 1000, Int_BRCallback);
ros::Subscriber sub12 = n.subscribe("Bateria", 1000, BatCallback);
ros::Subscriber sub13 = n.subscribe("Vcc", 1000, VccCallback);
ros::Subscriber sub14 = n.subscribe("Control_FL", 1000, Cont_FLCallback);
ros::Subscriber sub15 = n.subscribe("Control_FR", 1000, Cont_FRCallback);
ros::Subscriber sub16 = n.subscribe("Control_BL", 1000, Cont_BLCallback);
ros::Subscriber sub17 = n.subscribe("Control_BR", 1000, Cont_BRCallback);
ros::Subscriber sub18 = n.subscribe("Pos_X", 1000, PosXCallback);
ros::Subscriber sub19 = n.subscribe("Pos_Y", 1000, PosYCallback);
ros::Subscriber sub20 = n.subscribe("Theta", 1000, ThetaCallback);


/**
* ros::spin() will enter a loop, pumping callbacks. With this version, all
* callbacks will be called from within this thread (the main one). ros::spin()
* will exit when Ctrl-C is pressed, or the node is shutdown by the master.
*/
ros::spin();

return 0;
}
```

E. COMUNICACIO PIC AMB PC

A continuació es descriuran tots els bytes de comunicació entre el PIC i el PC. Primer s'explicaran els bytes del PC cap al PIC i després els bytes de resposta del PIC cap al PC.

Al tractar-se de comunicació sèrie no es poden tractar les dades com "paquets" de forma literal, tot i que aquí es descriuen d'aquesta forma. La mida del paquet de dades sempre ha de concordar perfectament amb allò descrit pels flags per tal que es mantingui el sincronisme de les transmissions.

PAQUET PC A PIC

A continuació es detalla byte a byte el significat de les comandes per enviar al robot.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
IMU_pkg	Conf_PID	Err_ad	--	Corr_T	Corr_pos	T_order	Pos_order

Aquest byte defineix la mida del paquet total de dades enviades.
Aquest byte es obligatori en totes les transmissions.

Pos_order Determina si el paquet de dades conté una consigna de posició
0: No s'envia
1: Sí que s'envia

T_order Determina si el paquet de dades conté una consigna d'angle final
0: No s'envia
1: Sí que s'envia

Corr_pos Determina si el paquet de dades conté una correcció de posició
0: No s'envia
1: Sí que s'envia

Corr_T Determina si el paquet de dades conté una correcció de l'orientació
0: No s'envia
1: Sí que s'envia

Err_ad Determina si el paquet de dades conte un nou valor per l'error admissible durant el control de posició
0: No s'envia
1: Sí que s'envia

Conf_PID Determina si el paquet de dades conté nous paràmetres per al control de velocitat PID

0: No s'envia

1: Sí que s'envia

IMU_pkg Determina si el paquet conté únicament la correcció d'angle de l'IMU (cas especial de paquet de 3 bytes). L'activació d'aquest bit implica que el flag Corr_T hagi d'estar també activat.

0: No s'envia, el paquet és estàndard.

1: Sí que s'envia, el paquet només conté el yaw.

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Encóders		Diff. tick	Ctl. Sig.	Current	Position	Batery	Speed

Aquest byte defineix quines dades ha de retornar el dsPIC en el seu missatge.
Aquest byte es obligatori en totes les transmissions (excepte si IMU_pkg = 1)

Speed Determina si el dsPIC envia les velocitats de cada roda

0: No s'envia

1: Sí que s'envia

Batery Determina si el dsPIC envia l'estat de la bateria i de la font de 5 Vdc

0: No s'envia

1: Sí que s'envia

Position Determina si el dsPIC envia la posició del robot calculada per odometria

0: No s'envia

1: Sí que s'envia

Current Determina si el dsPIC envia els corrents mesurats en cada roda

0: No s'envia

1: Sí que s'envia

Ctl. Sig Determina si el dsPIC envia els senyals de control aplicats a cada roda

0: No s'envia

1: Sí que s'envia

Diff. Tick Determina si el dsPIC envia les diferencies de gir mesurades entre la roda de davant i la de darrere

0: No s'envia

1: Sí que s'envia

Encóders Determina si el dsPIC envia les lectures dels encóders

0: No s'envia

1: Sí que s'envia

byte 2	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
					Ang_rate	Acceleration	Euler_ang

Aquest byte defineix quines dades ha de retornar la Litestation dins el seu missatge. És un byte només interpretat per la Litestation (pel cas de comunicacions indirectes) i ignorat pel dsPIC.

