Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi:Grau en Enginyeria Electrònica Industrial i Automàtica

Títol: Control de moviment del robot BIGBOT de rescat amb sistema ROS en grans espais exteriors

Document: 1. Memòria

Alumne: Alex Martín Rio

Tutor: Albert Figueras Coma
Departament: Enginyeria Elèctrica, Electrónica i Automàtica
Àrea: Enginyeria de Sistemes i Automàtica

Convocatòria (mes/any): juny/2016

ÍNDEX

1. INTRODUCCIÓ

1.1 Antecedents

En el laboratori del grup de recerca ARLAB es disposa d'una sèrie de robots mòbils pel rescat de supervivents en catàstrofes. Aquests permeten la comunicació i cooperació amb la gent i amb altres robots i identificar situacions de perill en operacions de rescat.

Són robots de quatre rodes per moure's per exteriors i estan equipats amb diferents sistemes de sensorització com càmeres de localització, odometria amb encoders.

1.2 Objecte

El present projecte servirà per dur a terme un control de moviment per un robot mòbil de rescat que navegarà per exteriors on el terreny no és uniforme ni únic. Es pretén que tot l'estudi realitzat pugui significar el punt de partida en aplicacions futures on el robot sigui molt més manejable i més eficient en les actuacions en ambients exteriors.

1.3 Especificacions i abast

S'augmentarà el nombre de terrenys per on es pot moure el robot, és realitzarà en MATLAB el càlcul del factor de correcció de terreny.

Utilitzarem els assajos fets en els diferents terrenys per poder fer al final una prova, on el robot anirà sol per un circuit de diferents terrenys.

2. PROJECTE DE RESCAT

El projecte de recerca en robòtica de rescat és un projecte que s'està desenvolupant per investigadors del laboratori de Sistemes Intel·ligents de la UdG la qual es centra en la recerca i salvament de persones mitjançant elements robòtics.

Un dels mètodes més utilitzats en el rescat de persones és la utilització de gossos ensinistrats, ja que amb les seves dimensions i agilitat són capaços d'endinsar-se en zones on una persona no podria entrar-hi, a més, aquests animals disposen d'un molt desenvolupat sentit de l'olfacte, fet que ajudarà a trobar a persones, en situacions com l'ensorrament d'edificis.

La proposta de projecte de recerca es basa en la utilització d'un robot mòbil per a donar suport als equips de rescat en la recerca de persones en entorns pre-urbans i urbans on s'ha produït una catàstrofe.

La principal idea de la qual parteix aquest projecte, és realitzar un robot mòbil capaç de cooperar amb un gos ensinistrat dins una situació de recerca o rescat de persones. El robot, sempre supervisat per un operari expert, incorpora un gran nombre d'elements que juntament amb les característiques del gos, han de incrementar l'eficiència dels grups de rescat davant un gran nombre de catàstrofes.

Per realitzar-ho, s'està implementant un robot mòbil autònom, capaç de seguir al gos amb la capacitat d'evitar obstacles, però al mateix temps, emetent informació a l'operari a través dels diferents sensors que incorpora, com la càmera KINECT. Permeten així realitzar una comunicació entre l'ensinistrador i el gos utilitzant el robot d'intermediari. Per dur a terme aquest projecte, es parteix de la flota de robots de rescat realitzada al laboratori ARLAB, fabricats anteriorment, els quals s'adaptaran per poder utilitzar-los com a primer prototip d'enllaç entre l'operari expert i el gos.

Com es pot observar a la figura 1, aquesta flota està formada per robots amb estructures similars entre ells, però amb diferents característiques. No obstant, el projecte de rescat es centrarà en el desenvolupament del robot BIGBOT.

Figura 1. Flota actual de rescat

Un dels principals problemes de l'actual flota de rescat, és el baix nombre de terrenys que detecta, la qual cosa fa que el robot no pugui anar per totes les superfícies possibles. Per aquest fet s'ha plantejat augmentar el nombre de terrenys, això farà que sigui un vehicle més complert.

Als següents capítols s'exposaran els elements que formen part del projecte i la solució implementada.

3. ESTRUCTURA DEL BIGBOT

En aquest capítol s'explicaran el elements que formen part del BIGBOT. Com es pot observar a la figura 2, el robot disposa d'una estructura quadrada metal·litzada, adient per un bon aïllament galvànic. Per aconseguir una bona resposta davant qualsevol superfície, disposa quatre rodes de grans dimensions independents entre elles.

A la següent figura, es pot observar la estructura del BIGBOT.

Figura 2. Esquema actual del BIGBOT

El robot disposa d'uns encoders i uns motors de corrent continua per cada una de les rodes, una placa controladora amb un dsPIC que gestiona els diferents PID's de regulació de velocitat, encoders i l'etapa de potència per moure les rodes, d'un sensor IMU, que permet saber la posició X i posició Y respecte un punt inicial, una càmera RGB-D, que permet la gravació d'imatges, una càmera tèrmica, que permet la visió en zones de poca il·luminació i un mini ordinador que gestiona el sistema ROS.

A continuació es farà una petita descripció dels elements del BIGBOT que s'utilitzaran per l'execució d'aquest treball, i se'n mostraran algunes especificacions.

3.1. PC

Per donar autonomia al robot, s'ha col·locat un ordinador sobre la base del robot. El PC te una mida reduïda, és de baix consum, ja que ha d'incorporar-se adequadament al robot, té una potència de processament alta i que permet executar el sistema Ubuntu Linux de forma completa, i disposa d'un disc dur intern. El PC ha de tenir un processador Intel o AMD, ja sigui d'arquitectura x86 o x64, és a dir, processadors de 32 bits o 64 bits els quals són compatibles amb les versions de Ubuntu d'escriptori.

El mini-PC és un ordinador petit anomenat HTPC (Home Theater Personal Computer) els quals estan pensats principalment per a l'ús de sistemes Home Cinema, però com son molt compactes i compleixen amb les especificacions esmentades anteriorment, s'ha optat per la seva utilització. El mini-PC instal·lat en el BIGBOT és de la casa intense PC. A la taula 1 es poden observar les característiques principals del mini-PC.

Figura 3. Intense PC cara davantera i posterior

Mides	190x160x40mm
Potència de consum	17 Watts
Voltatge d'alimentació	De 10 a 16V
CPU	Intel core i3 3217UE 64bits 1.6GHz
Targeta gràfica	HD Graphics 4000 Intel
Memòria	2x2Gb DDR3-1066, 64-bits (ampliable a 16Gb)
Ports	USB 6x USB 2.0 a 480Mbit/s i 2x USB 3.0 a 5Gbit/s
Port sèrie	Mini RS232
Ethernet	2x RJ45
WLAN	WIFI 802.11b/g/n, dues antenes, 150Mbit/s
Disc dur	120GB (S-ATA)
Àudio	Entrada estèreo 7.1 S/PDIF, jack 3,5m;m; sortida mono

Taula 1. Especificacions del mini-PC

Aquest incorpora un processador i3, i una targeta gràfica integrada (de baix rendiment). El sistema d'emmagatzematge és amb disc durs SSD a escollir entre diferents capacitats. Aquest te un disc dur de 120 GB, suficients per incorporar un sistema operatiu (màxim dos, si és volgués posar Windows convivint amb el Ubuntu) i per als arxius que es desitgin posar.

Aquest mini-PC s'alimenta a 12V i té un consum de 17W. És la part més interessant d'aquest PC, que ofereix unes prestacions molt bones amb un consum elèctric molt baix, i no tindrem una descarrega de bateries molt ràpida.

3.2. Placa de control

La placa de control està dissenyada per la UdG per tal de controlar els motors del robot. Aquesta esta basada en un microcontrolador PIC, concretament el model dsPIC 33FJ256MC, 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 part de realitzar el control dels motors aquesta placa també realitza la lectura dels polsos dels encoders que porten incorporats els motors, la lectura dels corrents de cada motor, mitjançant cel·les Hall es mesura el corrent consumit per cada motor, la lectura del nivell de les bateries i per últim el circuit inclou fonts d'alimentació commutades de 3.3V i 5V per tal d'alimentar els múltiples dispositius del robot amb els qual podem fer el control de voltatge de la bateria i del circuit d alimentació.

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	24 x 12-bit @ 500 (ksps) 2-A/D
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 2. Especificacions del microcontrolador PIC

La comunicació entre la placa de control i el mini ordinador es fa amb trames de bytes variables sobre el protocol TCP/IP. La placa de control amb una estació router WiFi, te accés a xarxes inal·làmbriques que permeten una comunicació amb el mini-PC i accés al sistema ROS. Aquesta estació router té una IP fixa (192.160.1.8) i proporciona a la placa de control una IP fixa (192.168.1.18).

3.3. Motors i encoders

El BIGBOT és un robot de quatre rodes les quals són accionades per quatre motoreductors de corrent continu, independents entre ells ja que així permeten guanyar una gran agilitat en realitzar girs. Cada motor porta acoblat al seu eix un encoder òptic en quadratura d'alta resolució.

Els motors són uns servomotors de la casa Pittman. Per aconseguir un parell motor suficient per poder contrarestar el pes del robot i permetre una bona mobilitat hi ha instal·lats uns reductors d'engranatges que redueixen la velocitat de l'ordre 65.5 a 1, però amb els quals s'obté un parell nominal de 3.4Nm.

A la taula 3 es poden observar les especificacions dels motors i els encoders.

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
Encoder	
CPR	500
Canals 2 + índex	2 + índex

Taula 3. Característiques del motor i l'encoder

Els encoders, són els encarregats de llegir les velocitats de les 4 rodes. Disposen d'una resolució de 500 CPR (cicles per revolució), és a dir, per cada volta l'encoder llegeix 500 polsos, els quals seran suficients per llegir les velocitats que aconsegueix el robot que són de l'ordre de 0,4m/s.

Els encoders en quadratura es caracteritzen per tenir dos canals de sortida. Els polsos dels dos canals es troben desfasats 90º entre ells de forma que observant la successió dels flancs de pujada o baixada entre els canals es pot determinar el sentit de gir dels motors.

Per tal de poder determinar el sentit de gir de les rodes s'ha fet servir un circuit extern (basat en una vàlvula flip-flop) que pren com a entrada els dos canals de l'encoder i dona com a sortida un bit que indica el sentit de gir en tot moment.

3.4. Camara KINECT

Una càmera RGB-D és un a càmera RGB, la qual permetrà gravar imatges en color i que a més ens proporcionarà la informació sobre la seva profunditat.

La càmera RGB-D que hi ha instal·lada al robot és la càmera KINECT. La càmera KINECT, va ser desenvolupada per Microsoft l'any 2010 com a accessori de la consola XBOX 360.

Però degut al potencial de combinar una càmera RGB i sensors de profunditat per projecció d'infrarojos s'ha convertit en un dels sensors visuals més utilitzats en el món de la robòtica.

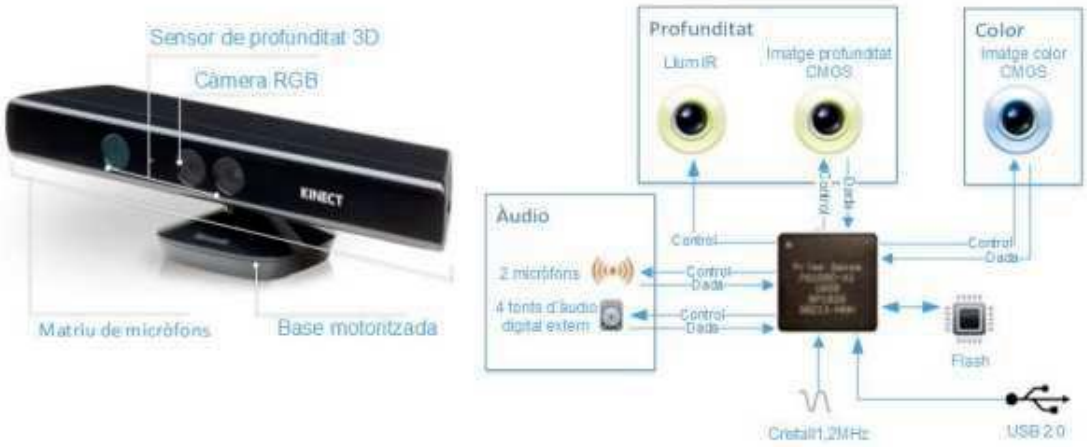

Figura 4. Camara KINECT

Per determinar la profunditat de la imatge, la càmera RGB-D incorpora un sensor de projecció per punts d'infraroig. Aquest sensor emet un seguit de punts a l'espai formant un patró de molts punts per trobar la profunditat de d'una figura respecte a un fons del mateix color. A partir d'aquest, amb l'ajuda de la càmera RGB, s'utilitzen càlculs trigonomètrics per obtenir la detecció dels objectes i la profunditat dels quals respecte el seu fons.

Utilitzant aquestes càlculs, la KINECT genera les imatges en color, les imatges en blanc i negre, la profunditat dels objectes, una visualització 3D de les imatges, unint les propietats de la càmera RGB i la projecció infraroja de punts.

3.5. Bateries

Les bateries del robot estan formades per dos grups de bateries les quals consten de dos cel·les de cinc bateries connectades en sèrie. Les bateries són de níquel – metall Hidrur de mida estàndard LR20 de 10 Ah i 1,2 V cadascuna, per tant, proporcionen una tensió de 12V en total.

Un dels grups de bateries s'utilitza per a alimentar la placa de control, els quatre motors amb els encoders i les cel·les Hall.

L'altre grup de bateries s'utilitza per alimentar el mini ordinador, la càmera tèrmica i la càmera KINECT. Aquest grup de bateries està connectat a un regulador de voltatge de 12V perquè el nivell de tensió que entregui al mini ordinador sigui estable als 12V.

La lectura del nivell de les bateries es realitza a traves d'un convertidor AD del dsPIC de la placa de control.

3.6. Giroscopi

Mitjançant el giroscopi es pot obtenir amb molta precisió la posició angular del robot respecte al camp magnètic de la terra. Així mateix. també en permet obtenir en diversos formats la seva acceleració i velocitat angular en els tres eixos. Aquest sensor es troba connectat al robot mitjançant un X-Port. En aquest cas és possible connectar-s'hi de dues formes com mostra la figura 5.

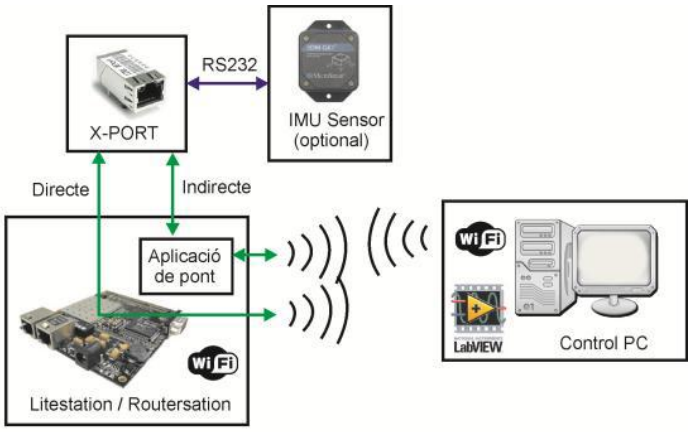

Figura 5. Comunicacions giroscopi

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

4. SOFTWARE DEL BIGBOT

El BIGBOT consta d'un ordinador el qual li proporciona autonomia, aquest funciona amb el sistema operatiu Ubuntu i per fer funcionar el robot s'ha instal·lat dins d'aquest la plataforma ROS que és un sistema operatiu robòtic. En aquest capítol es farà un resum de les característiques dels dos sistemes que conformen l'arquitectura del BIGBOT.

4.1. Ubuntu

Ubuntu és un sistema operatiu basat en el sistema GNU/Linux i que es distribueix com a programari lliure, que inclou el seu propi entorn d'escriptori anomenat Unity Ubuntu al ser un sistema operatiu de codi lliure disposa de molt material de codi compilat distribuït per Internet. És un sistema operatiu més obert que Windows, cosa que permet modificar aspectes interns si fos necessari. Pel BIGBOT s'ha escollit la versió Ubuntu 12.04LTS.

Per realitzar la comunicació entre l'usuari i el sistema operatiu, s'utilitza un intèrpret d'ordres.

Un intèrpret d'ordres és un programa que utilitza l'entrada de l'usuari, per exemple les ordres que tecleja, i la tradueix a instruccions. Aquest sistema operatiu utilitza un intèrpret de comandes anomenat Terminal. A la següent taula es poden observar les principals comandes utilitzades per la realització d'aquest projecte.

cd	Canvi de directori
echo	Mostrar un text per pantalla
sudo	Donar permisos d'administrador
ping	Permet comprovar una connexió
rm	Elimina un arxiu
export	Assigna valor a una variable d'entorn
killall	Permet finalitzar processos

Taula 5. Comandes per el terminal

Per fer el control del robot un cop esta en funcionament, es necessària la utilització d'un ordinador extern pel control, mitjançant una comunicació remota.

La comunicació remota es fa a través de xarxes sense fils WiFi sense la necessitat de connectivitat a Internet que treballa sota el protocol SSH. El SSH o intèrpret de comandes segura, és un protocol que serveix per permetre una comunicació remota entre dos equips a través d'una xarxa.

El SSH permet assegurar, en tot moment, una comunicació segura (xifrada) entre client i servidor, sent el robot el servidor i l'ordinador extern de control el client, això permet executar aplicacions gràfiques, és a dir, obrir una finestra a l'ordinador remot i veure-la a l'ordinador que ha iniciat la connexió, com seria el veure una imatge produïda per la càmera KINECT.

Per iniciar una connexió remota hem d'introduir el nom d'usuari i la direcció IP de l'ordinador al que ens connectarem, un cop introduïts els valors s'haurà d'introduir la contrasenya i veurem que la ruta del terminal varia i es converteix en la ruta de l'ordinador remot.

$ssh –X NOM_USUARI@DIRECCIÓ_IP

Un cop s'ha executat aquesta comanda ja es poden activar i desactivar tots els processos al robot. Hem de tenir en compte que els processos els hem de cridar en un nou terminal, aquest no te la sessió SSH activa, per tant la hem de tornar a iniciar, també hem de tenir present que un cop es tanqui el terminal aquesta sessió acabarà.

4.2. ROS

ROS és una plataforma de treball, la qual esta basada en el sistema operatiu Ubuntu, que està pensada pel desenvolupament del software per a robots. Aquesta plataforma proporciona la funcionalitat d'un sistema operatiu en un clúster heterogeni, permeten l'agrupació de varis ordinadors dins d'un mateix sistema operatiu per distribuir la computació, aconseguint així repartir la carrega. També permetrà la diversificació de hardware.

ROS és una plataforma de codi lliure, el qual permet una adquisició lliure del producte i un ús tant comercial, com per a la investigació. Aquest fet ha generat una gran divulgació i intercanvi de programes, tant per aficionats com per professionals d'aquest sector. Per facilitar aquest intercanvi, els desenvolupadors de ROS han realitzat una

wiki on s'incorporen uns tutorials, llibreries i paquets creats pels desenvolupadors i els usuaris de ROS.

ROS s'executa en sistemes operatius del tipus UNIX, principalment està pensat per executar-se en Ubuntu. Actualment, altres sistemes operatius com Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian, Windows i Android estan en fase experimental. Una característica important de ROS, és la diversitat de llenguatges de programació que admet, com el C++, Python i Lisp quan s'han de desenvolupar els diferents programes.

