Universitat de Vic
Escola Politècnica Superior

Treball de Fi de Grau

Introducció a ROS

Guiu Riera Riera

Grau en Enginyeria Mecatrònica

Tutors: Moisès Serra Serra, Juli Ordeix Rigo

Vic, juny de 2016

Índex

RESUM TREBALL FINAL DE GRAU
GRAU EN ENGINYERIA MECATRÒNICA

Títol: *Introducció a ROS (Robotic Operating System)*
Paraules clau: ROS, robòtica mòbil, anàlisi de l'entorn, control remot, Leap Motion
Autor: Guiu Riera Riera
Tutors: Moisès Serra Serra (UVic), Juli Ordeix Rigo (UVic)
Data: Juny de 2016

En aquest *Treball de Final de Grau* es fa una introducció al sistema ROS *"Robotic Operating System"* sorgit com a motivació personal en l'ús de noves eines de programació de robots, molt avançat en robòtica mòbil i de serveis, però poc aplicat a nivell industrial. El projecte pretén ser una base per a qualsevol que es vulgui iniciar en aquest món, per això després de fer una introducció teòrica es pretén demostrar a nivell físic les possibilitats que ofereix ROS, per això es construirà una plataforma .

L'objectiu principal del projecte és el d'oferir una introducció clara i entenedora del que significa ROS i com poder configurar les màquines utilitzades des de zero, de tal manera que el lector pugui començar a utilitzar aquesta eina només amb la lectura d'aquest treball, per tant també pretén ser un tutorial de com es pot aplicar ROS en la robòtica mòbil. Com a objectiu secundari, però també important, es planteja el repte de construir la plataforma mòbil per poder exemplificar de manera real el que s'explica en la part teòrica. Com a tema innovador es vol utilitzar un Leap Motion com a control remot de la plataforma, i no un joystick o simplement el teclat de l'ordinador.

Els resultats del projecte demostren que a diferència de les eines clàssiques utilitzades en robòtica, ROS permet desenvolupar robots de manera senzilla i econòmica i que qualsevol amant de la robòtica pugui construir i programar el seu prototip sense la necessitat d'estar en un laboratori o en un recinte industrial, jaque ROS és un sistema de codi obert, sense la necessitat d'haver de pagar per llicències de programari i que qualsevol pot participar en l'evolució i millora d'aquest sistema. També es demostra que es pot controlar una plataforma mòbil sense la interacció amb sistemes mecànics, simplement amb el moviment de la mà, tot gràcies al Leap Motion.

SUMMARY FINAL GRADE
DEGREE IN MECHATRONICS

Title: Introduction to ROS (Robotic Operating System)
Keywords: ROS, mobile robotics, environment analysis, remote control, Leap Motion
Author: Guiu Riera Riera
Tutors: Moisès Serra Serra (UVic), Juli Ordeix Rigo (UVic)
Date: June 2016

This Final Grade is an introduction to the ROS system *"Robotic Operating System"* emerged as motivation in the use of new programming tools and design of robots, advanced in mobile robotics and service robotics, but little applied at industrial level. The project is intended as a base for anyone who wants to start in this world and can be a starting point, so after making a theoretical introduction is intended to show the physical or tangible possibilities offered by this system, in order to doing so may be designed and built a mobile platform for an introduction to what can be achieved.

Thus, the main objective is to provide a clear and understandable introduction of what it means and how to configure ROS used machines from scratch, so that the reader can start using this tool only reading this work, therefore also intended to be a tutorial on how ROS can be applied to mobile robotics. As a secondary objective, but also important is the challenge to design and build mobile platform to illustrate in the real way what is explained in the theoretical part. As a matter innovative Leap Motion wants to be used as a remote control platform, not just a joystick or the computer keyboard.

The project results show that unlike the classic tools used in robotics to develop robots, ROS so simple and inexpensive that any lover of robotics and programming can build self prototype without the need to be in an experienced laboratory or in an industrial area, this is because ROS is an open source system, without having to make costly investments in software license fees and anyone can participate in the development and improvement of this system. It also shows that you can control a mobile platform without interaction with mechanical systems, simply moving the hand, thanks to the Leap Motion.

2. Introducció

2.1. Introducció

Introducció a ROS (Robotic Operating System), com molt bé diu el títol, aquest projecte pretén fer una introducció bàsica al funcionament de ROS, ja que és un sistema relativament nou i que de moment només s'utilitza en laboratoris d'investigació (com per exemple a Eurecat, Centre Tecnològic de Catalunya), sobretot destinat a la robòtica mòbil (tant terrestre, com aèria (drones)), però cada vegada se li estant buscant més aplicacions a nivell industrial. Es busca que el lector es pugui fer una idea del que és ROS, com funciona i quines aplicacions pot tenir més enllà de la robòtica mòbil que és l'aplicació que es busca en aquest projecte.

Durant el desenvolupament del projecte es pretén construir una plataforma mòbil (comunament coneguda com a "turtlebot" per la seva forma) per ésser controlada remotament en un entorn no controlat. Per això es buscarà quins sensors són més adients per ser utilitzats. Bàsicament està compost per un ordinador amb Linux/Ubuntu on s'instal·larà ROS, rebrà informació d'un Leap Motion per ser processada i definir els moviments de la plataforma, un cop processada la informació enviarà dades a la controladora (Raspberry Pi 2 i Arduino Mega) que es comunicarà amb els motors per realitzar el moviment necessari i llegirà la informació dels sensors per actuar en conseqüència. Per fer-se una idea global del que significa veure la figura 2.1.

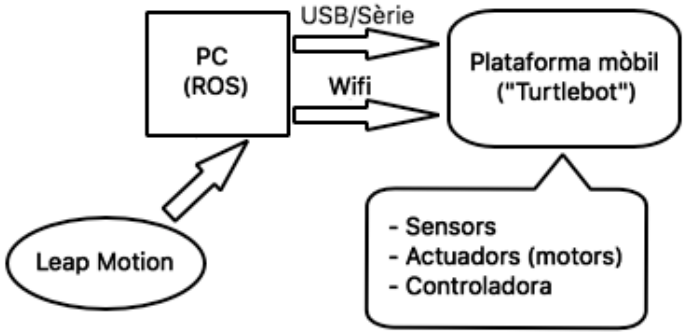

Figura 2.1. Diagrama general del muntatge a realitzar

La motivació principal per dur a terme un projecte d'aquestes característiques és perquè des de ben petit sempre he estat interessat en el món de la robòtica, i després de fer un curs introductori a l'entorn de programació ROS a ASCAMM

(actual Eurecat), m'ha fascinat la idea de poder treballar amb robòtica mòbil. També he escollit ROS pel fet de ser un sistema obert i gratuït, el que fa que sigui assequible per a tothom i que hi hagi molta informació i paquets de programes que es poden utilitzar, ja sigui per control, simulació, comunicació...
El Leap Motion és un petit dispositiu que es connecta a un ordinador amb un cable USB. Aquest dispositiu mitjançant dues càmeres IR treballant en estèreo permet la detecció dels dits, les mans i l'avantbraç, així com la seva posició i orientació en tres dimensions. En l'apartat de hardware es fa una explicació més detallada del dispositiu.

2.2. Objectius

Per tal de fer visible i tangible aquesta introducció, l'objectiu principal del projecte és construir una plataforma mòbil que pugui ser controlada amb les eines de desenvolupament que ofereix ROS. Amb aquest projecte no es pretén que el lector sigui un expert en ROS un cop finalitzada la lectura, però si que es busca que comprengui el que significa i que serveixi com a part introductòria per poder tenir una guia d'inicialització i seguir investigant per si se'n vol aprendre més. Així doncs, els objectius que es busca complir són els llistats a continuació:

- Introducció al sistema ROS (Robotic Operating System)
- Instal·lació i funcionament de les eines ROS
- Construcció i programació plataforma mòbil
- Controlar plataforma utilitzant Leap Motion
- Buscar sensors més adients per evasió d'obstacles
- Comunicació Wifi entre ordinador i plataforma mòbil
- Programar sistema SLAM (Simultaneous Localization And Mapping)

Un cop finalitzat el projecte es podrà comprovar si s'ha pogut complir amb tots els objectius o no.