Aquest byte és obligatori en totes les transmissions (excepte si IMU_pkg = 1)

Euler_ang Determina si la Litestation envia els angles de l'IMU

0: No s'envia

1: Sí que s'envia

Accelerations Determina si la Litestation envia les acceleracions de l'IMU

0: No s'envia

1: Sí que s'envia

Ang_rate Determina si la Litestation envia les velocitats angulars de l'IMU

0: No s'envia

1: Sí que s'envia

byte 3	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
PID_sel	Set_offset		Led_2	Led_1	Mode_PID_1	Mode_PID_0	Ctl_Speed

Aquest byte defineix configuracions generals del robot

Aquest byte és obligatori en totes les transmissions (excepte si IMU_pkg = 1)

Ctl_Speed Determina si es fa control de velocitat

0: PID desactivat (llaç obert)

1: PID activat (llaç tancat)

Mode_PID Determina si el PID s'evalua independentment per cada roda o per parelles

00: PID per parelles (dreta i esquerra) (velocitat)

01: PID independent per cada roda (velocitat)

10: PID independent per cada roda (corrent)

Led_x Controla els leds d'alta intensitat per al sistema de localització (opcional)
0: Led desactivat
1: Led activat

Set_offset Actualitza el valor d'offset de les corrents al valor actual
0: No fa res
1: Actualitza amb la última lectura

PID_sel Selecciona sobre quin PID aplicar les configuracions
0: PID de velocitat (depèn del mode de PID)
1: PID de corrent (depèn del mode de PID)

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

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

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

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

Aquests 4 bytes defineixen la consigna de velocitat per cada roda
Aquests bytes són obligatoris en totes les transmissions (excepte si IMU_pkg=1)

Order_xy Consignes de velocitat
Si Ctl_Speed = 0: Senyal de control [-100% ~ +100%]
Si Ctl_Speed = 1: Consigna de velocitat [-125cm/s ~ +125cm/s]

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Order_X_cm							

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Order_Y_cm							

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Order_X_m							

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Order_Y_m							

Aquests 4 bytes permeten donar una consigna de posició al robot. Sempre que la consigna de velocitat ho permeti, el robot es mourà fins arribar al punt donat a la consigna.
Aquests bytes són opcionals en funció del bit Pos_order

Order_x_cm Consignes de posició per als eixos X i Y, fracció de centímetres [-99cm ~ +99cm]
Order_x_m Consignes de posició per als eixos X i Y, part entera [-125m ~ +125m]

byte 12	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Order_T							

Aquest byte permet donar una consigna de posició angular. Si aquest byte es combina amb una consigna de posició, el robot es traslladarà al punt de consigna i seguidament aplicarà la consigna d'angle al final.
Aquest byte és opcional en funció del bit T_order.

Order_T Consignes de posició angular proporcional a 0º~360º [0 ~ 255]

byte 13	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Corr_X_cm							

byte 14	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Corr_Y_cm							

byte 15	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Corr_X_m							

byte 16	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Corr_Y_m							

Aquests 4 bytes permeten donar una correcció de la posició del robot en els eixos de coordenades.
Aquests bytes son opcionals en funció del bit Corr_pos.

Corr_x_cm Correcció de posició per als eixos X i Y, fracció de centímetres [-99cm ~ +99cm]

Corr_x_m Correcció de posició per als eixos X i Y, part entera [-125m ~ +125m]

byte 17	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Correction_T_Hi							

byte 18	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Correction_T_Lo							

Aquest byte permet donar una correcció de la posició angular del robot. Si el bit yaw_to_PIC es troba actiu, la Litestation inserirà en aquests bytes la lectura de l'IMU. En cas contrari es podrà inserir una correcció des del sistema de control.
Aquest byte és opcional en funció del bit Corr_T.

Correction_T Correcció de posició angular proporcional a 0º~360º x 100.

byte 19	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Error_ad							

Aquest byte permet fixar la precisió amb la qual es realitza el control de posició del robot.
Aquest byte és opcional en funció del bit Err_ad.

Eorror_ad Error de posició admissible (distància absoluta) [0cm ~ 100cm]

byte 20	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Kp_Hi							

byte 21	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Kp_Lo							

byte 22	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ki_Hi							

byte 23	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ki_Lo							

byte 24	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Kd_Hi							

byte 25	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Kd_Lo							

Aquests 6 bytes permeten modificar els paràmetres Kp, Ki i Kd del control de velocitat. Aquests valors s'han d'enviar en format de coma flotant de 16 bits dividits en 2 bytes. Quan s'apliquen nous valors, es genera un codi d'error específic per informar que s'ha fet correctament el canvi.
Aquests bytes son opcionals en funció del bit Conf_PID.