ROS està organitzat en dues parts principals: la part del sistema operatiu, i ROS-pkg (Packages, paquets en català).

Els Packages són unitats d'organització del software del ROS, els quals implementen les funcions pel qual han sigut dissenyats. En aquests paquets hi ha continguts arxius de configuració, arxius executables, llibreries o conjunts de dades. Dins els paquets hi ha tres elements principals: els Nodes, els Tòpics, i els Services. També es pot trobar organitzats en Stacks (piles), el quals són agrupacions de paquets que tenen una mateixa funcionalitat

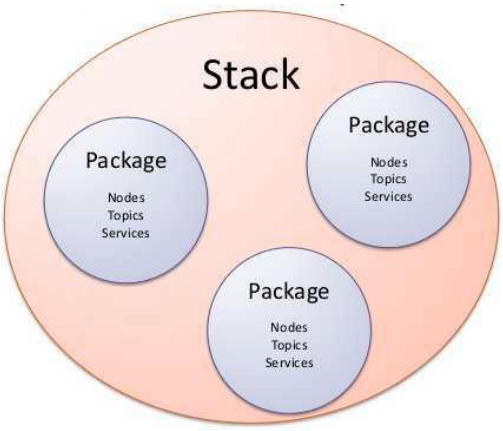

Figura 6. Estructura d'un Stack

Els nodes són arxius executables, els quals realitzen els càlculs i s'utilitzen per comunicar-se amb els altres nodes per tal d'enviar o rebre informació en forma de missatges. Per tal de realitzar aquesta comunicació, els nodes han de publicar i subscriure's, és a dir, escriure o escoltar, als diferents tòpics. Els tòpics són els busos

pels quals els nodes intercanvien els missatges o la informació. Quan un node publica un missatge, aquest s'inclou en un tòpic, d'aquesta manera quan un segon node vulgui llegir una informació haurà de subscriure's al tòpic en qüestió. Per entendre millor que són els nodes i els tòpics, els nodes equivalen als usuaris de Twitter i els tòpics llavors seria un "hashtag" en el qual usuaris envien els missatges. Els Services (o serveis) són els encarregats de definir l'arquitectura de la comunicació entre els diferents nodes. El servei que més s'utilitza és l'arquitectura client/servidor, on el node servidor es manté a l'espera fins que el node client li realitza una sol·licitud, és a dir, fins que el client li envia un missatge, tot seguit el servidor realitza un procés i respon al client.

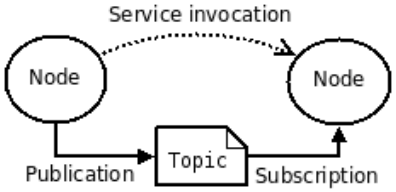

Figura 7. Estructura comunicació

Per la utilització del sistema ROS en Ubuntu, ROS disposa de comandes per facilitar l'ús a l'usuari, moltes d'elles tenen semblança amb les normalment usades en UNIX. Les principals comandes són les descrites a la taula 6.

Comanda	Descripció
roscd	Cd a la ruta origen del workspace
rospack	Permet gestionar els paquets
rosnode	Permet gestionar els nodes
rostopic	Permet gestionar els tòpics
rosrun	Executa un node
roslaunch	Executa una sèrie de processos
roscore	Executa el sistema ROS

Taula 6. Comandes bàsiques del ROS

La comanda roscore és la primera comanda a executar ja que és la comanda que realitza el inici del sistema ROS. Les altres comandes només funcionen completament amb el roscore actiu.

Les comandes rostopic i rosnode gestionen el tòpics, aquestes ens permeten conèixer la informació sobre elles i en el cas dels nodes permet desactivar-los. Les principals comandes que són les descrites a la taula 7.

Comanda bàsica	Comanda secundaria	Descripció
rosnode	kill	Desactiva un node
rosnode o rostopic	info	Imprimeix la informació sobre un nodes o un topics
rosnode o rostopic	list	Permet visualitzar els nodes o els topics en una llista
rostopic	echo	Mostra el valor d'un topic per la pantalla
rostopic	find	Busca un topic concret

Taula 7. Comandes per la gestió de nodes i topics

Per el que fa al robot, la versió de ROS que hi ha implementada és la Groovy Galapagos.

Aquesta és la sisena versió desenvolupada d'aquest sistema operatiu. Va ser llançada el 31 de desembre de 2012. Aquesta versió de ROS ha esta llançada per al sistema operatiu Ubuntu 12.04 (Precise).

En aquesta versió els desenvolupadors de ROS s'han centrat en la infraestructura central de ROS perquè sigui més fàcil de fer anar, més modular, més escalable, el treball a través d'un major nombre d'arquitectures de maquinari i robots i el més important, en aquesta versió, és que és més fàcil involucrar a la comunitat de ROS pel fet de que en aquesta versió s'ha fet una migració massiva de codi a GitHub, el qual és una plataforma de desenvolupament col·lectiu en la qual es poden fer reposicions de projectes i que ajudarà a que sigui més senzill el fet de compartir paquets entre els usuaris de ROS.

5. REALITZACIÓ

En aquest apartat es planteja els passos a seguir per fer la programació del robot. Parlarem de la part de comunicació entre el robot i el ordinador, i dels assajos fets.

5.1 Assajos

Els assajos que hem fet ens serveixen per començar a fer el projecte, aquests s'han realitzat prèviament. Tots els assajos han sigut dies de fer probes, rectificant consignes quan el robot no feia la trajectòria que volíem.

Per poder realitzar els assajos necessitem tres parts: el robot, un ordinador portàtil i una base amb cable Ethernet.

El objectiu del robot en aquestes probes es el de captar dades, es el encarregat de enviar les dades al ordinador perquè aquest les pugui llegir i guardar, així més endavant les podem processar i utilitzar amb el MATLAB, gracies al programa que hem comentat anteriorment.

El ordinador portàtil consta de un script fet amb LabView que ens permet controlar i dirigir el robot com volem, per poder fer anar el robot hem d'executar el programa, aquest té un panell frontal que utilitzarem per captar dades i donar les consignes de velocitat que volem al nostre robot, també podrem donar la ordre de si volem que faci un gir a una certa velocitat, o si volem que vagi recte amb el frenat. Un cop tenim totes les consignes introduïdes com les necessitem podem posar en Run el programa, aquest farà que el robot es mogui, un cop després de diferents intents tindrem totes les dades amb diferents velocitats del terreny on estàvem fent els assajos, tant de la trajectòria recta amb frenat, com del gir de 90º.

La base ens dota de Internet al ordinador portàtil, mitjançant un cable ethernet, que permet la comunicació del ordinador amb el robot. Al ser una base mòbil ens dona molt més rang d'actuació poden agafar dades de terrenys més allunyats del laboratori, ja que la connexió WiFi que hi ha al laboratori te un rang petit.

Les probes s'han realitzat als diferents terrenys que s'han trobat al campus de la universitat. Hem tingut en compte de seleccionar terrenys amb un desnivell casi zero, o sigui terrenys plans.

5.2 Càlcul del coeficient del terreny

Per poder realitzar el càlcul del coeficient del terreny necessitarem, primer de tot les dades del diferents terrenys que hem agafat prèviament en els assajos, després necessitarem el MATLAB per poder realitzar els càlculs.

Les dades dels assajos les tindrem guardades en format .txt, per tant utilitzarem el script fet en MATLAB per extreure les rectes dels terrenys.

A continuació vam fer un petit programa amb el MATLAB per poder extreure el valor del coeficient del terreny.

En aquest cas les dades que necessitarem i utilitzarem seran les del gir, ja que ens proporcionaran la suficient informació per fer el nostre càlcul. Primerament plantejarem les fórmules que utilitzarem, després mirarem la manera de com les passarem al MATLAB per realitzar el programa.

El coeficient del terreny és important, primer per l'error que genera i segon perquè en molts casos el gir amb el control de velocitat PID no és òptim.

Obvien la millora en el desplaçament recte, ja que es produeix un error molt baix, això és pot notar amb l'equació plantejada, ja que simplement es fa una mitja sobre el desplaçament recorregut per cada roda.

$$d=\frac{dR-dL}{2} \qquad \text{(Eq.1)}$$

És convenient explicar primer els càlculs d'odometria per a un robot de tipus rotació pura sense lliscament, anomenats differential drive. Són robots amb bandes de rotació d'esquerra i dreta en el pla, és a dir, amb dues rodes, una per cada banda.

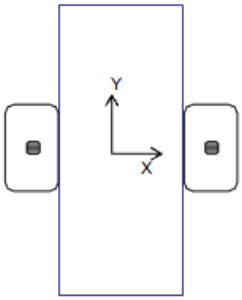

Figura 8. Vehicle de tipologia differential drive

El direccionalment ve donat per la diferència de velocitats de les rodes laterals. La tracció s'aconsegueix també amb aquestes mateixes rodes. Aquest sistema és molt útil presentant la possibilitat de canviar d'orientació sense moviment de translació. Les variables de control d'aquest sistema són les velocitats angulars de les rodes esquerra i dreta.

El procés experimental sobre aquests tipus de robots respon a la fórmula següent:

$$\emptyset = \frac{dR - dL}{B} \qquad \text{(Eq.2)}$$

on dR és el desplaçament de la banda dreta, dL és el desplaçament de la banda esquerra, B la distància entre les rodes de cada banda i ∅ l'angle de rotació en radiants.

El càlcul del gir en radiants es pot entendre senzillament amb la lectura que fan els encoders a les rodes del robot. Agafarem l'exemple de la roda dreta, la qual seguint l'equació següent s'obté el desplaçament total.

$$\Delta dr = \frac{Nint}{Nvolta} \cdot 2\pi r \qquad \text{(Eq.3)}$$

N_{int} són els polsos de la roda dreta, N_{volta} són els polsos totals en un gir i 2πr és el perímetre de la roda. Aquest és el càlcul que fan els encoders que porten acoblat al eix de cada motor. D'aquesta manera es pot determinar el desplaçament de cada roda.

Tot i que aquest sigui el càlcul que es produeix a les rodes gràcies als encoders, a l'hora de la lectura, el robot genera un document informatiu on no s'especifica el desplaçament recorregut, per això pel càlcul del desplaçament s'ha d'agafar la lectura de les velocitats per cada interval de temps de 50 ms.

$$d_{int}=v_{int.} \cdot \Delta t \qquad \text{(Eq.4)}$$

Per tant el desplaçament total serà la suma dels desplaçament de cada interval corresponent a l'equació 4.

Aquest apartat del present projecte té com a objectiu l'estudi teòric per la millora en temps real del control tenint en compte els efectes del desplaçament. Així com pels robots de rotació pura no cal diferenciar el terreny, pel tipus de robot amb el qual treballem pel projecte es necessari, mitjançant la millora de l'equació 3 proposada per Anthony Mandow i Wei Yu amb les seves tècniques experimentals.

Segons els autors esmentats abans, aquest estudi tracta sobre la millora del moviment en rotació del robot depenent del terreny. Per realitzar la millora, es buscarà el factor de correcció pel control d'orientació que s'escau a cadascun dels entorns proposats al projecte.

Com hem pogut comprovar abans, hi han moltes oscil·lacions a les rodes que tenen més tracció i que per 50 cm/s per alguns terrenys no s'arriba a la consigna. Per tant, al no aconseguir un gir òptim, serà interessant el control de l'orientació del robot.

Aquest estudi es basa en el control de les velocitats relatives de les unitats del costat esquerra i dret.

El robot del qual disposem és de tipus "skid-steer", ja que es disposen vàries rodes a cada costat del vehicle (en aquest cas dues rodes per banda) i actuen de forma simultània. El moviment és el resultat de combinar les velocitats de les rodes de l'esquerra amb les de la dreta.

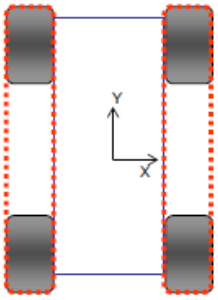

Figura 9. Vehicle de tipologia skid-steer

El procediment teòric per trobar el factor de correcció s'ha regit sota l'equació 5, una millora de l'equació 2.

$$\emptyset = \frac{dR - dL}{\alpha \cdot B} \qquad \text{(Eq.5)}$$

on dR, dL és el desplaçament de cada roda, ø és l'angle de rotació en radiants, B la distància entre les rodes de cada banda i α el factor de correcció a trobar per cada terreny. Els assajos fets per cada terreny ha estat un gir de 90º amb les diferents velocitats de treball (20 cm/s, 30 cm/s, 40 cm/s i 50 cm/s).

Un cop fet els assajos, amb un arxiu MATLAB es calculen les distàncies recorregudes per cada roda. Com que el robot és de tipus "skid-steer" amb dues rodes per banda s'ha fet una mitjana combinant les distàncies recorregudes de les rodes esquerres i el mateix procediment s'ha fet per les rodes de la dreta. Per fer el càlcul de les distàncies s'ha aprofitat la lectura dels encoders de les rodes. Seguint l'equació següent s'obté el desplaçament de la roda.

$$d = v \cdot \Delta t \qquad \text{(Eq.6)}$$

Tenint en compte que la lectura dels encoders es produeix amb un temps de mostreig de 50ms, s'ha de sumar la mitja del desplaçament de les dues rodes de cada banda realitzat per a cada lectura fins que el robot hagi aconseguit el gir complet de 90º.

$$\text{distancia} = \frac{v_{roda\ davant} + v_{roda\ darrera}}{2} \cdot t_{mostreig} \qquad \text{(Eq.7)}$$

Un cop obtingut el desplaçament total fet per cada roda, aplicant l'equació 8 s'obté el factor de correcció que s'ha d'aplicar a cada terreny.

$$\text{Alfa} = (\text{dR} - \text{dL})/(\text{B}\cdot(\text{Angle}\cdot\pi/180)) \qquad \text{(Eq.8)}$$

Els assajos han consistit en fer girar el robot 90º a les diferents velocitats, agafant la lectura de les diferents velocitats en tot el temps en el que el robot ha tardat en fer el gir. Un cop obtinguda aquesta lectura s'ha procedit a fer el càlcul de la distància total recorreguda per cada banda del robot (banda esquerra i banda dreta).

Terrenys	Velocitats			
	20	30	40	50
Herba	1,7662	2,1358	2,1583	1,9603
Asfalt	2,0385	2,0811	2,0593	2,2744
Ciment barracons	2,0429	2,0235	2,0513	2,0270
Terra PII	2,0566	2,1410	2,0670	2,0805
Mosaic PII	-	2,1182	2,0931	2,0390
Pedra gran Pati	2,3010	2,3060	2,3517	2,3856
Interior laboratori	2,0128	2,0467	2,0527	2,0347
Pedra barracons	2,7378	2,5014	2,6091	2,6532
Sorra laboratori	3,2887	3,1532	2,9108	3,1984
Interior laboratori	2,5869	2,0326	2,1051	2,9901

Taula 8. Valors d'alfa

Després d'obtenir els diferents desplaçaments, fent ús de la equació 2, s'han pogut trobar els diferents factors de correcció α reflectits en la taula 8.

5.3 Placa de control

La placa de control del robot està controlada per un programa anomenat base_controller, aquest es el principal programa i el més important, ja que activa la placa del robot. Nosaltres hem afegit petites coses a la placa per poder utilitzar el nostre programa. El que hem afegit al programa es pot visualitzar a l'annex A anomenat parts del base_controller. Per començar hem tingut que introduir les variables de la distancia del polsos. En el annex és la part inicialització variable dist.

```
int dist1L;

int dist1R;
```

Aquesta part inicialitza les variables dist de cada roda i per cada terreny, El terreny es indicat per el número i la roda per la lletra majúscula, aquesta variable ens servirà més endavant per situar el punt on estem segons els terrenys que hi han.

La següent cosa que hem afegit son variables que te cada terreny, aquestes estan guardades en el programa i les compara amb el terreny actual, i per proximitat sap a quin terreny estem. Aquesta part a l'annex és anomenada inicialització variables terreny.

```
float a1L;

float b1L;

float c1L;
```

El número com abans indica el terreny que es, i la lletra en majúscula indica la roda com hem dit anteriorment. El número varia segons el terreny.

A continuació posem el valor de cada variable del terreny, això ho hem de fer per cada superfície que tenim. Aquesta part esmentada es troba al annex com a dades terreny.

Seguidament hem de utilitzar la variable dist perquè ens servirà per saber la distancia dels polsos entre cada terreny i el actual, simplement es per informació. Per poder veure aquesta informació, hem fet l'apartat que s'anomena visualitzacio de la distancia entre terrenys a l'annex.

Un cop fet això podrem veure per pantalla la distancia que tenim per cada terreny.

Per acabar l'última cosa que hem afegit es la part del missatge que surt en la pantalla on ens diu en quin terreny està situat. Per fer-ho hem utilitzat un switch case, per posar tots els terrenys, i que segons els resultats obtinguts mostri un missatge o un altre. Al annex està esmentat com a missatge tipus de terreny.

Un cop fet això podem veure que si les dues rodes no coincideixen en quin terreny està et mostra un missatge d'error i torna a fer la prova, si ho detecta correctament ens mostrarà el nom del terreny que es.

5.4 Proves de gir

Una de les parts més importants són les proves amb el programa de detecció funcionant i un petit programa de circuit on podrem veure si fa be el gir de 90º o s'apropa molt. El codi complet està a l'annex D anomenat Programa Turn90.

Per començar primer hem de posar les dades extretes del MATLAB en el programa principal que utilitza el robot, en aquest cas es diu base_controller. Primer de tot hem fet la declaració de les variables dels terrenys nous, en aquesta part de codi és veuen les dades d'un dels terrenys en recte.

```
//Pedra

            a9L = 14.1223;

            b9L = -1;

            c9L = 320.1320;

            a9R = 12.0470;

            b9R = -1;

            c9R = 475.5917;
```

Un cop posades les dades i declarat totes les possibles variables, ja podem fer el programa de les proves.

El programa de proves es centra principalment, en l'execució correcta del gir de 90º en els diferents terrenys que tenim. Per començar tenim una inicialització de variables amb la utilització dels subscriptors i dels publicadors, com indica la programació en ROS, la part de inicialització és la figura 10:

```
#include <ros/ros.h>
#include <iostream>
#include <math.h>

#include <std_msgs/Float32.h>
#include <sensor_msgs/Joy.h>
#include <geometry_msgs/Twist.h>

using namespace std;

//variable for storing the number of counts of the encoder
int counter = 0;
int totalCount = 0;
bool turnFlag = false;
bool turnFinished = false;
int terrain_type = 1;

bool gostraight = true;
int travelled = 0;
int turns = 0;

void Enc_R_Callback(const std_msgs::Float32& msg){
    counter = msg.data;
}

void terrainCallback(const std_msgs::Float32& msg){
    terrain_type = msg.data;
```

Figura 10. Inicialització

```
void Enc_R_Callback(const std_msgs::Float32& msg){
    counter = msg.data;
}

void terrainCallback(const std_msgs::Float32& msg){
    terrain_type = msg.data;
}

void joyCallback (const sensor_msgs::Joy::ConstPtr& joy_data){
/* This function responds to any change on the state of the joystick. It
 * is mainly used to detect when the buttons are pressed and act according
 * the one that has been pressed
 */
    if (joy_data->buttons[5] == 1){ //this button will restart the PID
        turnFlag = true;
    }
    if (joy_data->buttons[6] == 1){
        turnFlag = false;
        cout <<  "Boto 4: desactiva" << endl;
}
```

Figura 11.Subscriptors i botons

El programa l'hi hem posat una part d'activació i desactivació per poder controlar una mica el robot, també ens servirà per si aquest es desvia o esta a punt d'impactar amb algun objecte, així podem parar l'execució del programa i posar-lo bé o en un lloc sense perill. Poder activar i desactivar el programa també ens serveix per poder rectificar el codi si veiem que al principi ja no funciona correctament, sense tenir la necessitat d'esperar a que finalitzi el programa.