2.3. Estructura de la memòria

Per tal que el lector tingui una idea clara en tot moment del projecte s'ha decidit dividir la memòria en els següents punts:

- Fonaments teòrics: en aquest apartat es pretén que el lector es faci una idea de què és i com funciona ROS. També s'explicarà com instal·lar-lo i els primers passos per posar-lo en funcionament

- Desenvolupament del Projecte: un cop entès el funcionament de ROS s'explica com s'ha elaborat la part física del projecte, tant a nivell hardware (tot el material utilitzat, tant mecànic com electrònic) com a software (tot el programari utilitzat), quines solucions s'han adoptat i perquè
- Resultats: es fa una mirada enrere al projecte i s'intenta justificar el compliment, o no, dels objectius marcats
- Conclusions: es pretén tenir una visió global de tot el projecte per veure què ha fallat i què ha funcionat i fer una valoració del projecte
- Futures aplicacions: fer una reflexió de tot el coneixement adquirit al realitzar aquest projecte i mirar cap a on es pot enfocar, que no quedi només amb un projecte

3. Fonaments Teòrics

Abans d'entrar en matèria i poder situar al lector en el que s'ha treballat és necessari fer una introducció a ROS i com funciona, així després serà més fàcil d'entendre tot el desenvolupament i el que es planteja.

3.1. Introducció a ROS

ROS és un sistema operatiu de codi obert i lliure distribució, destinat a la robòtica, ja sigui de serveis, industrial, mòbil... Va ser ideat al laboratori d'intel·ligència artificial de la universitat de Stanford, després va ser desenvolupat i mantingut per Willow Garge. Excepte la interfície gràfica o escriptori que tothom està acostumat d'un sistema operatiu (Windows o Mac), ROS reuneix totes les qualitats per ser-ho:

- Abstracció del hardware
- Baix nivell del control de dispositius
- Implementa les funcionalitats més comunes
- Intercanvi de missatges entre processos
- Gestió de paquets o programari
- Conté llibreries per desenvolupar i fer córrer programes en diversos ordinadors

Tot i reunir les característiques d'un sistema operatiu no pot funcionar tot sol i ha de ser instal·lat sobre un altre sistema operatiu, actualment pot córrer en Windows, Mac OS X, Linux i sistemes ARM basats en Linux (com per exemple Raspbian per la Raspberry Pi, Angstorm per la Beagle Bone Black), tot i que els desenvolupadors només recomanen i donen suport a Ubuntu (Linux), per això s'ha escollit Ubuntu com a sistema operatiu bàsic pel desenvolupament d'aquest projecte. Per tenir una idea més clara veure la figura 3.1.1.

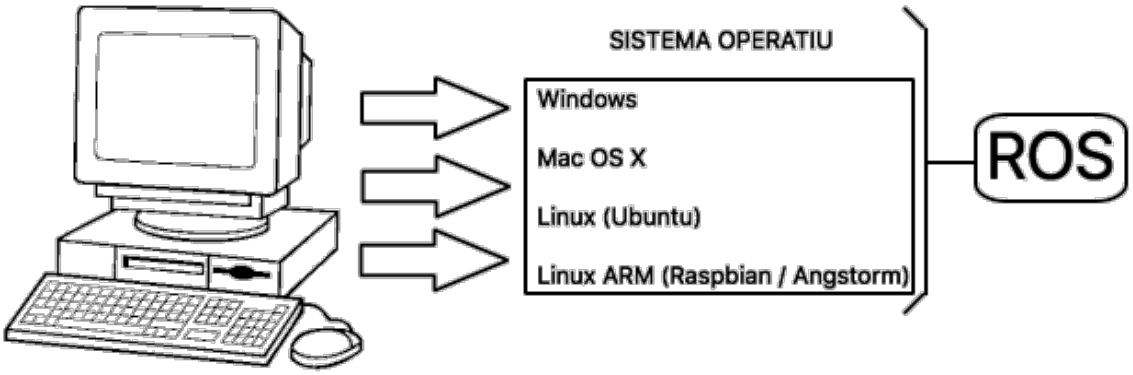

Figura 3.1.1. ROS dins un ordinador

ROS es base en la comunicació peer-to-peer (P2P o comunicació entre punts/ordinadors, figura 3.1.2) a través d'una xarxa LAN (amb o sense connexió a internet). Implementa diferents estils de comunicació, incloent comunicació síncrona RPC a través de *serveis* (envia una petició i espera una resposta), asíncrona a través de temes o *topics* segons ROS (envia missatges a la xarxa perquè algú els escolti) i emmagatzematge de dades al *Servidor de Paràmetres* (per poder ser llegits o modificats). ROS no és un sistema pensat per treballar en temps real, tot i que hi ha sistemes/aplicacions que utilitzen comunicació en temps real.

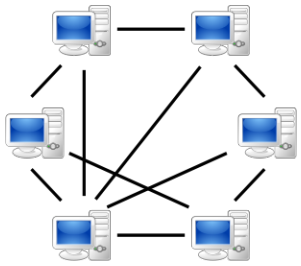

Figura 3.1.2. Exemple comunicació peer-to-peer

Un cop s'ha agafat una idea de com treballa ROS es pot procedir a escollir i instal·lar el sistema operatiu base, per després procedir a la instal·lació de ROS, a la pàgina web *wiki.ros.org* hi ha els passos a seguir per fer una instal·lació correcte en el nostre equip. Segons la versió del sistema operatiu s'ha d'instal·lar la versió adequada de ROS. Com que l'ordinador de que es disposa per la

realització del projecte conté Ubuntu 12.04 (Precise) la versió de ROS escollida és la Hydro.

Un cop configurat correctament, el següent pas és crear el nostre espai de treball, on hi guardarem tots els nostres programes o *paquets*, en la nomenclatura de ROS. Per fer-ho es pot seguir el tutorial número 3 a la mateixa wiki de ROS (*http://wiki.ros.org/ROS/Tutorials*)

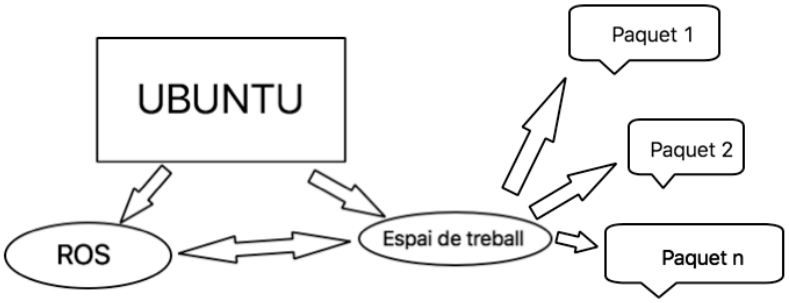

Figura 3.1.3. Esquema entorn de treball configurat

A dins de cada paquet hi guardarem els nostres programes o executables, ROS els anomena *nodes,* que són els que intercanvien missatges entre ells i contenen la programació per realitzar determinades tasques per les quals han estat creats (moure motors, planificar moviments, processar imatges...), així cada paquet pot contenir un o varis nodes. Els nodes es poden programar en tres llenguatges diferents: C++, Python o Lisp (els tutorials de la wiki només ensenyen a programar nodes en C++ i Python, que són els llenguatges més comuns). Com que la comunicació és entre punts (P2P) els nodes poden està funcionant en diferents màquines dins la mateixa xarxa, però només amb una restricció, una de les màquines ha d'actuar com a servidor per poder gestionar totes les comunicacions entre nodes, per tal que tots siguin accessibles, és el que s'anomena *roscore* i la resta de màquines han d'estar configurades perquè sàpiguen qui és el mestre (*master* o *roscore*).

3.2. Conceptes importants en ROS

Un cop ha quedat clar què és ROS, quina estructura té, com es comunica i com es programa, s'han de deixar clars un seguit de conceptes importants, però primer veure la figura 3.2.1 on es pot veure la relació que mantenen tots els conceptes que s'explicaran.