Kp_x	Constant proporcional del PID
Ki_x	Constant integral del PID (Ki=0 per control PD)
Kd_x	Constant derivativa del PID (Kd=0 per control PI)

PAQUET PIC A PC

La resposta del robot igualment està formada per una part fixe del paquet, és a dir, que sempre s'envia, i per una part variable. Les dades que inclou la part variable del paquet depenen de les dades que s'han sol·licitat mitjançant el paquet de comandes.

byte 0	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Encóders	Sensors	Diff. tick	Ctl. Sig.	Current	Position	Batery	Speed

Aquest byte defineix la mida del paquet total de dades enviades.
Aquest byte és obligatori en totes les transmissions.

Speed Determina si s'envien les velocitats de cada roda
0: No s'envia
1: Sí que s'envia

Batery Determina si s'envia l'estat de la bateria i de la font de 5 Vdc
0: No s'envia
1: Sí que s'envia

Position Determina si s'envia la posició del robot calculada per odometria
0: No s'envia
1: Sí que s'envia

Current Determina si s'envien els corrents mesurats en cada roda
0: No s'envia
1: Sí que s'envia

Ctl. Sig Determina si s'envien els senyals de control aplicats a cada roda
0: No s'envia
1: Sí que s'envia

Diff. Tick Determina si s'envien les diferencies de gir mesurades entre la roda de davant i de darrera
0: No s'envia
1: Sí que s'envia

Sensors Determina si s'envien les lectures dels sensors infrarrojos
0: No s'envia
1: Sí que s'envia

Encóders Determina si s'envien les lectures dels encóders
0: No s'envia
1: Sí que s'envia

byte 1	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
					Ang_rate	Acceleration	Euler_ang

Aquest byte defineix si el paquet conté dades afegides per la Litestation. Aquest

byte no té cap efecte si es fa servir el mètode de comunicació directe.
Aquest byte sempre s'envia

Euler_ang Determina si el paquet conté les orientacions llegiedes per l'IMU
0: No s'envia
1: Sí que s'envia

Accelerations Determina si el paquet conté les acceleracions
0: No s'envia
1: Sí que s'envia

Ang_rate Determina si el paquet conté les velocitats angulars
0: No s'envia
1: Sí que s'envia

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

Aquest byte marca la base de temps de les dades rebudes. El receptor ha de ser capaç de detectar el pas per zero. D'aquesta manera podrà interpretar correctament la base de temps.
Aquest byte sempre s'envia.

Time_stamp Base de temps [0 ~ 255ms] (quan passa de 255 torna al valor 0)

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

Aquest byte dona informació sobre mals funcionaments del robot. Cada codi correspon a un determinat missatge.
Aquest byte sempre s'envia.

001 Overflow T6 (RL)
002 Overflow T7 (FL)
003 Overflow T8 (FR)
004 Overflow T9 (RR)
005 Error flag driver M1 (FL)
006 Error flag driver M2 (FR)

007 Error flag driver M3 (RL)
008 Error flag driver M4 (RR)
011 Overflow en Con_FL
012 Overflow en Con_FR
013 Overflow en Con_RL
014 Overflow en Con_RR
016 Overflow en Speed FL (Speed value is max or min)
017 Overflow en Speed FR (Speed value is max or min)
018 Overflow en Speed RL (Speed value is max or min)
019 Overflow en Speed RR (Speed value is max or min)
020 PID parameters changed succesfully
021 Error I2C2 bus (sensors) lost data
022 Error I2C2 bus (sensors) was busy
023 Comunications timeout (causes stop)
026 Overflow on position correction (x)
027 Overflow on position correction (y)
028 Batery overvoltage
029 5V source overvoltage
030 Overflow Pos_x (Pos_x will be 0)
031 Overflow Pos_y (Pos_y will be 0)
032 Overflow on Int_FL (Int is max or min)
033 Overflow on Int_FR (Int is max or min)
034 Overflow on Int_RL (Int is max or min)
035 Overflow on Int_RR (Int is max or min)
036 Overflow on Sens_1 value (Sens is max)
037 Overflow on Sens_2 value (Sens is max)
038 Overflow on Sens_3 value (Sens is max)
039 Overflow on Sens_4 value (Sens is max)
040 Overflow on tic_R
041 Overflow on tic_L

byte 4	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
--	--	--	--	--	Arrived	Les_2_st	Led_1_st

Aquest byte dóna informacions generals.

Aquest byte sempre s'envia.

Led_x_st Informa sobre l'estat dels Led's

0: Apagat

1: Encès

Arrived En cas de treballar en mode de control de posició, informa sobre si s'ha assolit la posició final

0: El robot es troba en marxa

1: El robot ha arribat al destí

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

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

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

byte 8	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Speed_RR							

Aquests bytes donen la velocitat de cada roda del robot i s'envien en funció del bit Speed

Speed_xy Velocitat de la roda [-125cm/s ~ +125cm/s]

byte 9	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Batery							

byte 10	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Vcc							

Aquests bytes donen informació sobre l'estat de les alimentacions.