En el cos del programa o la part principal trobem un *case* amb tots els coeficients del terreny, també tindrà una variable anomenada *threshold* que vindrà del

base_controller. Hem afegit un contador perquè ens indiqui el nombre de polsos que fa en cada gir.

Com podem veure a la següent figura aquesta seria la part esmentada.

```
void turn(){

    float threshold;
    int totalpulse = 2100;
    cout << "Counter: " << counter << " totalCounter: " << totalCount << endl;
    totalCount = totalCount + counter;

    switch (terrain_type){

        case 1:
            threshold = totalpulse*2.05;
            break;
        case 2:
            threshold = totalpulse*2.99;
            break;
        case 3:
            threshold = totalpulse*2.91;
            break;
        case 4:
            threshold = totalpulse*2.16;
            break;
        case 5:
            threshold = totalpulse*2.09;
            break;
        case 6:
            threshold = totalpulse*2.059;
            break;
        case 7:
            threshold = totalpulse*2.05;
            break;
        case 8:
            threshold = totalpulse*2.15;
            break;
        case 9:
            threshold = totalpulse*2.15;
            break;

    }

    if (totalCount > threshold){
        turnFinished = true;
    }
}
```

Figura 12. Cos del programa

Un cop feta aquesta part només ens queda indicar la trajectòria que volem fer, en aquest cas volem fer un quadrat per poder veure si el robot fa els 90º correctament o se'ns desvia molt.

Aquesta part del programa es bastant senzilla ja que nomes consta de un *while* on indiquem si va recte o ha de girar. Indiquem la velocitat en el eix de les x que es 40cm/s i posem una variable de distancia recorreguda que es diu *travelled*. Aquestes ens serviran per anar rectes.

```
cmd_vel.linear.x = 0.4;
travelled = travelled + counter;
                        cout << travelled << endl;
                        if (travelled > 5000){
                               gostraight = false;
                        }
```

A la part del gir només hem de posar la velocitat angular en el eix z, i treure la linear anterior. I també hem afegit un contador de girs perquè fa més d'un gir.

Un cop acabat el programa el podrem executar en el ordinador amb el ROS. Per fer-lo funcionar primer haurem de compilar el node que hem guardat amb el nom *circuit*, després d'acabar la compilació podrem posar-lo en funcionament.

Per començar a executar aquest programa haurem de fer l'activació de la placa de control, i l'activació del programa de detecció de terreny. Per fer l'activació de la placa de control del BIGBOT, s'haurà d'executar el seu node corresponent, el qual es el base_controller, que ens servirà per fer el control dels motors i els encoders. Llavors per executar aquest node s'haurà de introduir la següent comanda al terminal Ubuntu.

```
rosrun projectbigbot base_controller
```

En el cas de la utilització del ordinador Arlab utilitzat al laboratori de sistemes intel·ligents de forma remota, per fer la activació d'aquest node se realitzarà a traves de la comanda inici master, el qual fa la activació del ROS tan al BIGBOT com al ordinador remot i executa el node de la placa de control. A més, aquesta comanda activa un node per el control teleoperat mitjançant un joystick des de l'ordinador Arlab, amb el qual podem controlar el moviment del BIGBOT en cas de que sigui necessari. També s'ha de fer la activació de detecció del terreny i del circuit tal com s'ha dit anteriorment, mitjançant el node check_terrain i circuit.

Finalment tindrem el programa funcionant amb tot el que necessitem per fer les corresponent proves. Per tal de garantir un bon funcionament del check_terrain i la

posterior visualització dels resultats de les proves, quan s'estigui treballant en els assajos, s'ha de mantenir els nodes actius en el terminal on s'hagi executat la comanda anterior. Quan el programa circuit s'està executant podem visualitzar el nombre de girs que esta fent en cada moment i el nombre de polsos que fa en cada gir, això ens servirà per afinar el programa, per acabar tenint la versió final d'aquest.

5.5 Gir circular

El nostre robot també pot girar de forma circular com si fos un cotxe, hem volgut fer una prova d'aquest gir circular per veure si els coeficients que hem obtingut anteriorment serveixen per aquest nou gir o no.

Hem fet més assajos i hem agafat les dades amb el LabView, hem fet proves en tres superfícies que son el interior del laboratori, la grava de fora i la sorra. Els assajos consistiran en fer un gir circular en radis diferents. Primer farem amb una velocitat mitjana de 40cm/s i després 30cm/s. Les dades que agafem les introduirem al programa en MATLAB que tenim i d'aquí obtindrem el coeficient per cada terreny. El resultats dels coeficients son els següents:

Interior Lab		
Velocitat		Alfa
Esquerra	Dreta	
10,00	50,00	2,441
20,00	40,00	2,172
23,00	37,00	2,466
25,00	35,00	2,106
30,00	50,00	2,447
33,00	47,00	2,448
35,00	45,00	2,054

Taula 9. Valors alfa gir circular interior

Exterior Lab		
Velocitat		Alfa
Esquerra	Dreta	
35,00	25,00	1,038
37,00	23,00	1,350
40,00	20,00	1,806
45,00	35,00	1,089
47,00	33,00	1,577
50,00	10,00	2,589
50,00	30,00	1,875

Taula 10. Valors alfa gir circular exterior o pati

Sorra		
Velocitat		Alfa
Esquerra	Dreta	
45,00	15,00	2,321
50,00	10,00	2,613
50,00	30,00	2,157
55,00	25,00	2,276

Taula 11. Valors alfa gir circular sorra

A la sorra tenim poques velocitats perquè el espai que teníem de sorra es petit i només podíem fer els girs circulars de radi més petit, però hem pogut veure que les alfes eren un pel mes baixes.

Els coeficients que tenim són el interior del laboratori, el exterior o pati i la sorra, aquests terrenys els utilitzarem per fer el mateix circuit que el gir sobre si mateix, però amb el gir circular on esperem que el moviment sigui més fluït.

Per finalitzar dir que les alfes que hem trobat son molt semblants a les que teníem calculades anteriorment en el gir de 90º sobre el propi eix. El únic cas que tenim que varia molt el valor es en el terreny del pati exterior, on el seu valor es més petit que el que teníem, això pot ser degut a que les rodes en aquest tipus de gir troben menys resistència i llisquen més en aquesta superfície, en canvi en les altres dues superfícies els valors son molt semblants. Amb la sorra passa el mateix, el valor baixa, però no es tan exagerat com el del exterior, i es per la mateixa raó que el cas del pati per la resistència que es més baixa en aquest tipus de gir.

5.6 Velocitats

Les velocitats amb les que s'ha treballat han estat a 20 cm/s, 30 cm/s, 40 cm/s i 50 cm/s i amb la seva respectiva reducció del 20% a l'hora de la detecció del terra.

S'ha considerat que el robot a 20 cm/s és la velocitat mínima a la que el robot pot funcionar ja que per velocitats inferiors es podria donar el cas que a causa de la fricció de les rodes amb el terreny el robot no es mogués. Per altra banda la velocitat màxima de treball considerada ha estat 50 cm/s ja que depenent de l'entorn el robot entra en saturació i no pot anar més ràpid.

Aquesta saturació es pot notar en la lectura del senyal de control la qual ens dóna el % del voltatge aplicat al motor. Quan el robot està funcionant sobre recte, el senyal de control per a tots els terres a 50 cm/s pren un valor aproximat del 70% del voltatge, cosa que indica que encara pot anar més ràpid. El problema arriba quan el robot ha de girar, en el terra de mosaic i en el terra d'herba aquest senyal de control a 50 cm/s arriba al 100% i per tant ens indica que per a velocitats més altes ja no pot arribar a la velocitat desitjada i per tant efectuarà un gir no òptim. Per altra banda als únics terrenys que a 50 cm/s el senyal de control no és del 100% és a la rajola i la sorra, on està al voltant del 88%-98%, però s'ha volgut agafar com a velocitat màxima per tots els terrenys.

Velocitat [cm/s]	Reducció 20% [cm/s]
20	16
30	24
40	32
50	40

Taula 12. Velocitats de treball

Per dur a terme la detecció de terra el robot redueix un 20% la velocitat de les rodes del darrera. La necessitat de la reducció d'aquest percentatge de la velocitat és perquè per intensitats dèbils seria complicat aconseguir una variació d'intensitat entre els terrenys. Al frenar aquest 20% a les rodes del darrera, el robot tindrà més parell resistent aconseguint més tracció i en conseqüència les rodes del davant consumeixen més intensitat i això provoca que es pugui detectar diferències entre els diferents terrenys.

La intenció de l'aplicació és que quan el robot detecti un canvi significant en la seva velocitat i per tant hi hagi un possible indici d'una transició d'un terra a un altra, pugui detectar el terreny en el que està navegant.

Per dur a terme la detecció de terra es treballa amb el controlador PID de sèrie on la contant Kp=0,0003, la constant Ki=0,05 i la constant Kd=0.

5.7 Programació en MATLAB detecció de terrenys i alfes de terreny

L'aplicació de la detecció de terra feta amb MATLAB ha estat realitzada pels terres on el robot ha funcionat gràcies als assajos però està totalment oberta per a possibles modificacions que suposin afegir nous entorns que el robot pugui conèixer. Aquesta aplicació és fàcil de modificar ja que és molt intuïtiva i senzilla. El codi està a l'annex amb el titol de programació de Matlab.

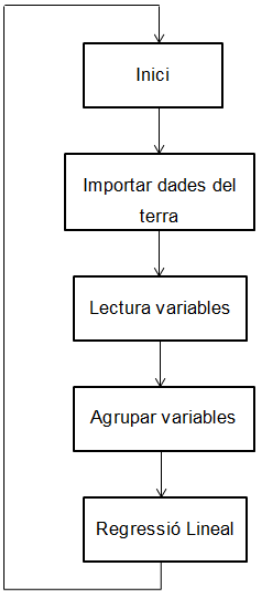

Figura 13. Programació detecció terra

Tal i com es mostra a l'organigrama, no es disposa de cap transició ja que l'estudi està realitzat en estat Offline.

L'estructura de l'aplicació es pot diferenciar en 4 blocs seqüencials començant per la importació de les dades, seguidament de la lectura de les variables necessàries per dur a terme la detecció, l'agrupació de variables i la regressió lineal que és el resultat final de l'aplicació.

S'ha escollit com a tècnica, per estudiar la relació de les variables, la regressió lineal. Ja que s'adapta a una gran varietat de situacions, entre altres per predir diferents aspectes en el comportament del robot.

El programa importa les dades dels diferents terres d'assajos des d'un arxiu de disc on s'ha guardat la simulació i tot seguit les gestiona agafant la lectura de les variables que es necessiten per la diferenciació d'aquests terrenys. El codi està al annex amb el titol de programació de Matlab. Aquesta part importa les dades.

```
Imp=importdata('BB3_asfalt_20_16.txt');

Data=Imp.data;
```

Les variables que es fan servir per la detecció de terra són les velocitats, els corrents i el senyal de control de les dues rodes del davant del Bigbot. També agafa la lectura del nivell de bateria del robot mòbil la qual també es farà servir. Aquesta petita part de codi detecten les velocitats i els corrents.

```
speed_FL = Data(101:200,2);

curr_FL = Data(101:200,6);
```

Aquestes variables són vectors que emmagatzemen dades cada 50 ms. Com que el treball realitzat ha estat en mode Offline en els assajos realitzats, a l'hora d'agafar les dades, s'ha treballat amb un temps de 10 segons de simulació per cada mostra.

La propietat Data (Dades) conté la simulació de les dades emmagatzemades. Aquesta propietat conté totes les dades registrades durant la simulació. Aquesta propietat és una matriu m x n (Data (m,n)), on m és el número de passos de temps i n és el número de quantitats registrades. Les files de la matriu estan etiquetats pels punts de temps en la propietat Time, i les columnes són etiquetades per les dades en la propietat Data.

Per tant les variables a llegir, fan la lectura a partir del pas de temps 101 fins el 200 ja que s'ha considerat que a partir d'aquest pas de temps la simulació ja està en règim estacionari i tot el règim transitori ja ha desaparegut. És a dir, és el temps en que el robot en la simulació de l'assaig, ha aconseguit arribar a la consigna i per tant, ja està en règim estacionari.

Per a l'agrupació de variables es creen setze vectors nous on s'adjuntaran totes les dades de les quatre velocitats dels diferents terres de treball amb un sol vector. Els vectors nous que es creen són quatre vectors per a l'agrupació de totes les velocitats (AllspeedsFL) dels diferents terres, quatre vectors més per a l'agrupació del nivell de bateria (Allbattery) en la simulació dels diferents terres, quatre vectors més per a l'agrupació del senyal de control (AllctlFL) dels diferents terres i quatre últims vectors més per a l'agrupació dels corrents (AllcurrentsFL) dels diferents terres en les seves respectives simulacions. En aquesta part agrupem les variables amb el for, i això es una petita part del codi.

```
for z=1:100

 %%Vectors de velocitats FL

 AllspeedsFL(z)=speed_FL(z);
```

Tal i com es veu en el codi, mitjançant una estructura for, recorrem els quatre vectors, un vector per cada velocitat, i les dades d'aquests les emmagatzemem al nou vector per tal d'agrupar-les. Per tant el vector AllspeedsFL, és un vector que hi conté el vector de la velocitat de 20 cm/s, el vector de la velocitat de 30 cm/s, el vector de la velocitat de 40 cm/s i el vector de la velocitat 50 cm/s.

Per acabar l'agrupació de dades, al final multipliquem els vectors AllctlFL, Allbattery i AllcurrentsFL entre sí per aconseguir el vector POT que ens servirà per trobar el resultat final. El que es vol aconseguir és una gràfica de les velocitats en funció de la potència.

Aquest treball és la potència que respon a la fórmula següent:

Pot=V*I=((PWM/100) * Bateria) * I=(AllctlFL * Allbattery) * AllcurrentFL (Eq.1)

Totes les figures segueixen el treball de programació del terreny rajola. A l'annex es pot veure tota l'aplicació al complet amb tots els terrenys treballats.

Finalment per dur a terme la detecció de terra, es fa una regressió lineal on en l'eix d'abscisses hi figuren les diferents velocitats (cm/s) i en l'eix y la Potència consumida (W).

Tal i com es veu a la figura 14, els terres estan diferenciats per colors i cada grup de punts és la lectura de la velocitat en funció de la Potència en el seu temps de mostreig quan el robot ja s'ha estabilitzat a la velocitat de consigna. Cada punt és una mostra agafada al fer-se la lectura cada 50 ms.

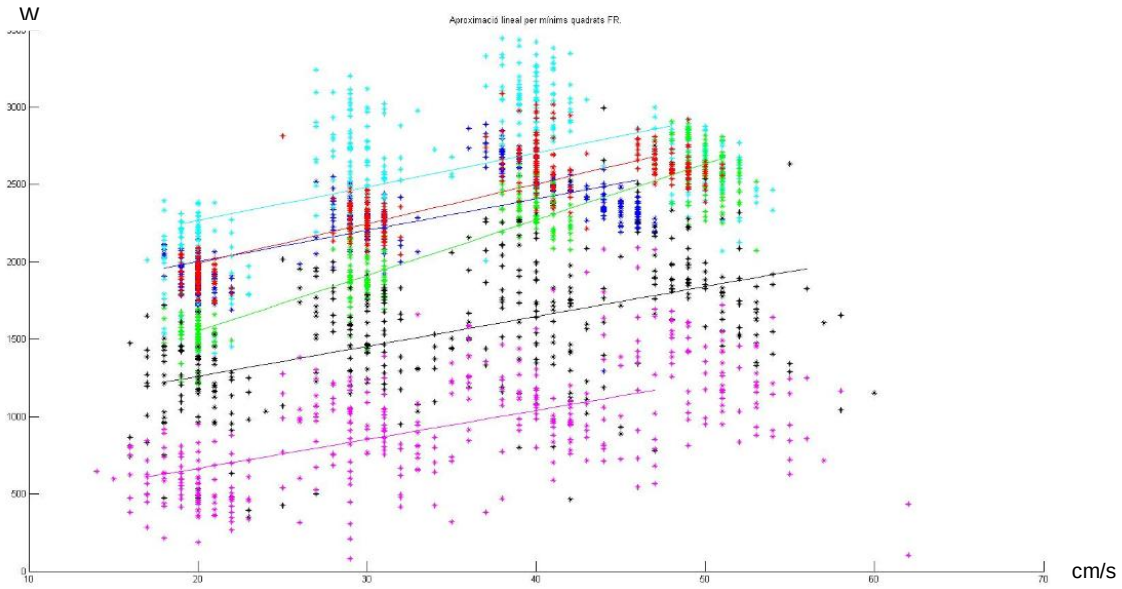

Figura 14. Rectes terrenys amb fre al 20% de la roda dreta

Amb una taula podrem veure quin terreny es cadascun:

Terrenys	Color
Herba	verd
Asfalt	blau fosc
Ciment barracons	vermell
Mosaic PII	blau cel
Pedra gran Pati	morat
Sorra	negre

Taula 13. Colors de les corbes

Com podem veure en la figura anterior hi han quatre terrenys que son molt semblants, aquests són el asfalt, el ciment de barracons, herba i interior laboratori. En principi no hi ha cap problema entre aquests terrenys, perquè tenen un coeficient de terreny molt semblant. Per tant el gir el farà bé igualment.

El motiu de frenar un 20% les rodes del darrera és perquè només amb aquesta frenada s'aconsegueix una separació clara entre els punts dels diferents terres, així els podem diferenciar.

Així doncs aquesta aplicació es el punt de partida per idear la programació en C del pic perquè el robot mòbil en estat online estigui llegint les dades i indiqui en quin terreny s'està movent mitjançant la lectura de la velocitat en funció de la potència. Aquest programa en C que fa aquesta feina s'anomena check_terrain.

5.8 Detecció del terreny

La part que detecta el terreny es un petit programa que ja estava fet anteriorment, aquest utilitza les dades que tenim del MATLAB i les compara amb les que li arriben quan esta en moviment. El programa fa que el robot estigui dos segons avançant en línea recte amb les rodes de darrera frenades un 20% així, l'hi es més fàcil detectar canvis en el terreny, per evitar possibles equivocacions. El codi d'aquest programa el trobarem al annex amb el títol check terrain.

Per poder executar aquest programa haurem de guardar-lo en el ordinador, després hem d'utilitzar la compilació perquè puguem executar el node check_terrain, que és el programa que volem utilitzar, un cop compilat aquest node el podrem executar en un terminal Ubuntu utilitzant la següent comanda:

```
rosrun projectbigbot check_terrain
```

Un cop tenim el node executat podrem visualitzar les comparacions que fa amb la desviació estàndard i la variança dels terrenys que hem guardat anteriorment en el base_controller, i amb el terreny que esta detectant ara.