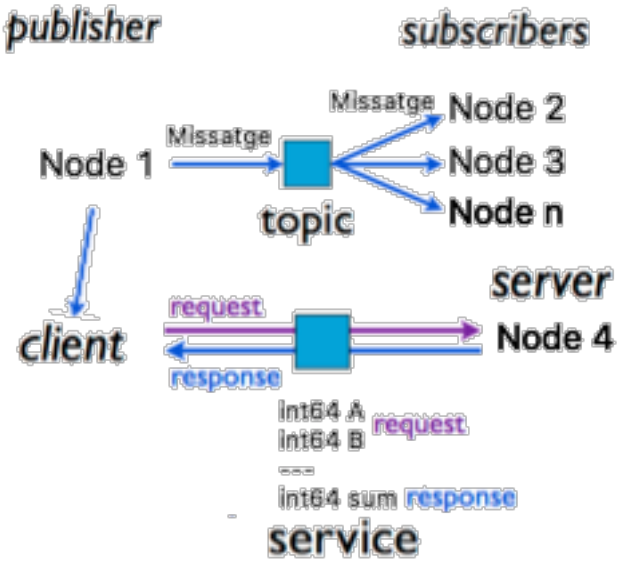

Figura 3.2.1. Relació entre els diferents conceptes de ROS

- Nodes: un node és un procés que realitza diferents tipus de computació. Els nodes es combinen entre ells en un gràfic i es comuniquen utilitzant *topics* en streaming, RPC *services* i el *Parameter Server*. Un robot pot utilitzar diversos nodes (un per controlar la distància amb un escàner làser, un altre el moviment de les rodes, un altre per planificar ruta...). D'aquesta manera ROS permet simplificar la programació, ja que cada node només conté la part de programa que afecta a la seva tasca, un altre benefici és que en un sistema no tots els nodes han d'estar programats en el mateix llenguatge, ja que ROS ofereix llibreries perquè un node escrit en C++ pugui comunicar-se amb un altre escrit en Pyhton sense problemes. El que és important és que cada node tingui un nom únic que l'identifiqui dins el gràfic de nodes per així poder evitar problemes.
- *Topics* o Temes: també s'anomenen busos per on els nodes intercanvien missatges. Els *topics* mantenen l'anonimat sobre qui hi està publicant o llegint els missatges, d'aquesta manera cada node no s'ha de preocupar amb qui es comunica. Pot haver-hi diversos nodes que publiquen i/o llegeixen d'un mateix *topic*. Els *topics* estant pensats per realitzar una comunicació unidireccional (és a dir, un mateix node no pot publicar i llegir missatges d'un mateix *topic*, o és publicador o subscriptor, no ambdós alhora). Si un node necessita realitzar un procés remot a partir d'una crida

(per exemple, rebre una resposta a partir d'una petició) ha d'utilitzar els *services*. També hi ha el *Parameter Server* on s'hi pot guardar una petita quantitat d'estats.

- *Messages* o Missatges: els nodes es comuniquen entre ells mitjançant la publicació de missatges als *topics*. Els missatges són simples estructures de dades, per exemple, tipus estàndard primitius (enters, reals, punts, booleans...), també arrays o cadenes de valors i cadenes de caràcters o *strings*. Els serveis també utilitzen els tipus de missatges per definir els paràmetres de sol·licitud i de resposta
- *Services* o Serveis: quan es disposa d'un sistema distribuït on l'estructura de publicador/subscriptor no és idònia per interaccions RPC de sol·licitud/resposta s'utilitzen els serveis. Els serveis estant compostos per parelles de missatges, un per la sol·licitud i l'altre per la resposta. Els nodes ROS ofereixen els serveis sota un nom en forma de *string*, i el client fa la crida al servei a partir d'enviar un missatge de sol·licitud i esperant la resposta. Els serveis estant definits en la carpeta ***srv*** dins de cada paquet que els utilitza, primer s'escriuen els paràmetres i del tipus que són (string, bool, int, float...) de la sol·licitud i després, separat per --- el paràmetre de resposta, que es compilen en codi per una llibreria client de ROS. Per altra banda, hi ha un node que fa de servidor, el qual rep els paràmetres de sol·licitud, opera amb ells i retorna una resposta; el node que fa de client envia la sol·licitud amb els paràmetres necessaris i espera que li retornin el paràmetre de resposta, ambdós han d'importar el fitxer ***srv*** per tal de poder-se comunicar.
- *Parameter Server* o Servidor de Paràmetres: és un servidor públic, diccionari multi variat que és accessible a través de APIs de xarxa. Els nodes utilitzen aquest servidor per guardar i rebre paràmetres mentre estant funcionant (*at runtime*). No està dissenyat per tenir una gran operativitat, per això s'utilitza bàsicament per guardar/modificar paràmetres de configuració de cada node. Està implementat utilitzant XMLRPC (*Remote Procedure* Call, crida a un procediment remot escrit en format XML (fitxers de codi escrits de fàcil interpretació per les persones i ordinadors)) i funciona dins el ROS *Master*, per tant, la seva API és accessible utilitzant llibreries normals de XMLRPC.

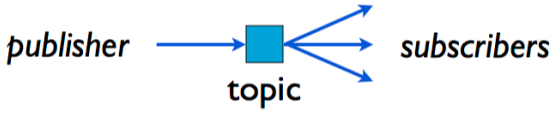

Figura 3.2.2. Esquema comunicació nodes (publicadors/subscriptors a *topics*)

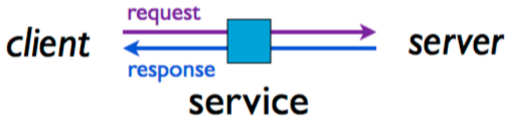

Figura 3.2.3. Esquema funcionament *Service*

4. Desenvolupament Projecte

Un cop adquirits els conceptes previs de ROS ja ens podem centrar en com s'ha desenvolupat el projecte, abans però, revisem la figura 4.1 que mostra la configuració del que es vol dissenyar.

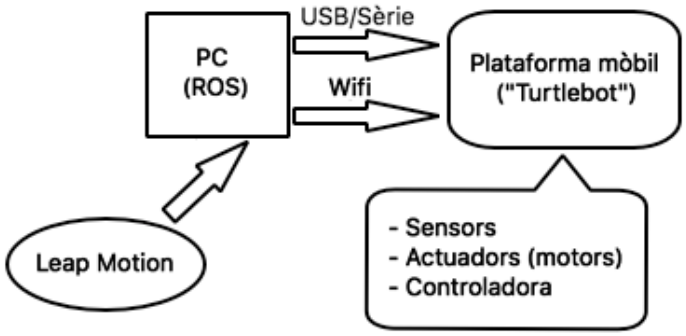

Figura 4.1. Diagrama general del muntatge a realitzar

Com es pot veure aquest sistema està compost per diverses parts, per tal que el lector no es perdi i mantenir tota la informació ordenada s'ha decidit dividir el desenvolupament del projecte en dues parts: *Hardware* (material físic utilitzat) i *Software* (programari desenvolupat i utilitzat).

4.1. Hardware

Així doncs, el maquinari utilitzat en la realització d'aquest projecte està compost per un Leap Motion, un ordinador amb Ubuntu, plataforma mòbil (comunament coneguda com a *Turtlebot*), dos motors Dynamixel AX-12W, 4 sensors d'ultrasons HC-SR04, placa Arduino Mega 2560, bateries de Liti, placa electrònica amb registres desplaçament, placa electrònica comunicació amb motors Dynamixel i placa electrònica distribució de corrent.

- Leap Motion: és un petit dispositiu perifèric amb comunicació USB, dissenyat per estar col·locat sobre una taula o ulleres de realitat virtual. Utilitza dues càmeres monocromàtiques IR i tres LEDs infrarojos, observa una zona semiesfèrica a una distància aproximada d'un metre, per identificar la posició, orientació i moviment de les mans. Els LEDs emeten llum infraroja, mentre les càmeres capturen la reflexió d'aquesta. Les dades capturades per les càmeres s'envien via USB a un ordinador que les processa i genera el model 3D de les mans. En la figura 4.1.1 es pot veure el dispositiu real, en la figura 4.1.2 mostra com està construït.

Figura 4.1.1. Leap Motion controller

En aquest projecte s'utilitza com a dispositiu teleoperador per controlar el moviment de la plataforma mòbil. En funció de la posició i orientació de la mà es farà moure la plataforma endavant, enrere, girar esquerra o girar dreta. El Leap Motion estarà connectat a l'ordinador amb Ubuntu per fer tot el processament de senyals i després enviar la informació cap a la controlodora (Arduino Mega 2560 o Raspberry Pi 2) instal·lada al robot per accionar el motor que correspongui.

Figura 4.1.2. Disseny intern Leap Motion

S'ha escollit aquest sistema com a dispositiu teleoperador i no un joystick o teclat de l'ordinador buscant la innovació en el sistema de control en tipus de robots com aquest i perquè el preu no és molt excessiu.

- Ordinador: en aquest cas s'utilitzarà un ordinador portàtil amb Ubuntu 12.04 (Precise) en cas d'utilitzar un cable USB directe per comunicar-se amb Arduino Mega 2560. Si es fa servir Raspberry Pi 2 amb Ubuntu 14.04 (Trusty), l'ordinador utilitzat portarà instal·lat Ubuntu 14.04 també, pel

tema de compatibilitat de versions de ROS (Ubuntu Precise -> ROS Hydro; Ubuntu Trusty -> ROS Indigo). També es pot fer servir un ordinador de sobretaula, però per comoditat i buscant la portabilitat del sistema s'utilitzarà un portàtil.

- Plataforma mòbil o *Turtlebot*: s'ha buscat una plataforma compacta i econòmica, ja que el que ha de contenir és poca cosa i no massa pes. S'ha comprat el model *Turtle-2WD Mobile Platform* de la marca *DFROBOT*.