Aquests bytes s'envien en funció del bit Batery

Batery Nivell de la bateria [0 ~ 160 V·10]

Vcc Nivell de la font de 5V [0 ~ 200V·100] (Al valor s'han de sumar 3.5V

per obtenir el nivell)

byte 11	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pos_X_cm							

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

Aquests 5 bytes donen la posició del robot en els eixos cartesians calculada per odometria i s'envien en funció del bit Position

Pos_x_cm Posició per als eixos X i Y, fracció de centímetres [-99cm ~ +99cm]
Pos_x_m Posició per als eixos X i Y, part entera [-125m ~ +125m]
Pos_T Posició angular [0~256] equivalent [0º~360º]

byte 16	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_FL_Hi							

byte 17	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_FL_Lo							

byte 18	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_FR_Hi							

byte 19	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_FR_Lo							

byte 20	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_RL_Hi							

byte 21	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_RL_Lo							

byte 22	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_RR_Hi							

byte 23	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Current_RR_Lo							

Aquests 8 bytes donen les lectures de corrent de cada motor. Els valors estan representats com a enters de 16 bits fraccionats en dos bytes.
Aquests bytes són opcionals en funció del bit Current.

Current_xy_z Valor de corrent. [-5000mA ~ +5000mA]

byte 24	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Control_FL							

byte 25	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Control_FR							

byte 26	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Control_RL							

byte 27	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Control_RR							

Aquests 4 bytes donen la senyal de control aplicada a cada roda del robot
Aquests bytes s'envien en funció del bit Control

Control_xy Senyal de control [-100% ~ +100%]

byte 28	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Diff_tick_R							

byte 29	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Diff_tick_L							

Aquests 2 bytes donen la diferència de gir entre la roda davantera i la de darrere de cada costat en polsos dels encóders.
Aquests bytes s'envien en funció del bit Diff_tick.

Diff_tick_x Diferència de gir per les rodes de la dreta i l'esquerra en valor absolut [0 ~ 255 polsos]

byte 30	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Temp_Hi							

byte 31	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Temp_Lo							

byte 32	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_1							

byte 33	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Sens_2							

Aquests 4 bytes donen la lectura de quatre sensors analògics.
Aquests bytes s'envien en funció del bit Sensor.

Temp_x Lectura del sensor de distància [0~4096] equivalent a [0~100ºC]
Sens_x Sensor de distància – actualment no implementat --

byte 34	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_L_Hi							

byte 35	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_L_Lo							

byte 36	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_R_Hi							

byte 37	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Enc_R_Lo							

Aquests 4 bytes donen la lectura dels encóders. Cada valor correspon a la mitjana dels valors de les rodes del costat dret i esquerre. Els valors s'expressen com un enter de 16 bits fraccionat en dos bytes. S'interpreten com un increment respecte l'últim valor enviat.
Aquest byte s'envia en funció del bit encóder.

Enc_x_y Lectura de l'encóder. [-32512 ~ +32512 polsos]

byte 38	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Roll_Hi							

byte 39	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Roll_Lo							

byte 40	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pitch_Hi							

byte 41	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Pitch_Lo							

byte 42	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Yaw_Hi							

byte 43	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Yaw_Lo							

Aquests 6 bytes donen la lectura d'orientació obtinguda amb l'IMU. Aquests bytes no es poden obtenir en mode de comunicació directa.
Aquest byte s'envia en funció del bit Euler_ang

Roll_x Angle Roll [0º~360º]

Yaw_x Angle Yaw [0º~360º]

Pitch_x Angle Pitch [0º~360º]

byte 44	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Accel_X_Hi							

byte 45	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Accel_X_Lo							

byte 46	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Accel_Y_Hi							

byte 47	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Accel_Y_Lo							

byte 48	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Accel_Z_Hi							

byte 49	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Accel_Z_Lo							

Aquests 6 bytes donen la lectura d'acceleracions en els 3 eixos obtinguda amb l'IMU. Aquests bytes no es poden obtenir en mode de comunicació directa. S'envien en funció del bit Accel

byte 50	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ang_rate_X_Hi							

byte 51	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ang_rate_X_Lo							

byte 52	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ang_rate_Y_Hi							

Byte 53	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ang_rate_Y_Lo							

byte 54	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ang_rate_Z_Hi							

byte 55	bit: 6	bit: 5	bit: 4	bit: 3	bit: 2	bit: 1	bit: 0
Ang_rate_Z_Lo							

Aquests 6 bytes donen la lectura de velocitat angular en els tres eixos obtinguda amb l'IMU. Aquests bytes no es poden obtenir en mode de comunicació directa. S'envien en funció del bit Ang_rate