Quan executem aquest node, es pot activar la prova de detecció de terreny mitjançant el joystick, però després seguirà funcionant automàticament. El que fa es calcula la

desviació estàndard del corrent en la superfície actual, quan aquest programa detecta un canvi en el corrent bastant significant durant un temps comença a fer la detecció de terreny automàticament, un cop feta envia aquestes dades al base_controller i aquest indicarà quin terreny es. En la figura següent es pot veure les comparacions que fa:

```
Your Hardware Enablement Stack (HWE) is supported until April 2017.

Last login: Thu May 12 11:31:35 2016 from 192.168.1.100
mobilerobot@BigBot:~$ rosrun projectbigbot check_terrain
Outliers: 18  Current standard deviation: 2.20792 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 18
Outliers: 16  Current standard deviation: 2.14965 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 16
Outliers: 17  Current standard deviation: 2.06532 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 17
Outliers: 16  Current standard deviation: 1.95154 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 16
Outliers: 16  Current standard deviation: 1.80276 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 16
Outliers: 16  Current standard deviation: 1.60928 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 16
Outliers: 15  Current standard deviation: 1.35207 FL: 339.562 FR: 270.122 BL: 21
0.254 BR: 156.24
Change of terrain detected!!!  Outliers: 15
Outliers: 1  Current standard deviation: 12.7901 FL: 350.482 FR: 293.093 BL: 181
.17 BR: 154.928
Outliers: 0  Current standard deviation: 12.8067 FL: 322.47 FR: 252.101 BL: 210.
816 BR: 148.084
```

Figura 15. Comparacions terrenys

Just quan acaba de fer la comparació en el terminal del base_controller, ens indica amb un missatge en quin terreny ens trobem actualment. El que ens permetrà el programa es veure també la distancia entre el corrent del terreny actual i el que tenim de les dades, com podem veure a les següents figures:

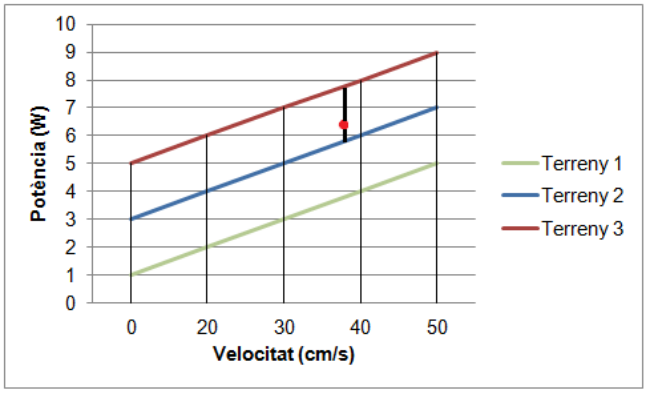

Figura 16. Exemple de distancia entre un punt i les rectes del corrent

Per fer la comparació, el corrent actual queda entre dues rectes, en aquest cas el punt vermell, i escull la recta que te menys distancia en vertical amb el punt, en el cas de la figura anterior el punt vermell esta entre dues rectes, i la que té menys distancia es la del terreny 2, per tant el programa ens indicarà que estem en el terreny 2.

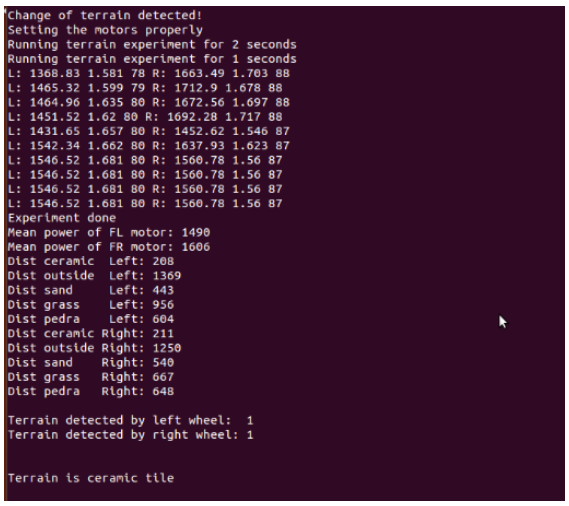

Figura 17. Missatge del terreny

6. RESULTATS FINALS

En el capítol 5 hem explicat les diferents parts que són el gir, el frenat del robot, la detecció del terreny amb els coeficients, la seva placa de control i ara passarem a exposar els resultats finals obtinguts i com quedarà el programa final dintre el robot.

6.1 Arquitectura ROS

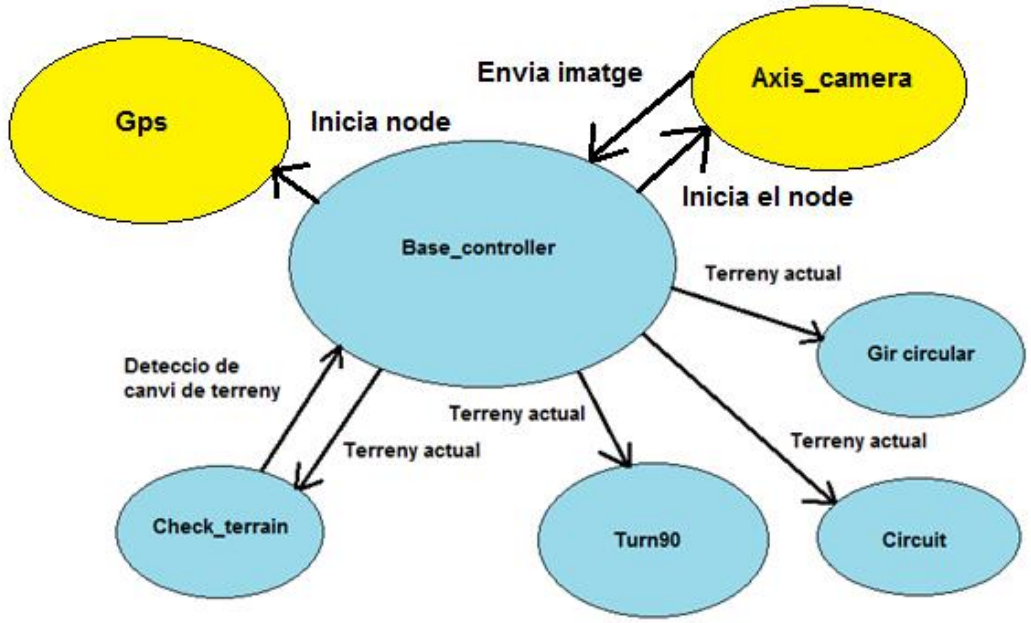

Figura 18. Organigrama del funcionament del ROS

En el organigrama podem veure com es relacionen els nostres programes, per començar el programa principal el base_controller és l'encarregat de dirigir els altres nodes, el node check_terrain envia els canvis de corrent al base_controller i aquest li diu en quin terreny estem actualment. El check_terrain intercanvia informació amb el base_controller només quan hi ha un canvi significant de superfície i ho fa de manera automàtica. Els altres nodes que contenen els circuits només reben la informació del terreny actual per poder saber quin coeficient aplicar en el moment del gir. El base_controller te més nodes relacionats, els que no hem utilitzat són els grocs. El axis_camera, s'inicia quan el base_controller s'activa. El axis_camera envia la imatge de la càmera al base_controller i aquest per el terminal treu la imatge que tenim. El node Gps per funcionar necessita que el base_controller estigui actiu.

6.2 Prova de validació

Per la part final hem decidit fer un circuit gran, l'hem dividit en tres parts per demostrar que pot anar per totes les superfícies on hem agafat dades. Com que els terrenys no estan prou junts per poder fer un circuit molt gran hem decidit fer-lo en tres circuits diferents.

Els tres circuits parteixen de la mateixa base per això no fa falta canviar moltes coses de un programa a un altre.

Com hem dit anteriorment en les proves que hem fet, la part principal del programa del circuit es centra en un switch amb diferents casos (case), cada un d'aquests casos representa un terreny, on a dins conté el valor del coeficient del terreny juntament amb la variable de la detecció de la superfície, també tenim una activació del circuit que s'aplicarà amb el joystick.

Una altra part que hem nombrat en la part dels assajos es la del gir, aquesta part si que canviarà una mica depenen del circuit que vulguem utilitzar o fer servir.

El primer circuit farem que el robot vagi en el asfalt i en el terra dels barracons, ja que en aquesta part tenim una transició bastant bona entre aquestes dues superfícies, a més a més els terrenys no tenen quasi desnivell, per tant son perfectes per al circuit. En aquest el robot sortirà del terra dels barracons farà un gir en la mateixa superfície, després de fer el gir anirà recte on hi haurà la transició entre el primer terreny i el següent, en aquest cas l'asfalt, un cop feta la transició farà un altre gir de 90º acabant així el circuit. Per fer el programa d'aquest hem utilitzat el mateix que als assajos anteriors, en aquest cas gira cap a la dreta en els dos casos.

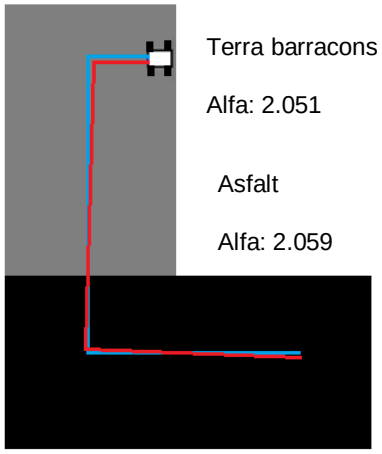

Figura 19. Circuit 1 amb trajectòria

Com podem veure a la figura anterior tenim en blau marcada la trajectòria ideal que volem i en vermell la trajectòria que fa, en els dos casos volem un gir de 90º, però en aquest cas ens dona un gir de 92º que és prou semblant al que busquem. Per tant tenim la següent taula:

Número de gir	Angle ideal (º)	Angle obtingut (º)	Error (%)
1	90,00	92,00	2,22
2	90.00	92,00	2,22

Taula 14. Angles circuit 1

En el segon grup de terrenys en transició trobem el mosaic de davant del PII i l'herba. En aquest cas el robot sortirà de la superfície del mosaic on anirà recte per tot el mosaic i farà un primer gir, seguidament seguirà recte en el mosaic i començarà la primera transició de terrenys un cop hagi fet el canvi amb la detecció corresponent farà el gir a la herba, després segueix en línia recta per tota la herba fins que torna girar els 90º, acte seguit continua recte fent l'última transició acabant en un altre gir al mosaic. El circuit resultant es un quadrat, on tots els girs són cap a l'esquerra.

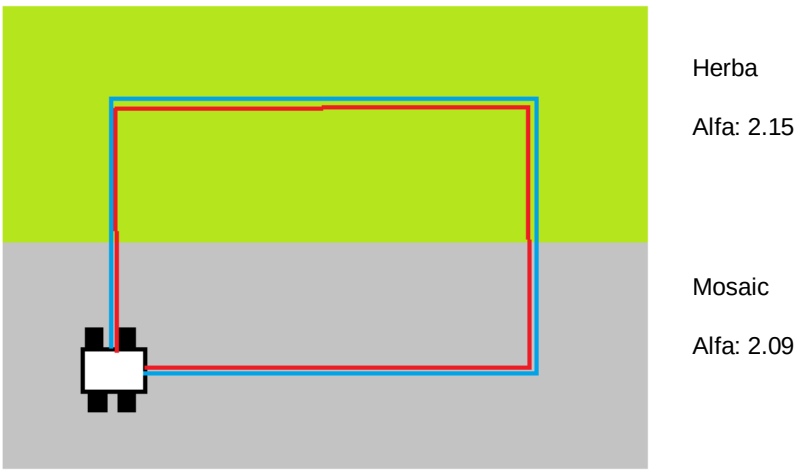

Figura 20. Circuit 2 amb trajectòria

En aquest cas tenim com abans on la trajectòria blava es la ideal on tots els angles son de 90º, per sobre tenim la vermella que es la que fa el robot, els resultats d'aquests angles són més petits de 90º així que tenim un petit error.

Número de gir	Angle ideal (º)	Angle obtingut (º)	Error (%)
1	90,00	88,00	2,22
2	90,00	89,00	1,11
3	90,00	87,00	3,33
4	90,00	89,00	1,11

Taula 15. Angles circuit 2

Per finalitzar fem el circuit que conformen totes les superfícies que podem trobar al laboratori de sistemes intel·ligents, en aquest cas tindrem terra de l'interior del laboratori, també utilitzarem sorra, a continuació el sòl del exterior del laboratori que és grava, i també hi haurà pedra. En aquest cas el circuit tindrà canvis de direcció a l'esquerra i a la dreta. En aquest cas hem hagut de posar la sorra del pati just a la porta per poder fer una transició entre el interior del laboratori i la sorra, també hem de crear una nova transició entre la grava i la pedra. Un cop tenim les transicions col·locades correctament podem executar tranquil·lament el circuit. El circuit començarà en el interior del laboratori on farà un primer gir de 90º, després seguirà recte per fer la transició amb la sorra per fer a continuació el gir en la sorra, seguidament continuarà el circuit anant recte per fer la transició amb la grava o el terra del exterior, a continuació del gir a la grava segueix recte i fa la última transició amb les pedres, on farà un últim gir. Aquest circuit utilitza els mateixos elements que els

anteriors, però combinant els dos sentits de gir. El primer gir es cap a la dreta, el següent a l'esquerra i els dos últims cap a la dreta.

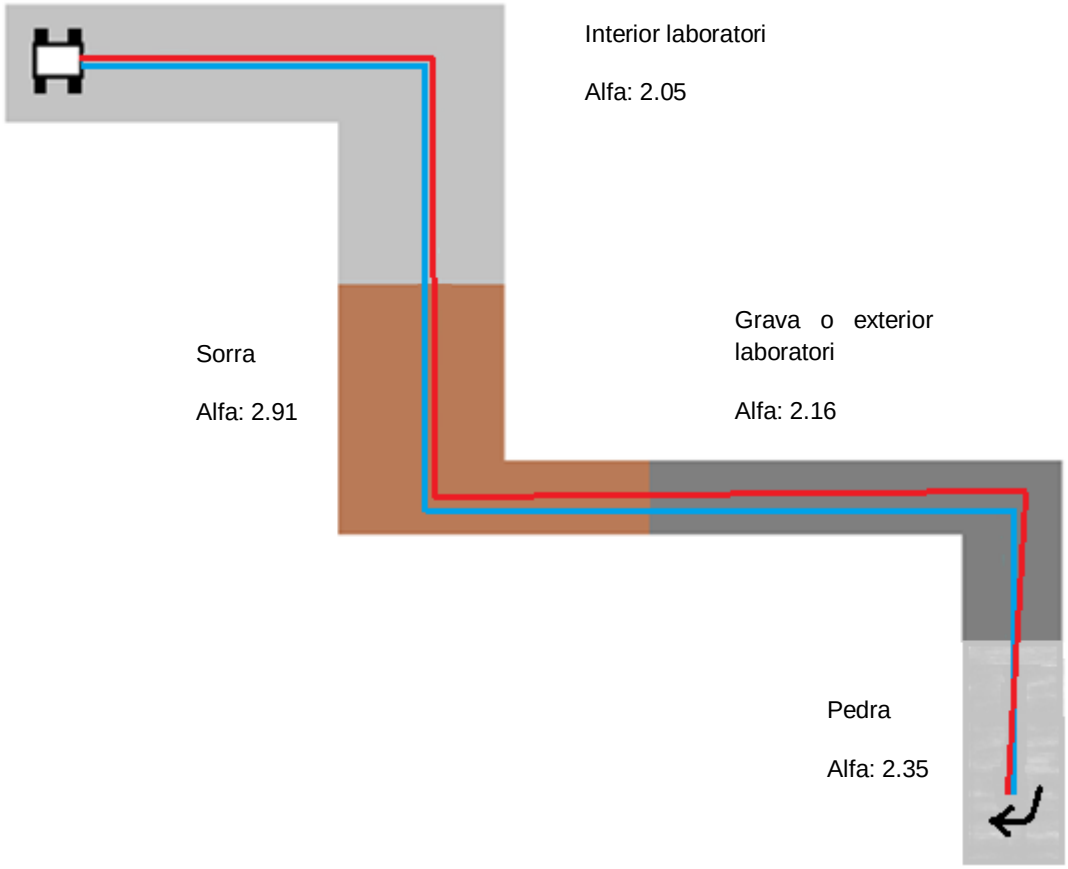

Figura 21. Circuit 3 amb trajectòria

En la figura anterior podem veure la trajectòria ideal juntament amb la trajectòria real, es pot apreciar com en el terreny del pati el robot ha girat menys que en altres terrenys, tot i que a simple vista no es pot veure del tot, amb els números dels angles obtinguts tenim la següent taula per demostrar-ho.

Número de gir	Angle ideal (º)	Angle obtingut (º)	Error (%)
1	90,00	91,00	1,11
2	90,00	88,00	2,22
3	90,00	86,00	4,44
4	90,00	89,00	1,11

Taula 16. Angles circuit 3

Un cop tenim aquests assajos finalitzats amb les corresponents demostracions de funcionament, podem dir que el resultat es correcte, ja que tenim errors molts petits, l'únic terreny on podríem tenir una mica de problemes com hem pogut veure es la superfície de fora el laboratori, perquè es molt rugosa i tendeix a quedar-se el gir una mica més curt.

Per poder fer els circuit exteriors que són els dos primer hem tingut que utilitzar el carro amb cable Ethernet, ja que el Internet del laboratori no té el rang suficient com per arribat a les distancies dels terrenys exteriors que hem utilitzar. També hem tingut que posar un joystick al robot per poder polsar l'activació que necessitàvem en aquell moment, que es la primera detecció de terreny. En canvi per el circuit interior hem pogut fer-ho de manera més automàtica, ja que el rang del Internet era el adequat i el joystick el podíem utilitzar des de el interior, on hem fet també la primera detecció i la detecció del terreny de la sorra també l'hem fet de manera manual ja que el espai que hi ha es reduït, però en els terrenys següents ho fa ell sol.

Hem fet un circuit com el tercer, però sense pedres, com a les figures anteriors tenim en blau la trajectòria desitjada i en vermell la que fa el robot. El angle real que fa el robot l'hem tret dels assajos fets en les diferents superfícies.

Com podem veure a la taula 17 tenim els valors de velocitat de les rodes dretes i de les rodes esquerres, que ens serviran per utilitzar un valor de alfa o un altre. En el cas de la sorra hem utilitzat un gir molt tancat perquè el espai que tenim es molt reduït i necessitem un gir petit.

Els valors de l'alfa els agafem depenent de les velocitats emprades en cada tipus de gir, i també tenint en compte el terreny en el que estem situats.