Figura 4.1. 3. Plataforma mòbil o *Turtlebot*

- Motor Dynamixel AX-12W: s'ha escollit aquest tipus de motor per les rodes motrius del *Turtlebot* per la seva simplicitat, utilitzen comunicació I2C i incorporen tot el control, encoder, comunicació, etc. La configuració, lectura d'informació, modificació de paràmetres (tipus moviment (per graus o gir complert), velocitat, ID del motor...), l'engegada i l'aturada es realitza a través del bus I2C (*Inter-Integrated Circuit*, utilitzat per enllaçar circuits de baixa velocitat a processadors i microcontroladors en distàncies curtes) i tant ROS com Arduino incorporen les llibreries pertinents per realitzar la comunicació encara més senzilla, per això s'han escollit aquests motors. Es muntaran dos motors, un per fer anar la roda dreta i l'altra per la roda esquerra.

Figura 4.1.4. Motor Dynamixel AX-12W

- Ultrasons HC-SR04: aquests sensors s'utilitzaran per tal de realitzar l'evasió d'obstacles quan es trobi en moviment. S'ha escollit aquest tipus de sensor pel seu preu i perquè Arduino té llibreries per realitzar mesures i cal·libració de sensors, el que simplifica la programació. S'han muntat quatre sensors, un per buscar obstacles per davant, un per l'esquerra, un per la dreta i un per darrera.

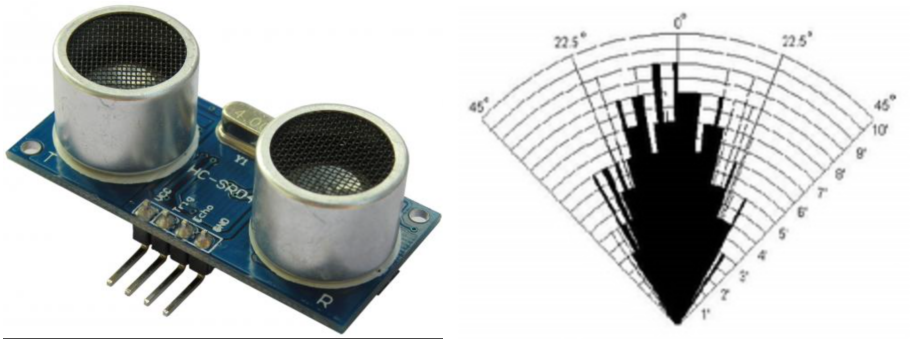

Figura 4.1.5. Sensor ultrasons HC-SR04 i diagrama detecció

- Arduino Mega 2560: controladora per la comunicació entre ordinador, actuadors i sensors. S'ha escollit aquest model pel nombre d'entrades i sortides de que disposa, així es pot col·locar la placa amb LEDs per mostrar de manera visual la proximitat d'algun obstacle.

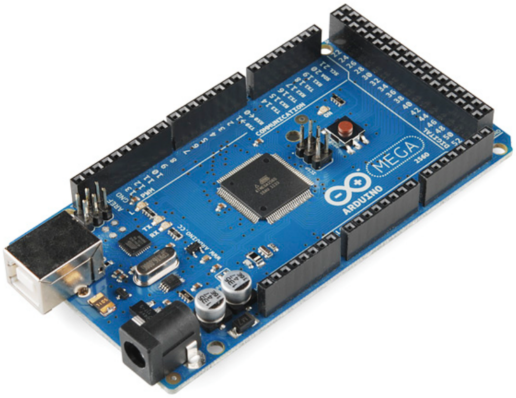

Figura 4.1.6. Arduino Mega 2560

- Bateries Liti: s'ha escollit posar bateries de Liti per la seva durada, facilitat de càrrega i econòmiques. Es connectaran en sèrie vàries bateries per poder aconseguir els 9V que necessiten els motors per ser accionats.

Figura 4.1. 7. Bateria de Liti

- Placa electrònica amb registres desplaçament: s'ha fet una placa personalitzada amb LEDs i registres de desplaçament (74LS194A) per poder indicar a l'usuari de manera visual la proximitat d'obstacles. Conté alimentació de 5V i per cada registre tres pins connectats a l'Arduino, un per la senyal de rellotge (C, per habilitar el desplaçament) i dos per indicar sentit de desplaçament (S0 i S1). Del 74LS194A s'ha connectat el pin MR a 5V per poder habilitar el registre desplaçament, ja que connectat a 0V inhabilita; el pin DSR (*Direction Serial Right*) s'ha connectat a 5V perquè cada vegada que es faci un desplaçament a la dreta activi un LED; el pin DSL (*Direction Serial Left*) s'ha connectat a 0V perquè quan es faci un

desplaçament a l'esquerra s'apagui un LED; VCC és el pin positiu de 5V del circuit integrat; el pin GND són els 0V del xip.

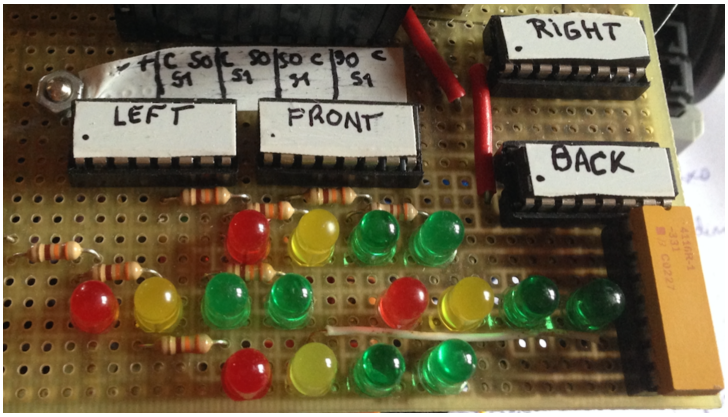

Figura 4.1.8. Placa amb registres desplaçament

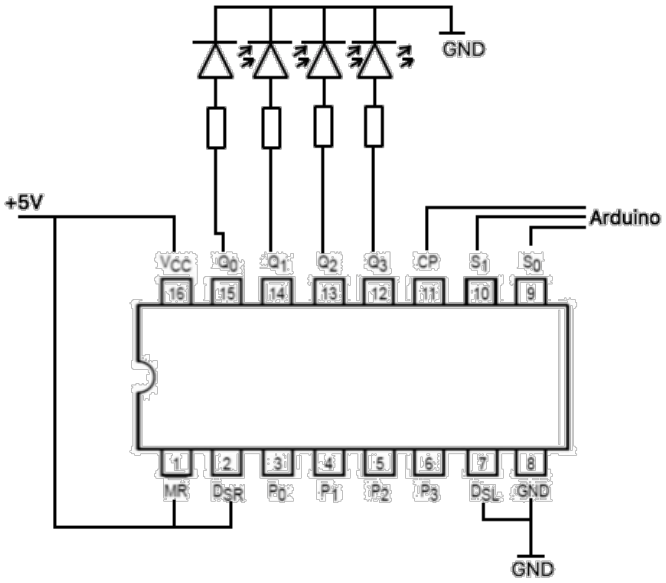

Figura 4.1.9. Esquema d'un registre desplaçament

- Placa comunicació motors: s'ha fet una placa personalitzada per adaptar la comunicació sèrie entre Arduino i motors amb un circuit integrat de buffers triestat (SN74LS241) per commutar entre enviar dades (Tx) i rebre dades(Rx). Això es fa perquè els motors només tenen un fil de comunicació i per diferenciar entre enviament i recepció de dades s'utilitza el circuit integrat, habilitant o deshabilitant el pin de dades es permet la comunicació en un sentit o altre.

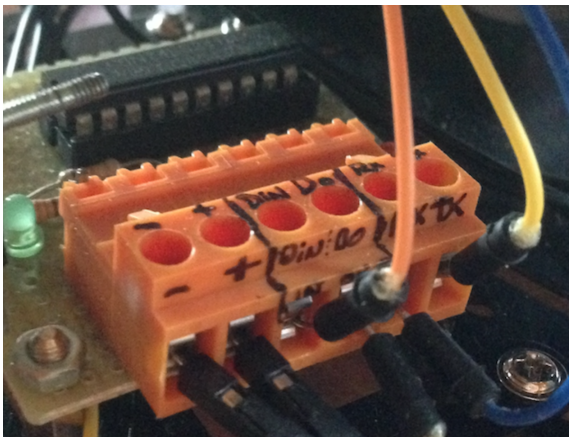

Figura 4.1.10. Placa electrònica adaptació comunicació sèrie

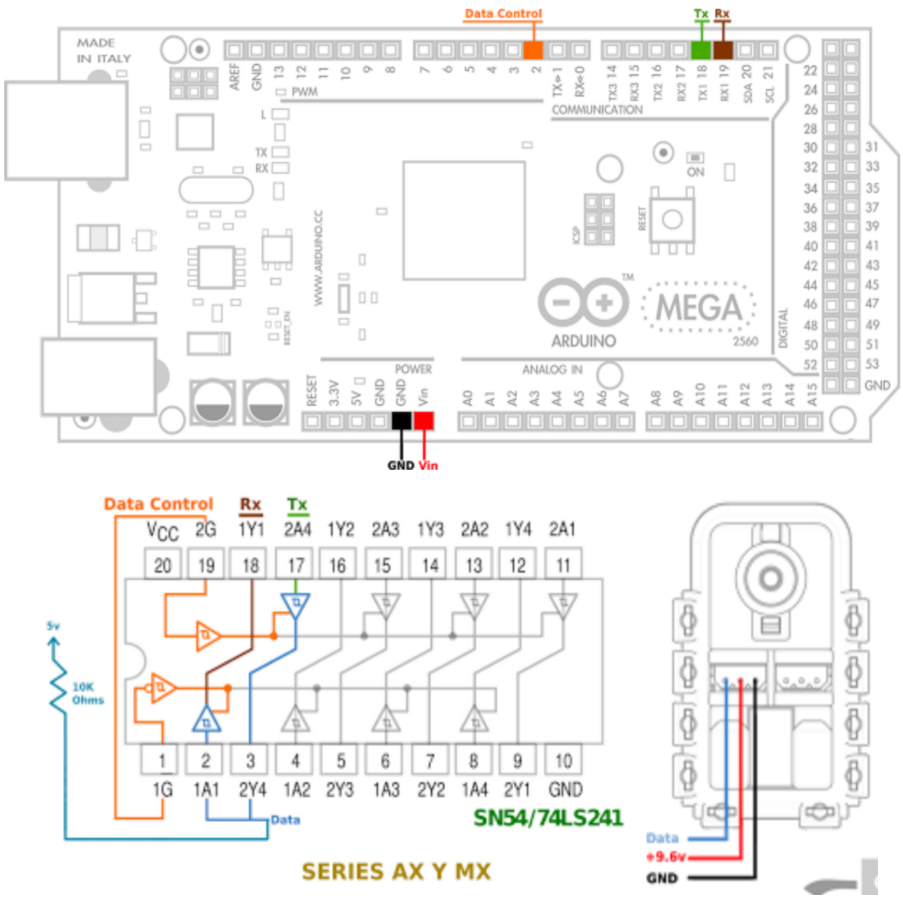

Figura 4.1.11. Esquema connexió placa comunicació Arduino i motors

- Placa electrònica distribució corrent: s'ha fet una placa per poder connectar diferents tensions utilitzades pels diversos elements utilitzats, així tenim la tensió d'entrada (9V) per l'Arduino i els motors, el negatiu o massa (GND) comuna per tots els elements i 5V que dóna l'Arduino per les plaques utilitzades.

Figura 4.1.12. Placa distribució corrent

- Raspberry Pi 2: s'ha decidit utilitzar-la pel fet que s'hi pot instal·lar ROS i connectar-se amb wifi amb l'ordinador que llegeix la informació del Leap Motion, així ens permet alliberar-nos de la limitació que suposa el cable que havíem de tenir connectat a l'Arduino. També s'ha aprofitat per connectar-hi els sensors d'ultrassons, així es pot crear un node per cada un i independitzar el sistema de control, d'aquesta manera l'Arduino s'utilitzarà com a sistema de control dels motors i connectat amb cable a USB a la Raspberry Pi 2.

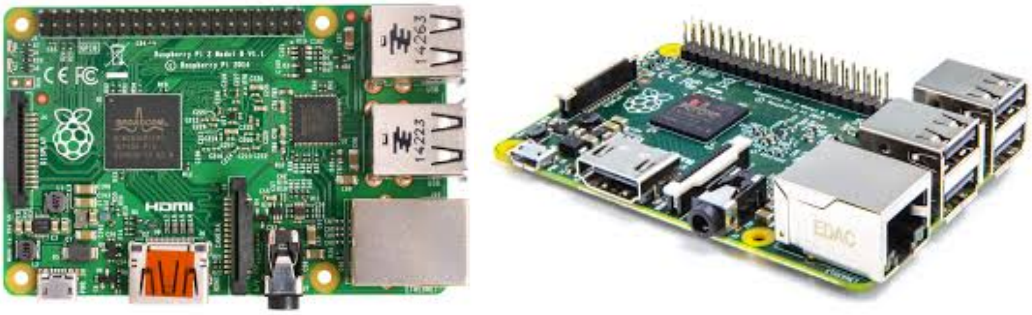

Figura 4.1.13. Raspberry Pi 2

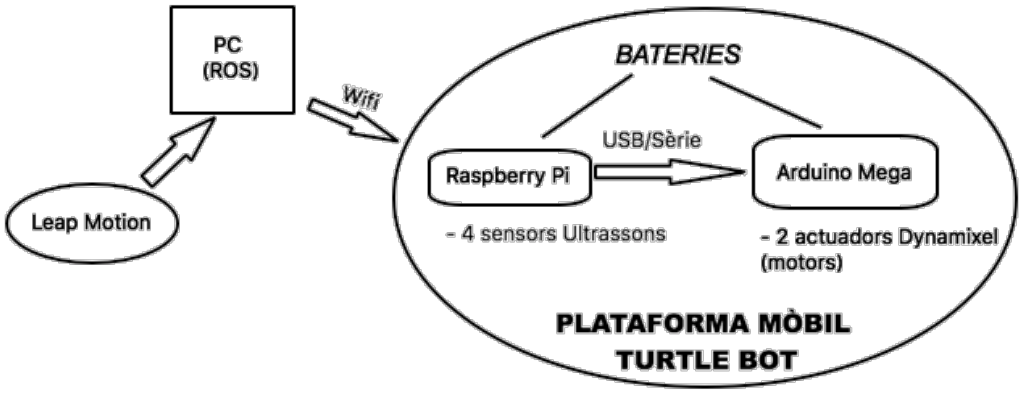

Figura 4.1.14. Esquema distribució hardware utilitzat

4.2. Software

Un cop ja està tot muntat es pot passar a la programació dels diferents nodes, tant a l'ordinador, com a l'Arduino i a la Raspberry Pi 2. Abans, però, cal seguir els passos previs per tal que l'ordinador pugui llegir el Leap Motion, convertir les dades en el protocol ROS i configurar comunicacions entre ordinador i Raspberry Pi 2 i Arduino. Així la configuració final escollida és la següent:

4.2.1. Instal·lació i configuració Leap Motion

Primer de tot cal instal·lar el SDK de Leap Motion per Ubuntu a l'ordinador, per fer-ho es pot anar a la pàgina web *http://www.leapmotion.com/setup/linux*, primer ens hem de donar d'alta com a desenvolupadors, entrar com a usuaris i clicar sobre el botó *"DEVELOPER INSTALLER (DESKTOP/VR)"*, un cop descarregat obrir l'arxiu *README* i seguir els passos que s'indiquen. Per iniciar el controlador obrir un terminal i escriure *sudo leapd*, el primer cop donarà un error, però no passa res, després, en una altra finestra del terminal escriure *LeapControlPanel* a la part de dalt esquerra apareixerà una petita icona de color verd (Leap Motion connectat i reconegut) o negre (Leap Motion no connectat). Si és el primer cop que s'instal·la s'ha de calibrà el sensor, per això clicar dues vegades sobre la icona verda i s'obrirà una finestra, anar a la pestanya *"Troubleshooting"* i clicar sobre *"Recalibrate Device"* a la part dreta s'indiquen els passos a seguir, un cop completat ja es té el Leap Motion preparat per utilitzar. Per comprovar que funcionar correctament, en la mateixa pestanya clicar sobre *"Diagnostic Visualizer"*, s'obrirà una altra finestra i si tot funciona correctament, al passar les mans sobre el dispositiu han d'aparèixer en aquesta finestra.

Seguidament s'han d'instal·lar els paquets de ROS per poder llegir les dades que proporciona el Leap Motion, per fer-ho anar a la pàgina *wiki.ros.org/leap_motion* i seguir els passos que s'indiquen. Un cop configurat ja es disposa del node que proporcionarà tota la informació del Leap Motion (el paquet que conté el node és *leap_motion*, i el node és *sender*.py, que està situat a la ubicació */opt/ros/indigo/lib/leap_motion/sender.py*). Per posar-lo en marxa escriure a una finestra del terminal *rosrun leap_motion sender.py* l'únic que s'ha de fer és crear un node que es subscrigui al *topic* on publica les dades (*/leapmotion/data*) i quedar-se amb les més interessants per la nostra aplicació.