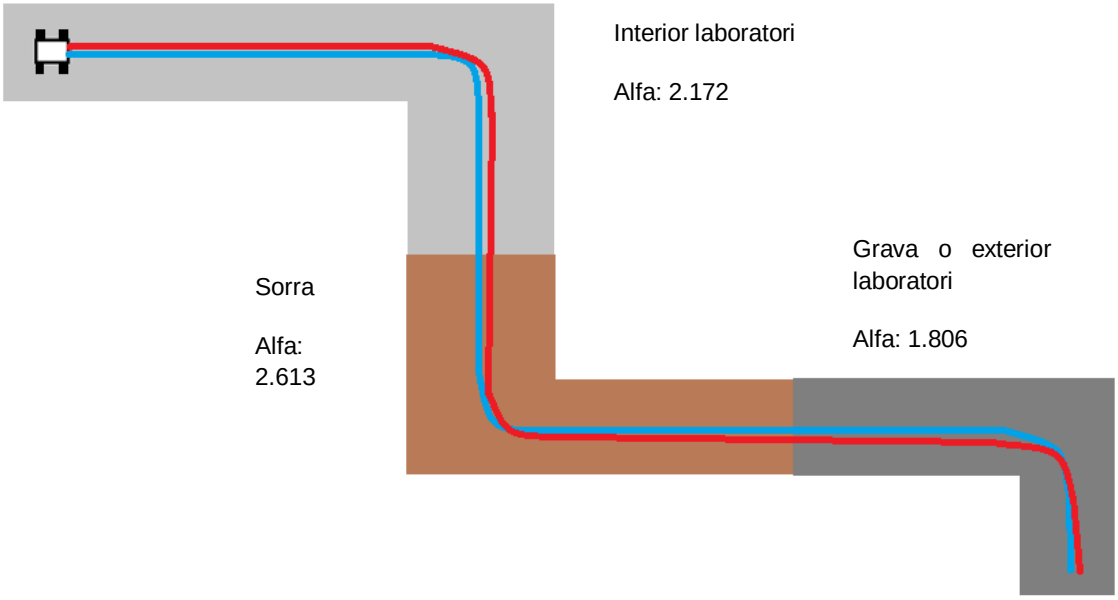

Figura 22. Circuit 3 amb trajectòria del gir circular

Com podem veure el gir que fa en totes les direccions es queda curt, a la taula següent podrem veure els angles esperats i els que obtenim realment, els angles resultants els hem obtingut amb els assajos fets a cada terreny provant aquest tipus de gir, que també hem utilitzat aquestes proves per agafar dades.

Número de gir (Vel. Rodes)	Angle ideal (º)	Angle obtingut (º)	Error (%)
1 (40-20)	90,00	87,00	3,33
2 (15-45)	90,00	86,00	4,44
3 (40-20)	90,00	85,00	5,56

Taula 17. Angles circuit 3 amb gir circular

Podem dir que el resultat es prou bo, ja que l'error màxim que tenim és a l'exterior que es el terreny més difícil de girar, el seu error màxim es de 5,56% que tampoc es molt elevat, si ho mirem a simple vista no hi ha molta diferència dels 90º.

7. RESUM DEL PRESSUPOST

El cost total per la realització dels assajos i de la programació implementada en aquest projecte és de quatre mil cinc-cents seixanta euros sense IVA.

8. CONCLUSIONS

Un cop acabat aquest projecte es pot concloure que hem estat capaços d'assolir els objectius proposats al principi.

S'han fet els estudis pertinents als terrenys escollits. Primerament estudiant les respostes d'aquests amb el robot funcionant en recte i girant, per tal de determinar si calia una nova sintonització o no.

Amb la realització d'aquest projecte, s'han aconseguit ampliar els meus coneixements en llenguatges de programació com Python i C++ i dels sistemes operatiu Ubuntu i ROS. Tot i ser un projecte amb molt de software al treballar amb un robot hem hagut de tocar també part de hardware, on han sorgit diferents problemes.

Un primer problema que vam poder apreciar és que les rodes del robot s'han d'anar comprovant, per evitar que quedin fluixes i no agafar malament les dades. Vam tenir un greu problema amb un encoder, no ens donava senyal i la roda quedava engegada tota l'estona, això va provocar que estiguéssim dues setmanes sense poder fer proves fins que vam solucionar l'encoder. Un problema que vam tenir en la comunicació de la placa exterior amb el router ens va deixar molt de temps sense poder fer gran cosa.

Al determinar que el gir no era òptim en molts dels casos, s'ha optat per estudiar el control d'orientació del robot, amb el qual s'han aconseguit els factors de correcció adients per a la seva millora en l'orientació.

S'ha aconseguit fer un programa pel BIGBOT que es capaç de girar en qualsevol superfície que tenim al campus, pot girar cap a la dreta i cap a l'esquerre, i es capaç de detectar qualsevol terreny i utilitzar el coeficient corresponent en cada cas.

També hem fet un petit incís en el gir circular del robot, perquè en camp obert no anirem rectificant o girant sempre sobre el propi eix, per això hem decidit fer el gir circular de 90º. Cal dir que hem trobat els coeficients per fer un arc de 90º, que son molt semblants als del gir en el propi eix, inclús en alguns terrenys eren més petits. Però encara falta fer més proves en diferents terrenys amb aquest tipus de gir.

Respecte al treball futur, el següent pas a fer seria el possible canvi dels gir sobre si mateix pel gir circular, o en un mateix circuit poder fer ús dels dos tipus de gir depenen del que es necessiti en aquell cas.

Seguint amb el gir circular es podrien fer proves per saber com son les corbes de velocitat segons el terreny i si es necessari o no utilitzar el frenat en aquest tipus de gir.

Aquesta implementació només és la continuació del projecte de rescat, per al futur s'haurà de continuar amb la millora de la navegació del robot, dotant al robot la capacitat d'esquivar obstacles i incorporar nous sensors que millorin la seva tasca.

Un altre aspecte que es podria considerar pel futur és l'actualització tant de la versió de ROS com la versió del Ubuntu, ja que aquestes versions s'estan quedant obsoletes. La robòtica evoluciona molt ràpidament, per tant, el ROS al ser un sistema operatiu robòtic també ho fa, actualment ja existeixen les versions de ROS Hidro Medusa, Indigo Igloo i Jade Turtle. Aquestes actualitzacions ajudarien a trobar paquets que puguin fer una millor comunicació amb altres sistemes com els implementats en aquest projecte.

Alex Martín Rio

Graduat en Enginyeria Electrònica Industrial i Automàtica

Girona, 6 de juny del 2016

9. RELACIÓ DE DOCUMENTS

Aquest projecte consta de quatre documents independents. Aquests són la memòria, el plec de condicions, l'estat d'amidaments i el pressupost.

10. BIBLIOGRAFIA

GITBOOK. Erle Robots Gitbook, a small-size Linux computer for making drones. (https://www.gitbook.com/book/erlerobotics/erlerobot/details, 10 d'abril de 2016)

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

LOCOSYS Mòdul GPS LS20031
(http://www.locosystech.com/product.php?zln=en&id=20, 20 d'abril de 2016)

McGraw-Hill, TCP/IP: Arquitectura, Protocolos e Implementacion, Madrid, Espanya, 2004

OpenNI, OpenNI Org., (http://www.openni.org/, 20 d'abril de 2016)

ROS, Ros, (http://www.ros.org/wiki/, 23 d'abril de 2016)

ROS INSTALLATION, Nootrix, (http://nootrix.com/2012/05/ros-installation/, 23 d'abril de 2016)

ROS WIKI. Informació d'elements de Ros, Ros. (http://www.ros.org/wiki/, 5 de maig de 2016)

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

UdG, Resum de Hardware, Girona, Espanya, 2006

11. GLOSSARI

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 Green 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. Parts del base_controller

```
//Inicialització variables dist

int dist1L;

int dist2L;

int dist3L;

int dist4L;

int dist5L;

int dist6L;

int dist7L;

int dist9L;

int dist1R;

int dist2R;

int dist3R;

int dist4R;

int dist5R;

int dist6R;

int dist7R;

int dist9R;


// Inicialització variables terreny

float a1L;

float b1L;

float c1L;

float a1R;

float b1R;

float c1R;
```

```
// Dades terrenys

//Ceramin tile (lab interior)

a1L = 21.0568;

b1L = -1;

c1L = 439.2821;

a1R = 24.3999;

b1R = -1;

c1R = 418.3002;


// Visualització de la distancia entre terrenys

dist1L = abs(a1L*speed+c1L-L_int_b);

dist1R = abs(a1R*speed+c1R-R_int_b);

cout << "Dist ceramic  Left: " << dist1L << endl;

cout << "Dist ceramic Right: " << dist1R << endl;

terrainL  =  min(dist9L,  min(dist7L,  min(dist6L,  min(dist5L,  min(dist1L,
min(dist2L, min(dist3L, dist4L)))))));

terrainR  =  min(dist9R,  min(dist7R,  min(dist6R,  min(dist5R,  min(dist1R,
min(dist2R, min(dist3R, dist4R)))))));

if(terrainL==dist1L){

      terrainL = 1;

if(terrainR==dist1R){

      terrainR = 1;

cout << endl;

cout << "Terrain detected by left wheel:  " << terrainL << endl;

cout << "Terrain detected by right wheel: " << terrainR << endl;

cout << endl;
```

```
//Missatge tipus de terrenys
if (terrainL == terrainR){
flag_wrong_test = false;
       switch(terrainL)
              case 1:
                     cout << endl << "Terrain is ceramic tile" << endl << endl;
                     terrtype = 1;
                     break;
              case 2:
                     cout << endl << "Terrain is compacted gravel" << endl <<
endl;
                     terrtype = 2;
                     break;
              case 3:
                     cout << endl << "Terrain is sand" << endl << endl;
                     terrtype = 3;
                     break;
              case 4:
                     cout << endl << "Terrain is grass" << endl << endl;
                     terrtype = 4;
                     break;
              case 5:
                     cout << endl << "Terrain is Mosaic P2" << endl << endl;
                     terrtype = 5;
                     break;
              case 6:
                     cout << endl << "Terrain is Asfalt" << endl << endl;
                     terrtype = 6;
```

```
                    break;
            case 7:
                    cout << endl << "Terrain is Asfalt" << endl << endl;
                    terrtype = 7;
                    break;
            case 9:
                    cout << endl << "Terrain is Pedra barracons" << endl <<
endl;
                    terrtype = 9;
                    break;
                    }
            }else{
                    cout << "An error ocurred during the experiment and it
                    should be repeated" << endl;
      flag_wrong_test = true;
```

B. Programa de Matlab

```
close all;

%clear all;

%Dades Asfalt

Imp=importdata('BB3_asfalt_20_16.txt');

Data=Imp.data;

Imp2=importdata('BB3_asfalt_30_24.txt');

Data2=Imp2.data;

Imp3=importdata('BB3_asfalt_40_32.txt');

Data3=Imp3.data;

Imp4=importdata('BB3_asfalt_50_40.txt');

Data4=Imp4.data;


%Dades Mosaic PII

Imp6=importdata('BB3_mosai_20_16.txt');

Data6=Imp6.data;

Imp7=importdata('BB3_mosai_30_24.txt.txt');

Data7=Imp7.data;

Imp8=importdata('BB3_mosai_40_32.txt');

Data8=Imp8.data;

Imp9=importdata('BB3_mosai_50_40.txt');

Data9=Imp9.data;


%Dades Herba

Imp11=importdata('BB3_herba_20_16.txt');

Data11=Imp11.data;

Imp12=importdata('BB3_herba_30_24.txt.txt');

Data12=Imp12.data;

Imp13=importdata('BB3_herba_40_32.txt');

Data13=Imp13.data;

Imp14=importdata('BB3_herba_50_40.txt');
```

```
Data14=Imp14.data;


%Dades Sorra PII
Imp16=importdata('BB3_terra_20_16.txt');
Data16=Imp16.data;
Imp17=importdata('BB3_terra_30_24.txt');
Data17=Imp17.data;
Imp18=importdata('BB3_terra_40_32.txt');
Data18=Imp18.data;
Imp19=importdata('BB3_terra_50_40.txt');
Data19=Imp19.data;


%Dades Ciment barracons
Imp21=importdata('BB3_ciment_barracons_20_16.txt');
Data21=Imp21.data;
Imp22=importdata('BB3_ciment_barracons_30_24.txt');
Data22=Imp22.data;
Imp23=importdata('BB3_ciment_barracons_40_32.txt');
Data23=Imp23.data;
Imp24=importdata('BB3_ciment_barracons_50_40.txt');
Data24=Imp24.data;


%Dades Pedra barracons
Imp26=importdata('BB3_pedra_barracons_20_16.txt');
Data26=Imp26.data;
Imp27=importdata('BB3_pedra_barracons_30_24.txt');
Data27=Imp27.data;
Imp28=importdata('BB3_pedra_barracons_40_32.txt');
Data28=Imp28.data;
Imp29=importdata('BB3_pedra_barracons_50_40.txt');
Data29=Imp29.data;
```

```
%% DADES ASFALT

%% Separate all variables Aspalt data1

speed_FL = Data(101:200,2);

curr_FL = Data(101:200,6);

speed_FR = Data(101:200,3);

curr_FR = Data(101:200,7);

battery = Data(101:200,13);

ctl_FL = Data(101:200,20);

ctl_FR = Data(101:200,21);


%% Separate all variables Aspalt data2

speed_FL2 = Data2(101:200,2);

curr_FL2 = Data2(101:200,6);

speed_FR2 = Data2(101:200,3);

curr_FR2 = Data2(101:200,7);

battery2 = Data2(101:200,13);

ctl_FL2 = Data2(101:200,20);

ctl_FR2 = Data2(101:200,21);


%% Separate all variables Aspalt data3

speed_FL3 = Data3(101:200,2);

curr_FL3 = Data3(101:200,6);

speed_FR3 = Data3(101:200,3);

curr_FR3 = Data3(101:200,7);

battery3 = Data3(101:200,13);

ctl_FL3 = Data3(101:200,20);

ctl_FR3 = Data3(101:200,21);


%% Separate all variables Aspalt data4

speed_FL4 = Data4(101:200,2);

curr_FL4 = Data4(101:200,6);

speed_FR4 = Data4(101:200,3);
```

```
curr_FR4 = Data4(101:200,7);

battery4 = Data4(101:200,13);

ctl_FL4 = Data4(101:200,20);

ctl_FR4 = Data4(101:200,21);


%% DADES MOSAIC


%% Separate all variables Mosaic data6

speed_FL6 = Data6(101:200,2);

curr_FL6 = Data6(101:200,6);

speed_FR6 = Data6(101:200,3);

curr_FR6 = Data6(101:200,7);

battery6 = Data6(101:200,13);

ctl_FL6 = Data6(101:200,20);

ctl_FR6 = Data6(101:200,21);


%% Separate all variables Mosaic data7

speed_FL7 = Data7(101:200,2);

curr_FL7 = Data7(101:200,6);

speed_FR7 = Data7(101:200,3);

curr_FR7 = Data7(101:200,7);

battery7 = Data7(101:200,13);

ctl_FL7 = Data7(101:200,20);

ctl_FR7 = Data7(101:200,21);


%% Separate all variables Mosaic data8

speed_FL8 = Data8(101:200,2);

curr_FL8 = Data8(101:200,6);

speed_FR8 = Data8(101:200,3);

curr_FR8 = Data8(101:200,7);

battery8 = Data8(101:200,13);

ctl_FL8 = Data8(101:200,20);
```

```
ctl_FR8 = Data8(101:200,21);


%% Separate all variables Mosaic data9

speed_FL9 = Data9(101:200,2);

curr_FL9 = Data9(101:200,6);

speed_FR9 = Data9(101:200,3);

curr_FR9 = Data9(101:200,7);

battery9 = Data9(101:200,13);

ctl_FL9 = Data9(101:200,20);

ctl_FR9 = Data9(101:200,21);


%% DADES HERBA


%% Separate all variables Herba data 11

speed_FL11 = Data11(101:200,2);

curr_FL11 = Data11(101:200,6);

speed_FR11 = Data11(101:200,3);

curr_FR11 = Data11(101:200,7);

battery11 = Data11(101:200,13);

ctl_FL11 = Data11(101:200,20);

ctl_FR11 = Data11(101:200,21);


%% Separate all variables Herba data 12

speed_FL12 = Data12(101:200,2);

curr_FL12 = Data12(101:200,6);

speed_FR12 = Data12(101:200,3);

curr_FR12 = Data12(101:200,7);

battery12 = Data12(101:200,13);

ctl_FL12 = Data12(101:200,20);

ctl_FR12 = Data12(101:200,21);


%% Separate all variables Herba data 13
```

```
speed_FL13 = Data13(101:200,2);

curr_FL13 = Data13(101:200,6);

speed_FR13 = Data13(101:200,3);

curr_FR13 = Data13(101:200,7);

battery13 = Data13(101:200,13);

ctl_FL13 = Data13(101:200,20);

ctl_FR13 = Data13(101:200,21);


%% Separate all variables Herba data 14

speed_FL14 = Data14(101:200,2);

curr_FL14 = Data14(101:200,6);

speed_FR14 = Data14(101:200,3);

curr_FR14 = Data14(101:200,7);

battery14 = Data14(101:200,13);

ctl_FL14 = Data14(101:200,20);

ctl_FR14 = Data14(101:200,21);


%% DADES SORRA PII


%% Separate all variables Sorra data16

speed_FL16 = Data16(101:200,2);

curr_FL16 = Data16(101:200,6);

speed_FR16 = Data16(101:200,3);

curr_FR16 = Data16(101:200,7);

battery16 = Data16(101:200,13);

ctl_FL16 = Data16(101:200,20);

ctl_FR16 = Data16(101:200,21);


%% Separate all variables Sorra data17

speed_FL17 = Data17(101:200,2);

curr_FL17 = Data17(101:200,6);

speed_FR17 = Data17(101:200,3);
```

```
curr_FR17 = Data17(101:200,7);

battery17 = Data17(101:200,13);

ctl_FL17 = Data17(101:200,20);

ctl_FR17 = Data17(101:200,21);


%% Separate all variables Sorra data18

speed_FL18 = Data18(101:200,2);

curr_FL18 = Data18(101:200,6);

speed_FR18 = Data18(101:200,3);

curr_FR18 = Data18(101:200,7);

battery18 = Data18(101:200,13);

ctl_FL18 = Data18(101:200,20);

ctl_FR18 = Data18(101:200,21);


%% Separate all variables Sorra data19

speed_FL19 = Data19(101:200,2);

curr_FL19 = Data19(101:200,6);

speed_FR19 = Data19(101:200,3);

curr_FR19 = Data19(101:200,7);

battery19 = Data19(101:200,13);

ctl_FL19 = Data19(101:200,20);

ctl_FR19 = Data19(101:200,21);


%% DADES CIMENT BARRACONS


%% Separate all variables ciment barraconsdata 21

speed_FL21 = Data21(101:200,2);

curr_FL21 = Data21(101:200,6);

speed_FR21 = Data21(101:200,3);

curr_FR21 = Data21(101:200,7);

battery21 = Data21(101:200,13);

ctl_FL21 = Data21(101:200,20);
```

```
ctl_FR21 = Data21(101:200,21);


%% Separate all variables ciment barraconsdata 22

speed_FL22 = Data22(101:200,2);

curr_FL22 = Data22(101:200,6);

speed_FR22 = Data22(101:200,3);

curr_FR22 = Data22(101:200,7);

battery22 = Data22(101:200,13);

ctl_FL22 = Data22(101:200,20);

ctl_FR22 = Data22(101:200,21);


%% Separate all variables ciment barraconsdata 23

speed_FL23 = Data23(101:200,2);

curr_FL23 = Data23(101:200,6);

speed_FR23 = Data23(101:200,3);

curr_FR23 = Data23(101:200,7);

battery23 = Data23(101:200,13);

ctl_FL23 = Data23(101:200,20);

ctl_FR23 = Data23(101:200,21);


%% Separate all variables ciment barraconsdata 24

speed_FL24 = Data24(101:200,2);

curr_FL24 = Data24(101:200,6);

speed_FR24 = Data24(101:200,3);

curr_FR24 = Data24(101:200,7);

battery24 = Data24(101:200,13);

ctl_FL24 = Data24(101:200,20);

ctl_FR24 = Data24(101:200,21);


%% DADES PEDRA BARRACONS


%% Separate all variables pedra barracons data26
```

```
speed_FL26 = Data26(101:200,2);

curr_FL26 = Data26(101:200,6);

speed_FR26 = Data26(101:200,3);

curr_FR26 = Data26(101:200,7);

battery26 = Data26(101:200,13);

ctl_FL26 = Data26(101:200,20);

ctl_FR26 = Data26(101:200,21);


%% Separate all variables pedra barracons data27

speed_FL27 = Data27(101:200,2);

curr_FL27 = Data27(101:200,6);

speed_FR27 = Data27(101:200,3);

curr_FR27 = Data27(101:200,7);

battery27 = Data27(101:200,13);

ctl_FL27 = Data27(101:200,20);

ctl_FR27 = Data27(101:200,21);


%% Separate all variables pedra barracons data28

speed_FL28 = Data28(101:200,2);

curr_FL28 = Data28(101:200,6);

speed_FR28 = Data28(101:200,3);

curr_FR28 = Data28(101:200,7);

battery28 = Data28(101:200,13);

ctl_FL28 = Data28(101:200,20);

ctl_FR28 = Data28(101:200,21);


%% Separate all variables pedra barracons data29

speed_FL29 = Data29(101:200,2);

curr_FL29 = Data29(101:200,6);

speed_FR29 = Data29(101:200,3);

curr_FR29 = Data29(101:200,7);

battery29 = Data29(101:200,13);
```

```
ctl_FL29 = Data29(101:200,20);

ctl_FR29 = Data29(101:200,21);


for z=1:100

 %%Vectors de velocitats FL

 AllspeedsFL(z)=speed_FL(z);

 AllspeedsFL(z+100)=speed_FL2(z);

 AllspeedsFL(z+200)=speed_FL3(z);

 AllspeedsFL(z+300)=speed_FL4(z);


 Allspeeds2FL(z)=speed_FL6(z);

 Allspeeds2FL(z+100)=speed_FL7(z);

 Allspeeds2FL(z+200)=speed_FL8(z);

 Allspeeds2FL(z+300)=speed_FL9(z);


 Allspeeds3FL(z)=speed_FL11(z);

 Allspeeds3FL(z+100)=speed_FL12(z);

 Allspeeds3FL(z+200)=speed_FL13(z);

 Allspeeds3FL(z+300)=speed_FL14(z);


 Allspeeds4FL(z)=speed_FL16(z);

 Allspeeds4FL(z+100)=speed_FL17(z);

 Allspeeds4FL(z+200)=speed_FL18(z);

 Allspeeds4FL(z+300)=speed_FL19(z);


 Allspeeds5FL(z)=speed_FL21(z);

 Allspeeds5FL(z+100)=speed_FL22(z);

 Allspeeds5FL(z+200)=speed_FL23(z);

 Allspeeds5FL(z+300)=speed_FL24(z);


 Allspeeds6FL(z)=speed_FL26(z);

 Allspeeds6FL(z+100)=speed_FL27(z);
```

```
Allspeeds6FL(z+200)=speed_FL28(z);

Allspeeds6FL(z+300)=speed_FL29(z);


%%Vectors de velocitats FR

AllspeedsFR(z)=speed_FR(z);

AllspeedsFR(z+100)=speed_FR2(z);

AllspeedsFR(z+200)=speed_FR3(z);

AllspeedsFR(z+300)=speed_FR4(z);


Allspeeds2FR(z)=speed_FR6(z);

Allspeeds2FR(z+100)=speed_FR7(z);

Allspeeds2FR(z+200)=speed_FR8(z);

Allspeeds2FR(z+300)=speed_FR9(z);


Allspeeds3FR(z)=speed_FR11(z);

Allspeeds3FR(z+100)=speed_FR12(z);

Allspeeds3FR(z+200)=speed_FR13(z);

Allspeeds3FR(z+300)=speed_FR14(z);


Allspeeds4FR(z)=speed_FR16(z);

Allspeeds4FR(z+100)=speed_FR17(z);

Allspeeds4FR(z+200)=speed_FR18(z);

Allspeeds4FR(z+300)=speed_FR19(z);


Allspeeds5FR(z)=speed_FR21(z);

Allspeeds5FR(z+100)=speed_FR22(z);

Allspeeds5FR(z+200)=speed_FR23(z);

Allspeeds5FR(z+300)=speed_FR24(z);


Allspeeds6FR(z)=speed_FR26(z);

Allspeeds6FR(z+100)=speed_FR27(z);

Allspeeds6FR(z+200)=speed_FR28(z);
```

```
Allspeeds6FR(z+300)=speed_FR29(z);


%%Vectors de bateria
Allbattery(z)=battery(z);
Allbattery(z+100)=battery2(z);
Allbattery(z+200)=battery3(z);
Allbattery(z+300)=battery4(z);


Allbattery2(z)=battery6(z);
Allbattery2(z+100)=battery7(z);
Allbattery2(z+200)=battery8(z);
Allbattery2(z+300)=battery9(z);


Allbattery3(z)=battery11(z);
Allbattery3(z+100)=battery12(z);
Allbattery3(z+200)=battery13(z);
Allbattery3(z+300)=battery14(z);


Allbattery4(z)=battery16(z);
Allbattery4(z+100)=battery17(z);
Allbattery4(z+200)=battery18(z);
Allbattery4(z+300)=battery19(z);


Allbattery5(z)=battery21(z);
Allbattery5(z+100)=battery22(z);
Allbattery5(z+200)=battery23(z);
Allbattery5(z+300)=battery24(z);


Allbattery6(z)=battery26(z);
Allbattery6(z+100)=battery27(z);
Allbattery6(z+200)=battery28(z);
Allbattery6(z+300)=battery29(z);
```

```
%%Vectors de senyal de control FL

AllctlFL(z)=ctl_FL(z);

AllctlFL(z+100)=ctl_FL2(z);

AllctlFL(z+200)=ctl_FL3(z);

AllctlFL(z+300)=ctl_FL4(z);


Allctl2FL(z)=ctl_FL6(z);

Allctl2FL(z+100)=ctl_FL7(z);

Allctl2FL(z+200)=ctl_FL8(z);

Allctl2FL(z+300)=ctl_FL9(z);


Allctl3FL(z)=ctl_FL11(z);

Allctl3FL(z+100)=ctl_FL12(z);

Allctl3FL(z+200)=ctl_FL13(z);

Allctl3FL(z+300)=ctl_FL14(z);


Allctl4FL(z)=ctl_FL16(z);

Allctl4FL(z+100)=ctl_FL17(z);

Allctl4FL(z+200)=ctl_FL18(z);

Allctl4FL(z+300)=ctl_FL19(z);


Allctl5FL(z)=ctl_FL21(z);

Allctl5FL(z+100)=ctl_FL22(z);

Allctl5FL(z+200)=ctl_FL23(z);

Allctl5FL(z+300)=ctl_FL24(z);


Allctl6FL(z)=ctl_FL26(z);

Allctl6FL(z+100)=ctl_FL27(z);

Allctl6FL(z+200)=ctl_FL28(z);

Allctl6FL(z+300)=ctl_FL29(z);


%%Vectors de senyal de control FR
```

```
AllctlFR(z)=ctl_FR(z);

AllctlFR(z+100)=ctl_FR2(z);

AllctlFR(z+200)=ctl_FR3(z);

AllctlFR(z+300)=ctl_FR4(z);


Allctl2FR(z)=ctl_FR6(z);

Allctl2FR(z+100)=ctl_FR7(z);

Allctl2FR(z+200)=ctl_FR8(z);

Allctl2FR(z+300)=ctl_FR9(z);


Allctl3FR(z)=ctl_FR11(z);

Allctl3FR(z+100)=ctl_FR12(z);

Allctl3FR(z+200)=ctl_FR13(z);

Allctl3FR(z+300)=ctl_FR14(z);


Allctl4FR(z)=ctl_FR16(z);

Allctl4FR(z+100)=ctl_FR17(z);

Allctl4FR(z+200)=ctl_FR18(z);

Allctl4FR(z+300)=ctl_FR19(z);


Allctl5FR(z)=ctl_FR21(z);

Allctl5FR(z+100)=ctl_FR22(z);

Allctl5FR(z+200)=ctl_FR23(z);

Allctl5FR(z+300)=ctl_FR24(z);


Allctl6FR(z)=ctl_FR26(z);

Allctl6FR(z+100)=ctl_FR27(z);

Allctl6FR(z+200)=ctl_FR28(z);

Allctl6FR(z+300)=ctl_FR29(z);


%%Vector de corrents FL

AllcurrentsFL(z)=curr_FL(z);
```

```
AllcurrentsFL(z+100)=curr_FL2(z);

AllcurrentsFL(z+200)=curr_FL3(z);

AllcurrentsFL(z+300)=curr_FL4(z);


Allcurrents2FL(z)=curr_FL6(z);

Allcurrents2FL(z+100)=curr_FL7(z);

Allcurrents2FL(z+200)=curr_FL8(z);

Allcurrents2FL(z+300)=curr_FL9(z);


Allcurrents3FL(z)=curr_FL11(z);

Allcurrents3FL(z+100)=curr_FL12(z);

Allcurrents3FL(z+200)=curr_FL13(z);

Allcurrents3FL(z+300)=curr_FL14(z);


Allcurrents4FL(z)=curr_FL16(z);

Allcurrents4FL(z+100)=curr_FL17(z);

Allcurrents4FL(z+200)=curr_FL18(z);

Allcurrents4FL(z+300)=curr_FL19(z);


Allcurrents5FL(z)=curr_FL21(z);

Allcurrents5FL(z+100)=curr_FL22(z);

Allcurrents5FL(z+200)=curr_FL23(z);

Allcurrents5FL(z+300)=curr_FL24(z);


Allcurrents6FL(z)=curr_FL26(z);

Allcurrents6FL(z+100)=curr_FL27(z);

Allcurrents6FL(z+200)=curr_FL28(z);

Allcurrents6FL(z+300)=curr_FL29(z);


%%Vector de corrents FR

AllcurrentsFR(z)=curr_FR(z);

AllcurrentsFR(z+100)=curr_FR2(z);
```

```
AllcurrentsFR(z+200)=curr_FR3(z);

AllcurrentsFR(z+300)=curr_FR4(z);


Allcurrents2FR(z)=curr_FR6(z);

Allcurrents2FR(z+100)=curr_FR7(z);

Allcurrents2FR(z+200)=curr_FR8(z);

Allcurrents2FR(z+300)=curr_FR9(z);


Allcurrents3FR(z)=curr_FR11(z);

Allcurrents3FR(z+100)=curr_FR12(z);

Allcurrents3FR(z+200)=curr_FR13(z);

Allcurrents3FR(z+300)=curr_FR14(z);


Allcurrents4FR(z)=curr_FR16(z);

Allcurrents4FR(z+100)=curr_FR17(z);

Allcurrents4FR(z+200)=curr_FR18(z);

Allcurrents4FR(z+300)=curr_FR19(z);


Allcurrents5FR(z)=curr_FR21(z);

Allcurrents5FR(z+100)=curr_FR22(z);

Allcurrents5FR(z+200)=curr_FR23(z);

Allcurrents5FR(z+300)=curr_FR24(z);


Allcurrents6FR(z)=curr_FR26(z);

Allcurrents6FR(z+100)=curr_FR27(z);

Allcurrents6FR(z+200)=curr_FR28(z);

Allcurrents6FR(z+300)=curr_FR29(z);


%%Vector de POTFL

AllPotFL(z)=AllctlFL(z)*Allbattery(z)*AllcurrentsFL(z);

AllPotFL(z+100)=AllctlFL(z+100)*Allbattery(z+100)*AllcurrentsFL(z+100);

AllPotFL(z+200)=AllctlFL(z+200)*Allbattery(z+200)*AllcurrentsFL(z+200);
```

```
AllPotFL(z+300)=AllctlFL(z+300)*Allbattery(z+300)*AllcurrentsFL(z+300);


AllPot2FL(z)=Allctl2FL(z)*Allbattery2(z)*Allcurrents2FL(z);
AllPot2FL(z+100)=Allctl2FL(z+100)*Allbattery2(z+100)*Allcurrents2FL(z+100);
AllPot2FL(z+200)=Allctl2FL(z+200)*Allbattery2(z+200)*Allcurrents2FL(z+200);
AllPot2FL(z+300)=Allctl2FL(z+300)*Allbattery2(z+300)*Allcurrents2FL(z+300);


AllPot3FL(z)=Allctl3FL(z)*Allbattery3(z)*Allcurrents3FL(z);
AllPot3FL(z+100)=Allctl3FL(z+100)*Allbattery3(z+100)*Allcurrents3FL(z+100);
AllPot3FL(z+200)=Allctl3FL(z+200)*Allbattery3(z+200)*Allcurrents3FL(z+200);
AllPot3FL(z+300)=Allctl3FL(z+300)*Allbattery3(z+300)*Allcurrents3FL(z+300);


AllPot4FL(z)=Allctl4FL(z)*Allbattery4(z)*Allcurrents4FL(z);
AllPot4FL(z+100)=Allctl4FL(z+100)*Allbattery4(z+100)*Allcurrents4FL(z+100);
AllPot4FL(z+200)=Allctl4FL(z+200)*Allbattery4(z+200)*Allcurrents4FL(z+200);
AllPot4FL(z+300)=Allctl4FL(z+300)*Allbattery4(z+300)*Allcurrents4FL(z+300);


AllPot5FL(z)=Allctl5FL(z)*Allbattery5(z)*Allcurrents5FL(z);
AllPot5FL(z+100)=Allctl5FL(z+100)*Allbattery5(z+100)*Allcurrents5FL(z+100);
AllPot5FL(z+200)=Allctl5FL(z+200)*Allbattery5(z+200)*Allcurrents5FL(z+200);
AllPot5FL(z+300)=Allctl5FL(z+300)*Allbattery5(z+300)*Allcurrents5FL(z+300);


AllPot6FL(z)=Allctl6FL(z)*Allbattery6(z)*Allcurrents6FL(z);
AllPot6FL(z+100)=Allctl6FL(z+100)*Allbattery6(z+100)*Allcurrents6FL(z+100);
AllPot6FL(z+200)=Allctl6FL(z+200)*Allbattery6(z+200)*Allcurrents6FL(z+200);
AllPot6FL(z+300)=Allctl6FL(z+300)*Allbattery6(z+300)*Allcurrents6FL(z+300);


%%Vector de POTFR
AllPotFR(z)=AllctlFR(z)*Allbattery(z)*AllcurrentsFR(z);
AllPotFR(z+100)=AllctlFR(z+100)*Allbattery(z+100)*AllcurrentsFR(z+100);
AllPotFR(z+200)=AllctlFR(z+200)*Allbattery(z+200)*AllcurrentsFR(z+200);
AllPotFR(z+300)=AllctlFR(z+300)*Allbattery(z+300)*AllcurrentsFR(z+300);
```

```
 AllPot2FR(z)=Allctl2FR(z)*Allbattery2(z)*Allcurrents2FR(z);
 AllPot2FR(z+100)=Allctl2FR(z+100)*Allbattery2(z+100)*Allcurrents2FR(z+100);
 AllPot2FR(z+200)=Allctl2FR(z+200)*Allbattery2(z+200)*Allcurrents2FR(z+200);
 AllPot2FR(z+300)=Allctl2FR(z+300)*Allbattery2(z+300)*Allcurrents2FR(z+300);

 AllPot3FR(z)=Allctl3FR(z)*Allbattery3(z)*Allcurrents3FR(z);
 AllPot3FR(z+100)=Allctl3FR(z+100)*Allbattery3(z+100)*Allcurrents3FR(z+100);
 AllPot3FR(z+200)=Allctl3FR(z+200)*Allbattery3(z+200)*Allcurrents3FR(z+200);
 AllPot3FR(z+300)=Allctl3FR(z+300)*Allbattery3(z+300)*Allcurrents3FR(z+300);

 AllPot4FR(z)=Allctl4FR(z)*Allbattery4(z)*Allcurrents4FR(z);
 AllPot4FR(z+100)=Allctl4FR(z+100)*Allbattery4(z+100)*Allcurrents4FR(z+100);
 AllPot4FR(z+200)=Allctl4FR(z+200)*Allbattery4(z+200)*Allcurrents4FR(z+200);
 AllPot4FR(z+300)=Allctl4FR(z+300)*Allbattery4(z+300)*Allcurrents4FR(z+300);

 AllPot5FR(z)=Allctl5FR(z)*Allbattery5(z)*Allcurrents5FR(z);
 AllPot5FR(z+100)=Allctl5FR(z+100)*Allbattery5(z+100)*Allcurrents5FR(z+100);
 AllPot5FR(z+200)=Allctl5FR(z+200)*Allbattery5(z+200)*Allcurrents5FR(z+200);
 AllPot5FR(z+300)=Allctl5FR(z+300)*Allbattery5(z+300)*Allcurrents5FR(z+300);

 AllPot6FR(z)=Allctl6FR(z)*Allbattery6(z)*Allcurrents6FR(z);
 AllPot6FR(z+100)=Allctl6FR(z+100)*Allbattery6(z+100)*Allcurrents6FR(z+100);
 AllPot6FR(z+200)=Allctl6FR(z+200)*Allbattery6(z+200)*Allcurrents6FR(z+200);
 AllPot6FR(z+300)=Allctl6FR(z+300)*Allbattery6(z+300)*Allcurrents6FR(z+300);


end

%% Plot Aprox. Lineal (mínims quadrats) de POT en funció de velocitat de totes
les dades

figure ('Name','Aproximació lineal PWMFL','NumberTitle','off','Color', [0.8
0.8 0.8])

vectoraproxlinealFLA=[AllspeedsFL;AllPotFL];

vectoraproxlinealFLM=[Allspeeds2FL;AllPot2FL];
```

```
vectoraproxlinealFLH=[Allspeeds3FL;AllPot3FL];

vectoraproxlinealFLS=[Allspeeds4FL;AllPot4FL];

vectoraproxlinealFLC=[Allspeeds5FL;AllPot5FL];

vectoraproxlinealFLP=[Allspeeds6FL;AllPot6FL];


global mFLA

mFLA

global bFLA

bFLA

global mFLM

mFLM

global bFLM

bFLM

global mFLH

mFLH

global bFLH

bFLH

global mFLS

mFLS

global bFLS

bFLS

global mFLC

mFLC

global bFLC

bFLC

global mFLP

mFLP

global bFLP

bFLP


save [mFLA,bFLA]=mincuadlin(vectoraproxlinealFLA);

save [mFLM,bFLM]=mincuadlin(vectoraproxlinealFLM);
```

```
save [mFLH,bFLH]=mincuadlin(vectoraproxlinealFLH);

save [mFLS,bFLS]=mincuadlin(vectoraproxlinealFLP);

save [mFLC,bFLC]=mincuadlin(vectoraproxlinealFLC);

save [mFLP,bFLP]=mincuadlin(vectoraproxlinealFLP);


nFLA=length(vectoraproxlinealFLA(1,:));

nFLM=length(vectoraproxlinealFLM(1,:));

nFLH=length(vectoraproxlinealFLH(1,:));

nFLS=length(vectoraproxlinealFLS(1,:));

nFLC=length(vectoraproxlinealFLC(1,:));

nFLP=length(vectoraproxlinealFLP(1,:));


AFLA=0;

AFLM=0;

AFLH=0;

AFLS=0;

AFLC=0;

AFLP=0;

BFLA=0;

BFLM=0;

BFLH=0;

BFLS=0;

BFLC=0;

BFLP=0;

CFLA=0;

CFLM=0;

CFLH=0;

CFLS=0;

CFLC=0;

CFLP=0;

DFLA=0;

DFLM=0;
```

```
DFLH=0;

DFLS=0;

DFLC=0;

DFLP=0;


for iii=1:nFLA;

 AFLA=AFLA+vectoraproxlinealFLA(1,iii);

 BFLA=BFLA+vectoraproxlinealFLA(2,iii);

 CFLA=CFLA+(vectoraproxlinealFLA(1,iii))^2;

 DFLA=DFLA+vectoraproxlinealFLA(1,iii)*vectoraproxlinealFLA(2,iii);

end


for iii=1:nFLM;

 AFLM=AFLM+vectoraproxlinealFLM(1,iii);

 BFLM=BFLM+vectoraproxlinealFLM(2,iii);

 CFLM=CFLM+(vectoraproxlinealFLM(1,iii))^2;

 DFLM=DFLM+vectoraproxlinealFLM(1,iii)*vectoraproxlinealFLM(2,iii);

end


for iii=1:nFLH;

 AFLH=AFLH+vectoraproxlinealFLH(1,iii);

 BFLH=BFLH+vectoraproxlinealFLH(2,iii);

 CFLH=CFLH+(vectoraproxlinealFLH(1,iii))^2;

 DFLH=DFLH+vectoraproxlinealFLH(1,iii)*vectoraproxlinealFLH(2,iii);

end


for iii=1:nFLS;

 AFLS=AFLS+vectoraproxlinealFLS(1,iii);

 BFLS=BFLS+vectoraproxlinealFLS(2,iii);

 CFLS=CFLS+(vectoraproxlinealFLS(1,iii))^2;

 DFLS=DFLS+vectoraproxlinealFLS(1,iii)*vectoraproxlinealFLS(2,iii);

end
```

```
for iii=1:nFLC;
 AFLC=AFLC+vectoraproxlinealFLC(1,iii);
 BFLC=BFLC+vectoraproxlinealFLC(2,iii);
 CFLC=CFLC+(vectoraproxlinealFLC(1,iii))^2;
 DFLC=DFLC+vectoraproxlinealFLC(1,iii)*vectoraproxlinealFLC(2,iii);
end

for iii=1:nFLP;
 AFLP=AFLP+vectoraproxlinealFLP(1,iii);
 BFLP=BFLP+vectoraproxlinealFLP(2,iii);
 CFLP=CFLP+(vectoraproxlinealFLP(1,iii))^2;
 DFLP=DFLP+vectoraproxlinealFLP(1,iii)*vectoraproxlinealFLP(2,iii);
end

mFLA=(nFLA*DFLA-AFLA*BFLA)/(nFLA*CFLA-AFLA^2);
bFLA=(CFLA*BFLA-DFLA*AFLA)/(nFLA*CFLA-AFLA^2);
mFLM=(nFLM*DFLM-AFLM*BFLM)/(nFLM*CFLM-AFLM^2);
bFLM=(CFLM*BFLM-DFLM*AFLM)/(nFLM*CFLM-AFLM^2);
mFLH=(nFLH*DFLH-AFLH*BFLH)/(nFLH*CFLH-AFLH^2);
bFLH=(CFLH*BFLH-DFLH*AFLH)/(nFLH*CFLH-AFLH^2);
mFLS=(nFLS*DFLS-AFLS*BFLS)/(nFLS*CFLS-AFLS^2);
bFLS=(CFLS*BFLS-DFLS*AFLS)/(nFLS*CFLS-AFLS^2);
mFLC=(nFLC*DFLC-AFLC*BFLC)/(nFLC*CFLC-AFLC^2);
bFLC=(CFLC*BFLC-DFLC*AFLC)/(nFLC*CFLC-AFLC^2);
mFLP=(nFLP*DFLP-AFLP*BFLP)/(nFLP*CFLP-AFLP^2);
bFLP=(CFLP*BFLP-DFLP*AFLP)/(nFLP*CFLP-AFLP^2);

for iii=1:nFLA;
 hold on;

plot(vectoraproxlinealFLA(1,iii),vectoraproxlinealFLA(2,iii),'*','MarkerEdgeCo
lor','b','LineWidth',1);
```

```
end

for iii=1:nFLM;

 hold on;


plot(vectoraproxlinealFLM(1,iii),vectoraproxlinealFLM(2,iii),'*','MarkerEdgeCo
lor','c','LineWidth',1);

end

for iii=1:nFLH;

 hold on;


plot(vectoraproxlinealFLH(1,iii),vectoraproxlinealFLH(2,iii),'*','MarkerEdgeCo
lor','g','LineWidth',1);

end

for iii=1:nFLS;

 hold on;


plot(vectoraproxlinealFLS(1,iii),vectoraproxlinealFLS(2,iii),'*','MarkerEdgeCo
lor','k','LineWidth',1);

end

for iii=1:nFLC;

 hold on;


plot(vectoraproxlinealFLC(1,iii),vectoraproxlinealFLC(2,iii),'*','MarkerEdgeCo
lor','r','LineWidth',1);

end

for iii=1:nFLP;

 hold on;


plot(vectoraproxlinealFLP(1,iii),vectoraproxlinealFLP(2,iii),'*','MarkerEdgeCo
lor','m','LineWidth',1);

end



xaproxlinealFLA=vectoraproxlinealFLA(1,1):1:vectoraproxlinealFLA(1,nFLA);

yaproxlinealFLA=mFLA*xaproxlinealFLA+bFLA;

xaproxlinealFLM=vectoraproxlinealFLM(1,1):1:vectoraproxlinealFLM(1,nFLM);

yaproxlinealFLM=mFLM*xaproxlinealFLM+bFLM;

xaproxlinealFLH=vectoraproxlinealFLH(1,1):1:vectoraproxlinealFLH(1,nFLH);
```

```
yaproxlinealFLH=mFLH*xaproxlinealFLH+bFLH;

xaproxlinealFLS=vectoraproxlinealFLS(1,1):1:vectoraproxlinealFLS(1,nFLS);

yaproxlinealFLS=mFLS*xaproxlinealFLS+bFLS;

xaproxlinealFLC=vectoraproxlinealFLC(1,1):1:vectoraproxlinealFLC(1,nFLC);

yaproxlinealFLC=mFLC*xaproxlinealFLC+bFLC;

xaproxlinealFLP=vectoraproxlinealFLP(1,1):1:vectoraproxlinealFLP(1,nFLP);

yaproxlinealFLP=mFLP*xaproxlinealFLP+bFLP;

plot(xaproxlinealFLA,yaproxlinealFLA,'b',xaproxlinealFLM,yaproxlinealFLM,'c',x
aproxlinealFLH,yaproxlinealFLH,'g',xaproxlinealFLS,yaproxlinealFLS,'k',xaproxl
inealFLC,yaproxlinealFLC,'r',xaproxlinealFLP,yaproxlinealFLP,'m');

title('Aproximació lineal per mínims quadrats FL.');


%% Plot Aprox. Lineal (mínims quadrats) de POT en funció de velocitat de totes
les dades

figure ('Name','Aproximació lineal PWMFR','NumberTitle','off','Color', [0.8
0.8 0.8])

vectoraproxlinealFRA=[AllspeedsFR;AllPotFR];

vectoraproxlinealFRM=[Allspeeds2FR;AllPot2FR];

vectoraproxlinealFRH=[Allspeeds3FR;AllPot3FR];

vectoraproxlinealFRS=[Allspeeds4FR;AllPot4FR];

vectoraproxlinealFRC=[Allspeeds5FR;AllPot5FR];

vectoraproxlinealFRP=[Allspeeds6FR;AllPot6FR];


global mFRA

mFRA

global bFRA

bFRA

global mFRM

mFRM

global bFRM

bFRM

global mFRH

mFRH

global bFRH
```

```
bFRH

global mFRS

mFRS

global bFRS

bFRS

global mFRC

mFRC

global bFRC

bFRC

global mFRP

mFRP

global bFRP

bFRP


save [mFRA,bFRA]=mincuadlin(vectoraproxlinealFRA);

save [mFRM,bFRM]=mincuadlin(vectoraproxlinealFRM);

save [mFRH,bFRH]=mincuadlin(vectoraproxlinealFRH);

save [mFRS,bFRS]=mincuadlin(vectoraproxlinealFRP);

save [mFRC,bFRC]=mincuadlin(vectoraproxlinealFRC);

save [mFRP,bFRP]=mincuadlin(vectoraproxlinealFRP);


nFRA=length(vectoraproxlinealFRA(1,:));

nFRM=length(vectoraproxlinealFRM(1,:));

nFRH=length(vectoraproxlinealFRH(1,:));

nFRS=length(vectoraproxlinealFRS(1,:));

nFRC=length(vectoraproxlinealFRC(1,:));

nFRP=length(vectoraproxlinealFRP(1,:));


AFRA=0;

AFRM=0;

AFRH=0;

AFRS=0;
```

```
AFRC=0;

AFRP=0;

BFRA=0;

BFRM=0;

BFRH=0;

BFRS=0;

BFRC=0;

BFRP=0;

CFRA=0;

CFRM=0;

CFRH=0;

CFRS=0;

CFRC=0;

CFRP=0;

DFRA=0;

DFRM=0;

DFRH=0;

DFRS=0;

DFRC=0;

DFRP=0;


for iii=1:nFRA;

 AFRA=AFRA+vectoraproxlinealFRA(1,iii);

 BFRA=BFRA+vectoraproxlinealFRA(2,iii);

 CFRA=CFRA+(vectoraproxlinealFRA(1,iii))^2;

 DFRA=DFRA+vectoraproxlinealFRA(1,iii)*vectoraproxlinealFRA(2,iii);

end


for iii=1:nFRM;

 AFRM=AFRM+vectoraproxlinealFRM(1,iii);

 BFRM=BFRM+vectoraproxlinealFRM(2,iii);

 CFRM=CFRM+(vectoraproxlinealFRM(1,iii))^2;
```

```
 DFRM=DFRM+vectoraproxlinealFRM(1,iii)*vectoraproxlinealFRM(2,iii);
end

for iii=1:nFRH;
 AFRH=AFRH+vectoraproxlinealFRH(1,iii);
 BFRH=BFRH+vectoraproxlinealFRH(2,iii);
 CFRH=CFRH+(vectoraproxlinealFRH(1,iii))^2;
 DFRH=DFRH+vectoraproxlinealFRH(1,iii)*vectoraproxlinealFRH(2,iii);
end

for iii=1:nFRS;
 AFRS=AFRS+vectoraproxlinealFRS(1,iii);
 BFRS=BFRS+vectoraproxlinealFRS(2,iii);
 CFRS=CFRS+(vectoraproxlinealFRS(1,iii))^2;
 DFRS=DFRS+vectoraproxlinealFRS(1,iii)*vectoraproxlinealFRS(2,iii);
end

for iii=1:nFRC;
 AFRC=AFRC+vectoraproxlinealFRC(1,iii);
 BFRC=BFRC+vectoraproxlinealFRC(2,iii);
 CFRC=CFRC+(vectoraproxlinealFRC(1,iii))^2;
 DFRC=DFRC+vectoraproxlinealFRC(1,iii)*vectoraproxlinealFRC(2,iii);
end

for iii=1:nFRP;
 AFRP=AFRP+vectoraproxlinealFRP(1,iii);
 BFRP=BFRP+vectoraproxlinealFRP(2,iii);
 CFRP=CFRP+(vectoraproxlinealFRP(1,iii))^2;
 DFRP=DFRP+vectoraproxlinealFRP(1,iii)*vectoraproxlinealFRP(2,iii);
end

mFRA=(nFRA*DFRA-AFRA*BFRA)/(nFRA*CFRA-AFRA^2);
```

```
bFRA=(CFRA*BFRA-DFRA*AFRA)/(nFRA*CFRA-AFRA^2);

mFRM=(nFRM*DFRM-AFRM*BFRM)/(nFRM*CFRM-AFRM^2);

bFRM=(CFRM*BFRM-DFRM*AFRM)/(nFRM*CFRM-AFRM^2);

mFRH=(nFRH*DFRH-AFRH*BFRH)/(nFRH*CFRH-AFRH^2);

bFRH=(CFRH*BFRH-DFRH*AFRH)/(nFRH*CFRH-AFRH^2);

mFRS=(nFRS*DFRS-AFRS*BFRS)/(nFRS*CFRS-AFRS^2);

bFRS=(CFRS*BFRS-DFRS*AFRS)/(nFRS*CFRS-AFRS^2);

mFRC=(nFRC*DFRC-AFRC*BFRC)/(nFRC*CFRC-AFRC^2);

bFRC=(CFRC*BFRC-DFRC*AFRC)/(nFRC*CFRC-AFRC^2);

mFRP=(nFRP*DFRP-AFRP*BFRP)/(nFRP*CFRP-AFRP^2);

bFRP=(CFRP*BFRP-DFRP*AFRP)/(nFRP*CFRP-AFRP^2);


for iii=1:nFRA;

 hold on;

plot(vectoraproxlinealFRA(1,iii),vectoraproxlinealFRA(2,iii),'*','MarkerEdgeCo
lor','b','LineWidth',1);

end

for iii=1:nFRM;

 hold on;

plot(vectoraproxlinealFRM(1,iii),vectoraproxlinealFRM(2,iii),'*','MarkerEdgeCo
lor','c','LineWidth',1);

end

for iii=1:nFRH;

 hold on;

plot(vectoraproxlinealFRH(1,iii),vectoraproxlinealFRH(2,iii),'*','MarkerEdgeCo
lor','g','LineWidth',1);

end

for iii=1:nFRS;

 hold on;

plot(vectoraproxlinealFRS(1,iii),vectoraproxlinealFRS(2,iii),'*','MarkerEdgeCo
lor','k','LineWidth',1);

end
```

```
for iii=1:nFRC;

 hold on;


plot(vectoraproxlinealFRC(1,iii),vectoraproxlinealFRC(2,iii),'*','MarkerEdgeCo
lor','r','LineWidth',1);

end

for iii=1:nFRP;

 hold on;


plot(vectoraproxlinealFRP(1,iii),vectoraproxlinealFRP(2,iii),'*','MarkerEdgeCo
lor','m','LineWidth',1);

end



xaproxlinealFRA=vectoraproxlinealFRA(1,1):1:vectoraproxlinealFRA(1,nFRA);

yaproxlinealFRA=mFRA*xaproxlinealFRA+bFRA;

xaproxlinealFRM=vectoraproxlinealFRM(1,1):1:vectoraproxlinealFRM(1,nFRM);

yaproxlinealFRM=mFRM*xaproxlinealFRM+bFRM;

xaproxlinealFRH=vectoraproxlinealFRH(1,1):1:vectoraproxlinealFRH(1,nFRH);

yaproxlinealFRH=mFRH*xaproxlinealFRH+bFRH;

xaproxlinealFRS=vectoraproxlinealFRS(1,1):1:vectoraproxlinealFRS(1,nFRS);

yaproxlinealFRS=mFRS*xaproxlinealFRS+bFRS;

xaproxlinealFRC=vectoraproxlinealFRC(1,1):1:vectoraproxlinealFRC(1,nFRC);

yaproxlinealFRC=mFRC*xaproxlinealFRC+bFRC;

xaproxlinealFRP=vectoraproxlinealFRP(1,1):1:vectoraproxlinealFRP(1,nFRP);

yaproxlinealFRP=mFRP*xaproxlinealFRP+bFRP;

plot(xaproxlinealFRA,yaproxlinealFRA,'b',xaproxlinealFRM,yaproxlinealFRM,'c',x
aproxlinealFRH,yaproxlinealFRH,'g',xaproxlinealFRS,yaproxlinealFRS,'k',xaproxl
inealFRC,yaproxlinealFRC,'r',xaproxlinealFRP,yaproxlinealFRP,'m');

title('Aproximació lineal per mínims quadrats FR.');
```