4.2.2. Configuració comunicació Raspberry Pi i Arduino

Per tal de poder establir comunicació entre la Raspberry Pi i Arduino utilitzant els protocols de ROS s'ha d'instal·lar el paquet *rosserial* i *rosserial arduino*, que no venen amb la instal·lació de ROS i permet establir comunicació sèrie a través del port USB de la Raspberry Pi, per fer-ho visitar la pàgina *http://wiki.ros.org/rosserial_arduino/Tutorials/* i seguir el primer tutorial (*Arduino IDE Setup*). Un cop instal·lat, per establir la comunicació s'ha d'executar el node *serial_node.py* que es troba dins el paquet *rosserial_python* especificant el port USB que està connectat l'Arduino, en el cas d'aquest projecte es troba connectat a */dev/ttyACM0*, així doncs, la comanda complerta a introduir al terminal per establir comunicació és: *rosrun rosserial_python serial_node.py /dev/ttyACM0*. Si és el primer cop donarà un error, ja que es necessiten permisos d'administrador per accedir al port USB, per això abans s'ha d'executar la següent comanda en un terminal: *sudo usermod -a -G dialout* ***user_name***, en la realització del projecte, el nom d'usuari al configurar l'ubuntu a la Raspberry Pi es va escollir *ubuntu*, per tant, el que s'hauria d'escruire és: *sudo usermod -a -G dialout ubuntu*, ara ja es podrà accedir al port USB amb ROS.

4.2.3. Configuració Arduino IDE

En el pas anterior que s'ha instal·lat el paquet *rosserial arduino* també s'explica com instal·lar la llibreria *ros_lib* a l'Arduino IDE perquè aquest pugui compilar els programes ROS i poder-los descarregar a la placa. Pel desenvolupament del projecte també és necessari instal·lar una altra llibreria a l'Arduino IDE, la que permet la comunicació i control dels motors Dynamixel, per fer-ho visitar la

pàgina *savageelectronics.blogspot.com.es/2011/01/arduino-y-dynamixel-ax-12.html* on a part d'explicar com utilitzar la llibreria també hi figura el disseny de la placa electrònica per la correcte comunicació entre Arduino i motors, ja que l'Arduino disposa de comunicació Full-duplex (pot enviar i rebre dades alhora) i en canvi, els motors Dynamixel són Half-duplex (només poden rebre o enviar dades en cada moment, per això només disposen d'un fil de comunicació).

4.2.4. Configuració comunicació PC i Raspberry Pi

Per tal de poder controlar la plataforma mòbil de manera remota (sense conexió física entre la plataforma i l'ordinador) s'ha escollit fer-ho amb wifi. Tant l'ordinador com la Raspberry Pi porten instal·lat el sistema operatiu Ubuntu 14.04 (versió *Trusty*) que amb la instal·lació bàsica no porten instal·lat cap sistema de comunicació remot entre diferents màquines, per la seva senzillesa i facilitat d'utilització s'ha escollit la connexió *ssh*, per tant, tant a l'ordinador com a la Raspberry Pi s'ha d'instal·lar, per fer-ho visitar la pàgina *https://help.ubuntu.com/lts/serverguide/openssh-server.html* seguir els passos a ambdues màquines fins a l'apartat *SSH Keys*, el que s'explica en aquest apartat només executar-ho en la Raspberry Pi i els arxius que es generen copiar-los a la carpeta *Home* de l'ordinador, ja que contenen les claus per poder-se connectar a la Raspberry Pi.

5. Resultats

Com s'ha pogut intuir en els diferents apartats anteriors el plantejament inicial i que funcionava era utilitzar un ordinador amb Ubuntu per connectar-hi el Leap Motion, fer-hi córrer tots els nodes necessaris pel control i connectat amb un cable força llarg l'Arduino via USB per enviar la informació necessària per fer moure els motors. Un cop vaig comprovar que aquesta configuració funcionava se'm va plantejar el dubte i/o l'interès de poder comunicar-me amb l'Arduino a través d'una xarxa sense fils, així la limitació del cable desapareixeria i la plataforma guanyaria llibertat. Després de fer vàries recerques vaig pensar que la millor opció, i la més econòmica, era utilitzar una xarxa wifi com a medi de transport de dades, però com que de moment el sistema ROS no permet o no ha desenvolupat la comunicació wifi per Arduino vaig decidir instal·lar-hi una Raspberry Pi. Després de vàries proves fallides amb la Raspberry Pi vaig provar la Raspberry Pi 2, que al tenir més prestacions vaig pensar que aniria millor. Al cap de 3 o 4 setmanes de proves i testos finalment s'ha aconseguit que tot funcioni de manera correcte. Així per situar al lector de quina ha sigut la configuració final del projecte veure la figura 5.1.

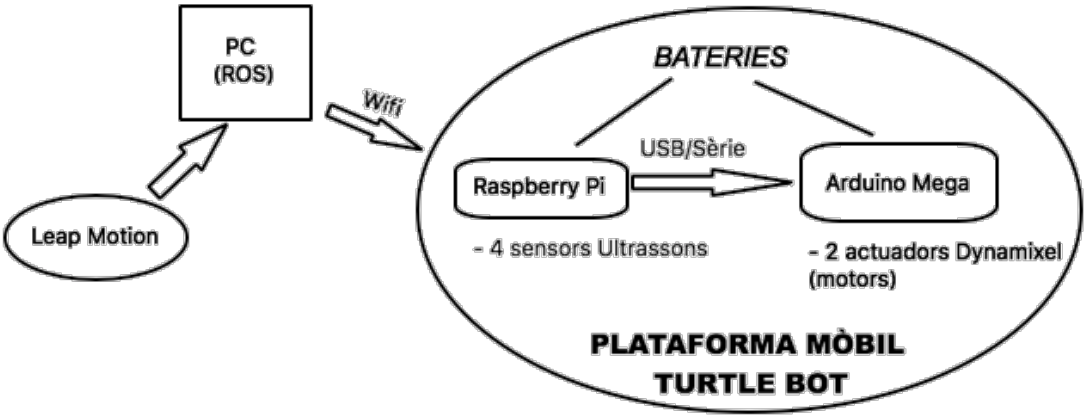

Figura 5.1. Esquema sistema final dissenyat

La figura 5.2 pretén que el lector pugui veure quina ha sigut la configuració final del hardware en la plataforma en l'elaboració d'aquest projecte.

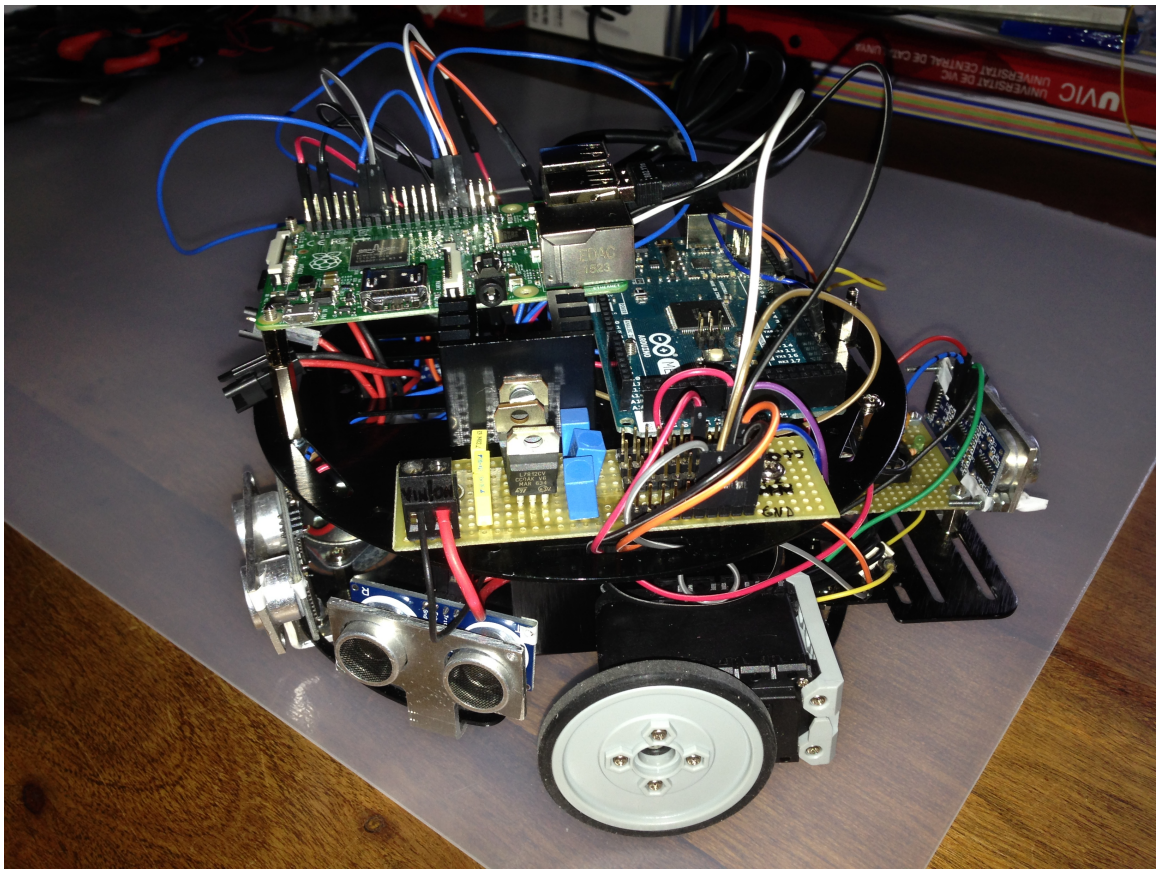

Figura 5.2. Construcció de la plataforma mòbil o "Turtle Bot"

S'ha decidit connectar els sensors a la Raspberry Pi 2 per deixar l'Arduino com a controlador dels motors i per poder il·lustrar la diversificació que permet ROS, és a dir, que en una mateixa màquina hi puguin està corrent diferents nodes, tant publicadors (*publishers*), subscriptors (*subscribers*) com publicadors i subscriptors alhora. En la figura 5.3 es pot veure tots els nodes o programes que estant corrent en tot el sistema, així com els *topics* a través dels quals es comuniquen.

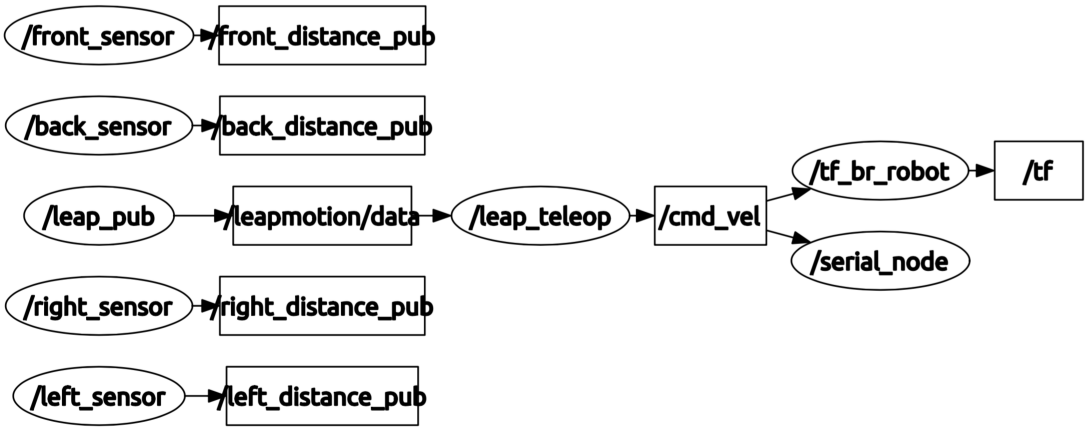

Figura 5.3. Nodes i *topics* del sistema construit

El que està dins les el·lipses representen els nodes, i el que està dins de rectangles representen els *topics* als quals publiquen o llegeixen els missatges. Com s'ha mencionat a l'apartat 3.2 aquí es pot veure que un node pot ser publicador, subscriptor o ambdós alhora i que un *topic* pot tenir varis subscriptors. Per veure en detall com s'han programat els diferents nodes i el diagrama de flux que representen veure l'Annex 1. Els nodes dins la Raspberry Pi 2 han estat programats en Python, ja que és un llenguatge nou per mi i en volia aprendre i també per la facilitat d'importar i d'utilitzar les llibreries per accedir als pins d'entrada i sortida de la placa (coneguts com a *GPIO* (*General Purpose Input Output*)). Mentre que el de l'Arduino ha estat programat en C++, que és el llenguatge que s'utilitza per desenvolupar els programes. D'aquesta manera també es pot comprovar com a dins d'un mateix sistema, amb ROS poden coexistir programes o nodes desenvolupats en diferents llenguatges sense complicacions, com també s'ha esmentat en l'apartat 3.2.

Com s'explica de manera detallada en l'Annex 1, un dels nodes (*tf_br_robot.py*) permet utilitzar l'eina visual *RVIZ* que ofereix ROS, d'aquesta manera es pot veure el robot (si es disposa de model 3D, que per aquest projecte no s'ha realitzat i no es pot mostrar), els diferents *frames* o eixos de coordenades definits en el robot (en aquest projecte s'ha definit el de referència, anomenat *map*, el centre de baix de la plataforma del robot *tf_br*, i els quatre sensors d'ultrassons *front_us_sensor, back_us_sensor, right_us_sensor* i *left_us_sensor* respecte del *tf_br*) i la informació de cada sensor. En la figura 5.4 es mostra una captura de pantalla on es pot veure tota aquesta informació.

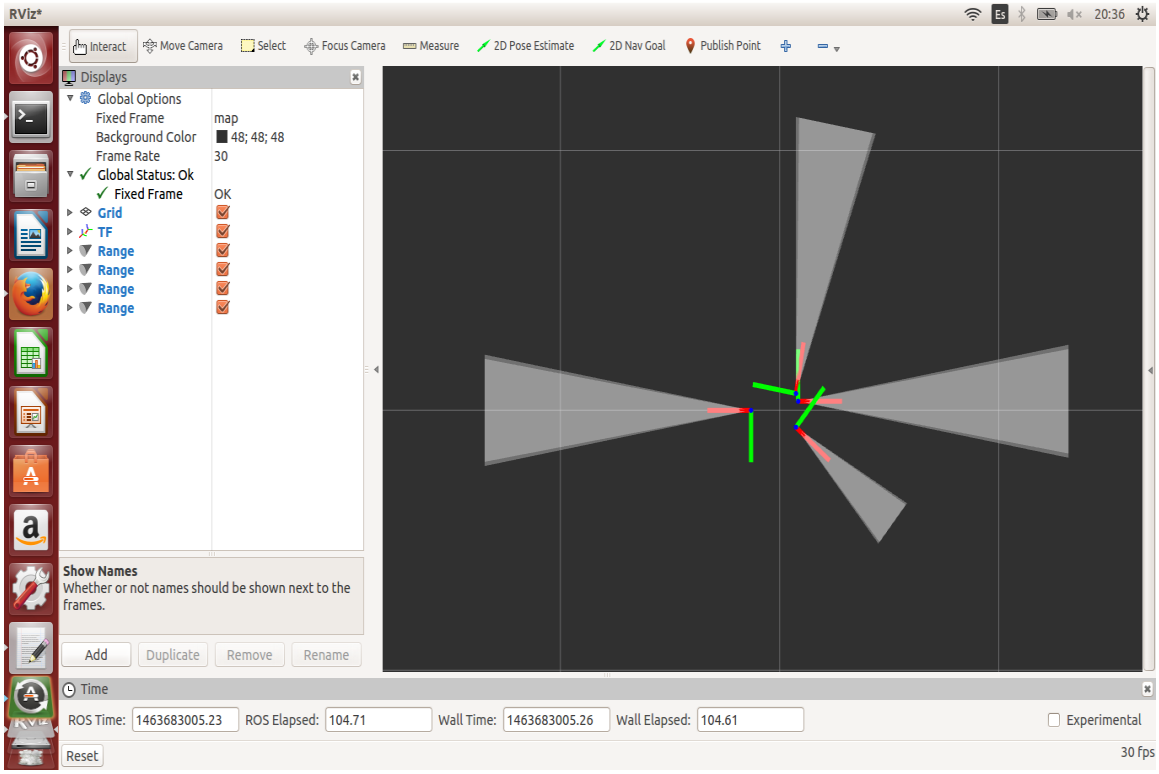

Figura 5.4. Captura pantalla amb *RVIZ* mostrant eixos coordenades i lectura sensors

En la figura la plataforma del robot està girada 90º cap a la dreta, així el davant queda apuntant a la dreta i el darrera a l'esquerra, el sensor de l'esquerra és a la part de dalt i el de la dreta a la part de baix. El con que es mostra representa la distància que està mesurant cada sensor, cada quadrat de la imatge té una superfície d'1m^2, per poder tenir una idea visual del que mostra, cada costat dels quadrats fa 1m de longitud.