C. Programa Check Terrain

```
#include <ros/ros.h>

#include <iostream>

#include <math.h>

#include <std_msgs/Float32.h>

#include <std_msgs/Float64.h>

#include <std_msgs/Bool.h>

#include <geometry_msgs/Twist.h>

using namespace std;

//Intensities of the 4 motors. To be accessed at other places of the program

float int_fl;

float int_fr;

float int_bl;

float int_br;

float bat_lev;

float ctl_FL;

float ctl_FR;

float ctl_BL;

float ctl_BR;

float speedX;       //linear speed

float speedZ;       //angular speed

int ind = 0;

int indStDev = 0;

bool ArrayCompleted = false;     //this variable will tell when the array is
full and computations can start

bool stdDevArrCompleted = false;


//powerVariance variables

const int buffSize = 20;             //this variable determines the length of
the buffer, how many powers will be saved for computing the variance

float varArray[buffSize] = {};      //array of variance

const int buffStdDev = 20;               //size of the buffer of standard
deviations

float stDevArray[buffStdDev] = {};  //creation of the array for the stdDev
```

```
bool cleanFirstArray = true;          //this will empty the first array of
intensities since it is noise due to the stabilization of the PID signal



/* These callbacks will be used in order to extract the data sent
 * by the topics on ROS
 */
void I_flCallback(const std_msgs::Float64& msg){
    int_fl = msg.data;
}

void I_frCallback(const std_msgs::Float64& msg){
    int_fr = msg.data;
}

void I_blCallback(const std_msgs::Float64& msg){
    int_bl = msg.data;
}

void I_brCallback(const std_msgs::Float64& msg){
    int_br = msg.data;
}

void batCallback(const std_msgs::Float64& msg){
    bat_lev = msg.data;
}

void ctl_flCallback(const std_msgs::Float32::ConstPtr& msg){
    ctl_FL = msg->data;
}

void ctl_frCallback(const std_msgs::Float32::ConstPtr& msg){
    ctl_FR = msg->data;
}
```

```
void ctl_blCallback(const std_msgs::Float32::ConstPtr& msg){
    ctl_BL = msg->data;
}

void ctl_brCallback(const std_msgs::Float32::ConstPtr& msg){
    ctl_BR = msg->data;
}

void vel_Callback(const geometry_msgs::Twist::ConstPtr& msg){
    speedX = msg->linear.x;
    speedZ = msg->angular.z;
}

float* computePowers(){
//computing the powers for the 4 different wheels

    float* p = new float[4];

    p[0] = float((int_fl/1000.0)*bat_lev*ctl_FL);   //FL
    p[1] = float((int_fr/1000.0)*bat_lev*ctl_FR);   //FR
    p[2] = float((int_bl/1000.0)*bat_lev*ctl_BL);   //BL
    p[3] = float((int_br/1000.0)*bat_lev*ctl_BR);   //BR

    return p;
}

bool powerVariance(float* Pw){
/*
 * This function will provide information about the variance of the powers of
the different wheels.
 * The mean of the power of the 4 wheels will be computed and stored in an
array of X positions.
 * This wants to represent the variance of the power during time.
 *
 * If the Rate is set to 10 and there are 20 positions on the array it
represents 2 seconds of the
```

```
 * reality.

 *

 * If a big variance is observed for a certain terrain it will be necessary to
do a test to be sure

 * that the robot has changed or not the terrain.

 *

 * This variation will be determined by the standard deviation (given by the
squared root of the variance)

 * If the stdDev is changing and not constant during a certain period of time
it means that probably

 * the terrain has changed.

 *

 * */


    float Pw4w = 0;                     //Power of the four wheels

    float meanPw;

    float variance = 0;

    float stdDev = 0;

    bool testLaunch = false;             //this variable says if it is needed
to check the terrain

    float meanStdDev = 0;

    int outliers = 0;                        //number of stdDev that will be
outside the range. This will mark when the terrain has changed

    float thrMax = 1.3;                  //threshold for the stdDev. Right now
it is using a 30%. If a value is outside this thr it will be considered an
outlier

    float thrMin = 0.7;

    int maxOutliers = 7;                //set the maximum outliers allowed


    Pw4w = ( Pw[0] + Pw[1] + Pw[2] + Pw[3] ) / 4;   //power of the 4 wheels


    if (speedX == 0){

        /* If the linear speed is equal to 0 it means that the test has to
begin

         * again. In order to do that it is necessary to reset the arrays to 0
and

         * to set again the indices to 0. It is also necessary to set the
flags of

         * the arrays to 'false' so the program knows that it has to complete
again
```

```
         * them before doing new computations.
         */

        for(int jj=0; jj<buffSize; jj++){
            varArray[jj] = 0;
        }
        for(int hh=0; hh<buffStdDev; hh++){
            stDevArray[hh] = 0;
        }
        ind = 0;
        indStDev = 0;
        ArrayCompleted = false;
        stdDevArrCompleted = false;
        cleanFirstArray = true;
    }

    varArray[ind] = Pw4w;        //allocating the mean of the 4 powers in the
array
    ind++;

    if (ArrayCompleted && speedZ==0 && speedX>0){       //limit the program to
work only when the robot goes forward

        for (int i=0; i<buffSize; i++){     //compute the mean power
            meanPw = meanPw + varArray[i];
        }
        meanPw = meanPw / buffSize;

        for (int j=0; j<buffSize; j++){     //compute the variance
            variance = variance + pow((varArray[j] - meanPw), 2.0);
        }
        variance = variance / buffSize;   //averaging the variance

        stdDev = sqrt(variance);            //compute the standard deviation
```

```
        stDevArray[indStDev] = stdDev;            //allocating the standard
deviations in the array

        indStDev++;



        if (stdDevArrCompleted){



            for (int k=0; k<buffStdDev; k++){               //compute the mean
of the standard deviation

                meanStdDev = meanStdDev + stDevArray[k];

            }

            meanStdDev = meanStdDev / buffStdDev;



            for (int l=0; l<buffStdDev; l++){                   //checking the
outliers from the mean

                if             (stDevArray[l]>meanStdDev*thrMax             ||
stDevArray[l]<meanStdDev*thrMin){

                    outliers++;

                }

                //cout << stDevArray[l] << " ";

            }

            //cout << endl;

            cout << "Outliers: " << outliers << "   " << "Current standard
deviation: " << stdDev << " FL: " << Pw[0] << " FR: " << Pw[1] << " BL: " <<
Pw[2] << " BR: " << Pw[3] << endl;



            if (outliers > maxOutliers){

                testLaunch = true;

                cout << "Change of terrain detected!!!" << "  Outliers: " <<
outliers << endl;

            }

        }

    }



    //if the index is greater than 19 it means that we are outside of the
array

    //so it must be set to 0 again

    if (ind == buffSize){

        ind = 0;

        ArrayCompleted = true;
```

```
    }

    if (indStDev == buffStdDev){
        indStDev = 0;
        stdDevArrCompleted = true;

        if(cleanFirstArray==true){                //first set of stdDev suffer
from stabilization of PID signal so they must be "destroied"

            for(int jjj=0; jjj<buffSize; jjj++){
                varArray[jjj] = 0;
            }
            for(int hhh=0; hhh<buffStdDev; hhh++){
                stDevArray[hhh] = 0;
            }
            ind = 0;
            indStDev = 0;

            ArrayCompleted = false;
            stdDevArrCompleted = false;
            cleanFirstArray = false;
            //cout << "Reset the parameters" << endl;
        }
    }
return testLaunch;
}

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

    ros::init(argc, argv, "check_terrain");
    ros::NodeHandle m;

    /* Declaration of all the subscribers that will be needed during the
     * all the computations of this node*/
```

```
    ros::Subscriber I_fl = m.subscribe("Current_Front_Left", 100,
I_flCallback);

    ros::Subscriber I_fr = m.subscribe("Current_Front_Right", 100,
I_frCallback);

    ros::Subscriber I_bl = m.subscribe("Current_Back_Left", 100,
I_blCallback);

    ros::Subscriber I_br = m.subscribe("Current_Back_Right", 100,
I_brCallback);


    ros::Subscriber battery = m.subscribe("Battery_Level", 100, batCallback);


    ros::Subscriber ctl_fl = m.subscribe("Control_Signal_Front_Left", 100,
ctl_flCallback);

    ros::Subscriber ctl_fr = m.subscribe("Control_Signal_Front_Right", 100,
ctl_frCallback);

    ros::Subscriber ctl_bl = m.subscribe("Control_Signal_Back_Left", 100,
ctl_blCallback);

    ros::Subscriber ctl_br = m.subscribe("Control_Signal_Back_Right", 100,
ctl_brCallback);


    ros::Subscriber vel = m.subscribe("cmd_vel",100,vel_Callback);


    //Declaration of the publisher that will tell 'base_controller' to launch
the terrain test

    ros::Publisher pub = m.advertise<std_msgs::Bool>("runTest",0);

    std_msgs::Bool runTest;


    ros::Rate loop_rate(20);

    while (ros::ok()){


        float* Pw = computePowers();


        bool changeDetected = powerVariance(Pw);


        ros::spinOnce();

        loop_rate.sleep();


        //if a change is detected the message is published
```

```
        if (changeDetected){
            runTest.data = 1;
            pub.publish(runTest);
        }
    }

    return 0;
}
```

D. Programa Turn90 (gir cap a l'esquerra i cap a la dreta)

```
#include <ros/ros.h>
#include <iostream>
#include <math.h>
#include <std_msgs/Float32.h>
#include <sensor_msgs/Joy.h>
#include <geometry_msgs/Twist.h>
using namespace std;

//variable for storing the number of counts of the encoder
int counter = 0;
int totalCount = 0;
bool turnFlag = false;
bool turnFinished = false;
int terrain_type = 1;
bool turn_dreta = false;
bool turn_esquerra = false;
bool gostraight = true;
int travelled = 0;
int turns = 0;

void Enc_R_Callback(const std_msgs::Float32& msg){
    counter = msg.data;
}

void terrainCallback(const std_msgs::Float32& msg){
    terrain_type = msg.data;
}

void joyCallback (const sensor_msgs::Joy::ConstPtr& joy_data){
/* This function responds to any change on the state of the joystick. It
 * is mainly used to detect when the buttons are pressed and act according
 * the one that has been pressed
 */
```

```
	if (joy_data->buttons[5] == 1){  //this button will restart the PID
		turn_dreta = true;
	}
	if (joy_data->buttons[6] == 1){  //this button will restart the PID
		turn_esquerra = true;
	}
	if (joy_data->buttons[7] == 1){  //this button will restart the PID
		turn_esquerra = false;
	}
	if (joy_data->buttons[7] == 1){  //this button will restart the PID
		turn_dreta = false;
	}
}
void turn(){

	float threshold;

	cout << "Counter: " << counter << " totalCounter: " << totalCount <<
endl;
	totalCount = totalCount + counter;

	switch (terrain_type){

		case 1:
			threshold = totalpulse*2.05;
			break;
		case 2:
			threshold = totalpulse*2.16;
			break;
		case 3:
			threshold = totalpulse*2.91;
			break;
		case 4:
			threshold = totalpulse*2.16;
			break;
```

```
		case 5:
				threshold = totalpulse*2.09;
				break;
		case 6:
				threshold = totalpulse*2.059;
				break;
		case 7:
				threshold = totalpulse*2.05;
				break;
		case 8:
				threshold = totalpulse*2.15;
				break;
		case 9:
				threshold = totalpulse*2.15;
				break;

	}

	if (totalCount > threshold){
		turnFinished = true;
	}
}

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

    ros::init(argc, argv, "turn90");
    ros::NodeHandle handle;


    /* Declaration of all the subscribers that will be needed during the
     * all the computations of this node*/
      ros::Subscriber    I_fl    =    handle.subscribe("Encoder_Right",    100,
Enc_R_Callback);

      ros::Subscriber joy_sub = handle.subscribe("joy",0,joyCallback);
```

```
      ros::Publisher                          speed                          =
handle.advertise<geometry_msgs::Twist>("cmd_vel",0);

    geometry_msgs::Twist cmd_vel;


    ros::Subscriber                      terrain_type                       =
handle.subscribe("Terrain_type",0,terrainCallback);


    ros::Rate loop_rate(20);


    while (ros::ok()){

        if (turn_dreta){

                    int totalpulse = 1000;

                    if (gostraight){

                           cmd_vel.linear.x = 0.4;

                           cmd_vel.angular.z = 0;

                           speed.publish(cmd_vel);

                           travelled = travelled + counter;

                           cout << travelled << endl;

                           if (travelled > 5000){

                                  gostraight = false;

                           }


                    }else{

                           travelled = 0;

                           cmd_vel.linear.x = 0;

                           cmd_vel.angular.z = 0.4;

                           speed.publish(cmd_vel);

                           turn();

                           if(turnFinished){

                                  cmd_vel.angular.z = 0;

                                  totalCount = 0;

                                  speed.publish(cmd_vel);

                                  turnFinished = false;

                                  gostraight = true;

                                  turns = turns + 1;
```

```
                }
            }
            if (turns == 1){
                turns = 0;
                turn_dreta = false;
            }
        }

ros::spinOnce();
loop_rate.sleep();

    if (turn_esquerra){
        int totalpulse = 5750;
        if (gostraight){
            cmd_vel.linear.x = 0.4;
            cmd_vel.angular.z = 0;
            speed.publish(cmd_vel);
            travelled = travelled + counter;
            cout << travelled << endl;
            if (travelled > 5000){
                gostraight = false;
            }

        }else{
            travelled = 0;
            cmd_vel.linear.x = 0;
            cmd_vel.angular.z = -0.4;
            speed.publish(cmd_vel);
            turn();
            if(turnFinished){
                cmd_vel.angular.z = 0;
                totalCount = 0;
                speed.publish(cmd_vel);
                turnFinished = false;
                gostraight = true;
```

```
                                turns = turns + 1;
                        }
                }
                if (turns == 1){
                        turns = 0;
                        turn_esquerra = false;
                }
        }

    ros::spinOnce();
    loop_rate.sleep();

}

return 0;
}
```

F. Programa Circuit quadrat (4 girs cap a l'esquerra)

```
#include <ros/ros.h>
#include <iostream>
#include <math.h>
#include <std_msgs/Float32.h>
#include <sensor_msgs/Joy.h>
#include <geometry_msgs/Twist.h>
using namespace std;

//variable for storing the number of counts of the encoder
int counter = 0;
int totalCount = 0;
bool turnFlag = false;
bool turnFinished = false;
int terrain_type = 1;
bool gostraight = true;
int travelled = 0;
int turns = 0;
void Enc_R_Callback(const std_msgs::Float32& msg){
    counter = msg.data;
}

void terrainCallback(const std_msgs::Float32& msg){
    terrain_type = msg.data;
}
void joyCallback (const sensor_msgs::Joy::ConstPtr& joy_data){
/* This function responds to any change on the state of the joystick. It
 * is mainly used to detect when the buttons are pressed and act according
 * the one that has been pressed
 */
      if (joy_data->buttons[5] == 1){  //this button will restart the PID
            turnFlag = true;
      }
      if (joy_data->buttons[6] == 1){
```

```
		turnFlag = false;
		cout <<  "Boto 4: desactiva" << endl;
}
void turn(){
	float threshold;
	int totalpulse = 5750;
	cout << "Counter: " << counter << " totalCounter: " << totalCount << endl;
	totalCount = totalCount + counter;

	switch (terrain_type){

		case 1:
			threshold = totalpulse*2.05;
			break;
		case 2:
			threshold = totalpulse*2.16;
			break;
		case 3:
			threshold = totalpulse*2.91;
			break;
		case 4:
			threshold = totalpulse*2.16;
			break;
		case 5:
			threshold = totalpulse*2.09;
			break;
		case 6:
			threshold = totalpulse*2.059;
			break;
		case 7:
			threshold = totalpulse*2.05;
			break;
		case 8:
			threshold = totalpulse*2.15;
```

```
                        break;
            case 9:
                        threshold = totalpulse*2.15;
                        break;

      }
      if (totalCount > threshold){
            turnFinished = true;
      }
}
int main(int argc, char** argv){
    ros::init(argc, argv, "turn90");
    ros::NodeHandle handle;
    /* Declaration of all the subscribers that will be needed during the
     * all the computations of this node*/
      ros::Subscriber    I_fl    =    handle.subscribe("Encoder_Right",    100,
Enc_R_Callback);
      ros::Subscriber joy_sub = handle.subscribe("joy",0,joyCallback);
      ros::Publisher                              speed                              =
handle.advertise<geometry_msgs::Twist>("cmd_vel",0);
    geometry_msgs::Twist cmd_vel;
    ros::Subscriber                        terrain_type                        =
handle.subscribe("Terrain_type",0,terrainCallback);

    ros::Rate loop_rate(20);
    while (ros::ok()){

        if (turnFlag){
                    if (gostraight){
                        cmd_vel.linear.x = 0.4;
                        cmd_vel.angular.z = 0;
                        speed.publish(cmd_vel);
                        travelled = travelled + counter;
                        cout << travelled << endl;
                        if (travelled > 5000){
                              gostraight = false;
```

```
                        }
                    }else{
                        travelled = 0;
                        cmd_vel.linear.x = 0;
                        cmd_vel.angular.z = -0.4;
                        speed.publish(cmd_vel);
                        turn();
                        if(turnFinished){
                            cmd_vel.angular.z = 0;
                            totalCount = 0;
                            speed.publish(cmd_vel);
                            turnFinished = false;
                            gostraight = true;
                            turns = turns + 1;
                        }
                    }

                    if(turnFlag == false){
                        cmd_vel.linear.x = 0;
                        cmd_vel.angular.z = 0;
                        speed.publish(cmd_vel);
                    }

                    if (turns == 4){
                        turns = 0;
                        turnFlag = false;
                    }
                }
            ros::spinOnce();
            loop_rate.sleep();
        }
        return 0;
}
```

G. Programa gir circular (gir cap a l'esquerra i cap a la dreta)

```
#include <ros/ros.h>
#include <iostream>
#include <math.h>
#include <std_msgs/Float32.h>
#include <sensor_msgs/Joy.h>
#include <geometry_msgs/Twist.h>
using namespace std;
//variable for storing the number of counts of the encoder
int counter = 0;
int totalCount = 0;
bool turnFlag = false;
bool turnFinished = false;
int terrain_type = 1;
bool turn_dreta = false;
bool turn_esquerra = false;
bool gostraight = true;
int travelled = 0;
int turns = 0;

void Enc_R_Callback(const std_msgs::Float32& msg){
    counter = msg.data;
}
void terrainCallback(const std_msgs::Float32& msg){
    terrain_type = msg.data;
}
void joyCallback (const sensor_msgs::Joy::ConstPtr& joy_data){
/* This function responds to any change on the state of the joystick. It
 * is mainly used to detect when the buttons are pressed and act according
 * the one that has been pressed
 */
      if (joy_data->buttons[5] == 1){  //this button will restart the PID
             turn_dreta = true;
      }
```

```
	if (joy_data->buttons[6] == 1){  //this button will restart the PID
		turn_esquerra = true;
	}
	if (joy_data->buttons[7] == 1){  //this button will restart the PID
		turn_esquerra = false;
	}
	if (joy_data->buttons[7] == 1){  //this button will restart the PID
		turn_dreta = false;
	}
}
void turn(){
	float threshold;
	cout << "Counter: " << counter << " totalCounter: " << totalCount << endl;
	totalCount = totalCount + counter;

	switch (terrain_type){

		case 1:
			threshold = totalpulse*2.17;
			break;
		case 2:
			threshold = totalpulse*1.805;
			break;
		case 3:
			threshold = totalpulse*2.61;
			break;

	}

	if (totalCount > threshold){
		turnFinished = true;
	}
}
int main(int argc, char** argv){
```

```
    ros::init(argc, argv, "turn90");

    ros::NodeHandle handle;

    /* Declaration of all the subscribers that will be needed during the

     * all the computations of this node*/

      ros::Subscriber    I_fl    =    handle.subscribe("Encoder_Right",    100,
Enc_R_Callback);

      ros::Subscriber joy_sub = handle.subscribe("joy",0,joyCallback);

      ros::Publisher                              speed                              =
handle.advertise<geometry_msgs::Twist>("cmd_vel",0);

    geometry_msgs::Twist cmd_vel;

    ros::Subscriber                          terrain_type                          =
handle.subscribe("Terrain_type",0,terrainCallback);

    ros::Rate loop_rate(20);

    while (ros::ok()){




      if (turn_dreta){

                  int totalpulse = 2000;

                  if (gostraight){

                        cmd_vel.linear.x = 0.4;

                        cmd_vel.angular.z = 0;

                        speed.publish(cmd_vel);

                        travelled = travelled + counter;

                        cout << travelled << endl;

                        if (travelled > 5000){

                              gostraight = false;

                        }


                  }else{

                        travelled = 0;

                        cmd_vel.linear.x = 0.4;

                        cmd_vel.angular.z = 0.4;

                        speed.publish(cmd_vel);

                        turn();

                        if(turnFinished){
```

```
                        cmd_vel.angular.z = 0;
                        totalCount = 0;
                        speed.publish(cmd_vel);
                        turnFinished = false;
                        gostraight = true;
                        turns = turns + 1;
                }
        }
        if (turns == 1){
                turns = 0;
                turn_dreta = false;
        }
    }
ros::spinOnce();
loop_rate.sleep();
    if (turn_esquerra){
        int totalpulse = 11500;
        if (gostraight){
                cmd_vel.linear.x = 0.4;
                cmd_vel.angular.z = 0;
                speed.publish(cmd_vel);
                travelled = travelled + counter;
                cout << travelled << endl;
                if (travelled > 5000){
                        gostraight = false;
                }

        }else{
                travelled = 0;
                cmd_vel.linear.x = 0.4;
                cmd_vel.angular.z = -0.4;
                speed.publish(cmd_vel);
                turn();
                if(turnFinished){
                        cmd_vel.angular.z = 0;
```

```
                                totalCount = 0;
                                speed.publish(cmd_vel);
                                turnFinished = false;
                                gostraight = true;
                                turns = turns + 1;
                        }
                }
                if (turns == 1){
                        turns = 0;
                        turn_esquerra = false;
                }
            }
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```