L'*RVIZ* és una eina molt útil en ROS, ja que a part de poder veure el robot i la seva posició i orientació (tal com es mostra en la figura 5.4) també permet veure la informació que proporcionen els diferents sensors que pugui tenir instal·lats el robot, en el cas d'aquest projecte només hi ha els sensors d'ultrassons, que com que els nodes publiquen la distància segons el tipus de missatge *sensor_msgs/Range*, en la pantalla s'activa la informació de tipus *Range*. Per poder incorporà més eines visuals en *RVIZ*, i així poder tenir més informació (núvols de punts per càmeres de profunditat, mapes, imatges ...) cal seguir els passos següents:

- Prémer a la part inferior esquerra el botó que diu *Add*
- S'obre una finestra amb dues pestanyes, com es mostra a la figura 5.5, la

primera *By display type* per afegir visualització segons el tipus d'informació que es vulgui veure; la segona *By topic* permet seleccionar un dels *topics* que s'estan publicant i automàticament tria el tipus que li correspon

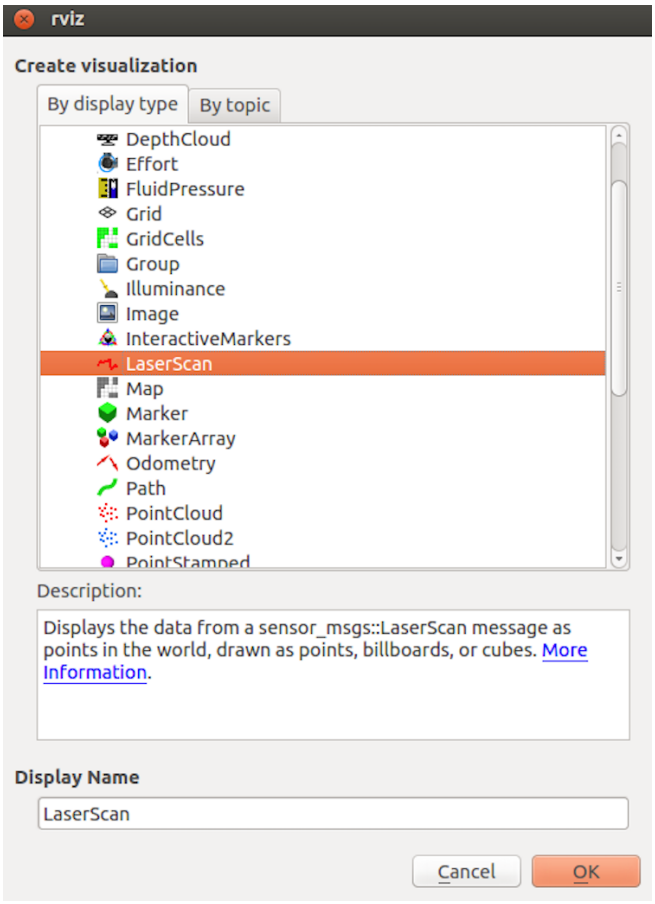

Figura 5.5. Captura imatge *RVIZ* finestra afegir visualització

- En el cas d'aquest projecte es va seleccionar la pestanya *By topic* i un per un es van afegir les dades dels quatre nodes que publiquen la informació dels sensors
- Per poder veure els eixos de coordenades, de la pestanya *By display type* seleccionar el tipus *TF*

Pel que fa a l'execució o posada en marxa de tots els nodes del sistema es realitza a través de l'ordinador, per això per executar tots els nodes de la Raspberry Pi 2 s'ha creat un fitxer *.launch* que mitjançant la comanda en ROS *roslaunch* permet executar varis nodes de manera simultània, per veure aquest fitxer consultar l'Annex 1, aquest fitxer posa en marxa tots els nodes excepte el que habilita la comunicació amb l'Arduino i posa en marxa els motors. S'ha

decidit així per qüestions de seguretat, ja que es prefereix que fins que tot el sistema estigui funcionant correctament no es posarà en marxa cap motor.
Per tant, per posar en funcionament la plataforma a l'ordinador s'ha d'obrir el terminal, primer de tot s'ha d'assegurar que hi ha comunicació entre ordinador i Raspberry Pi, per això escriure la comanda *ping ubuntu@ubuntu*, quan es vegi que la comunicació és correcte prémer *Ctrl+C* per aturar. Ara que estem segurs que hi ha comunicació ens hem de connectar a la Raspberry Pi, per això escriure *ssh ubuntu@ubuntu*, un cop connectats, com que es necessita accedir als pins *GPIO* s'ha de fer amb permisos d'administrador, però com que ROS no accepta la instrucció *sudo* ens hem de connectar com a administradors, per tant, s'ha d'escriure *sudo -s*, ara ja estem preparats per executar el fitxer *launch* esmentat anteriorment *roslaunch testing_leap testing_leap.launch*, una de les peculiaritats de la comanda *roslaunch* és que no necessita que estigui funcionant el *roscore* ja que si no està en marxa ell mateix l'executa abans d'arrencar els nodes. El següent pas és posar en marxa el Leap Motion, per fer-ho obrir un altre terminal i escriure *LeapControlPanel* per posar en marxa els drivers, un cop està en funcionament s'ha d'executar el node que envia la informació, per això, en un altre terminal escriure *rosrun leap_motion sender.py*. Finalment, només queda posar en marxa la comunicació entre Raspberry Pi i Arduino, per fer-ho, en un altre terminal ens hem de connectar amb *ssh* a la Raspberry Pi (com s'ha explicat unes línies més amunt), en aquest cas no es necessiten permisos d'administrador, per tant, un cop connectats via *ssh* escriure *rosrun rosserial_python serial_node.py* i ja tenim tot el sistema en funcionament. Per veure-ho més clarament seguidament es detalla en forma de punts com s'executa el sistema. Com que s'accedeix de forma remota a la Raspberry Pi 2, totes les comandes s'escriuen en finestres del terminal de l'ordinador:

- ***ping ubuntu@ubuntu***

  Un cop es vegi que hi ha comunicació entre les dues màquines prémer ***Ctrl+C*** per aturar la comanda
- ***ssh ubuntu@ubuntu***

  És per connectar-se a la Raspberry, ens demanarà la contrassenya i ja estarem connectats
- ***sudo -s***

Activar permisos de súper usuari per poder fer funcionar els nodes que han d'accedir als pins d'entrad/sortida (sensors)

- ***roslaunch testing_leap testing_leap.launch***

  Executa tots els nodes que han de córrer a la Raspberry
- Obrir un altre terminal i escriure: ***LeapControlPanel***

  Posa en marxa el driver del Leap Motion
- Obrir un altre terminal i escriure: ***rosrun leap_motion sender.py***

  Executa el node que llegeix informació del Leap Motion i les publica en format ROS
- Obrir un altre terminal i escriure: ***ssh ubuntu@ubuntu***

  Connectar-se a la Raspberry per poder engegar el node de comunicació amb l'Arduino
- ***rosrun rosserial_python serial_node.py***

  Posa en marxa el node de comunicació entre la Raspberry i l'Arduino
- Obrir un altre terminal i escriure: ***rosrun rviz rviz***

  Per posar en marxa l'eina de visualització RVIZ

Abans d'aconseguir el correcte funcionament d'aquest projecte s'ha passat per diferents fases. Una de les primeres proves que es van fer va ser posar en marxa el Leap Motion a l'ordinador amb Ubuntu i amb l'eina de visualització que porten els drivers veure que realment estava funcionant bé (com es pot veure a la figura 5.6), que va costar molt d'aconseguir perquè amb el primer ordinador que treballava era molt antic i hi havia problemes de compatibilitat.

Un cop es van solucionar els problemes del Leap Motion el següent pas va ser instal·lar el paquet de ROS que permet enviar les dades del Leap Motion en missatges ROS, per així poder analitzar tota la informació que dóna i discriminar quines dades eren les més útils pel projecte.

Ara ja es tenia tota la part d'ordinador testejada i en funcionament, ja podia començar a treballar en la part d'Arduino, ja que en un principi no s'utilitzava la Raspberry. Primer de tot vaig instal·lar el paquet de ROS que permet la comunicació via USB amb l'Arduino, i les primeres proves van ser llegir senyals de polsadors connectats a l'Arduino i publicar el seu estat en un *topic* per després amb l'ordinador comprovar que el podia llegir. Una altra prova va ser

crear un node a l'ordinador que publiqués un missatge per després ser llegit amb l'Arduino i en funció del contingut d'aquest missatge encendre o apagar un LED. Amb aquestes dues proves vaig assegurar que la comunicació ordinador-Arduino i Arduino-ordinador funcionava de manera correcte.

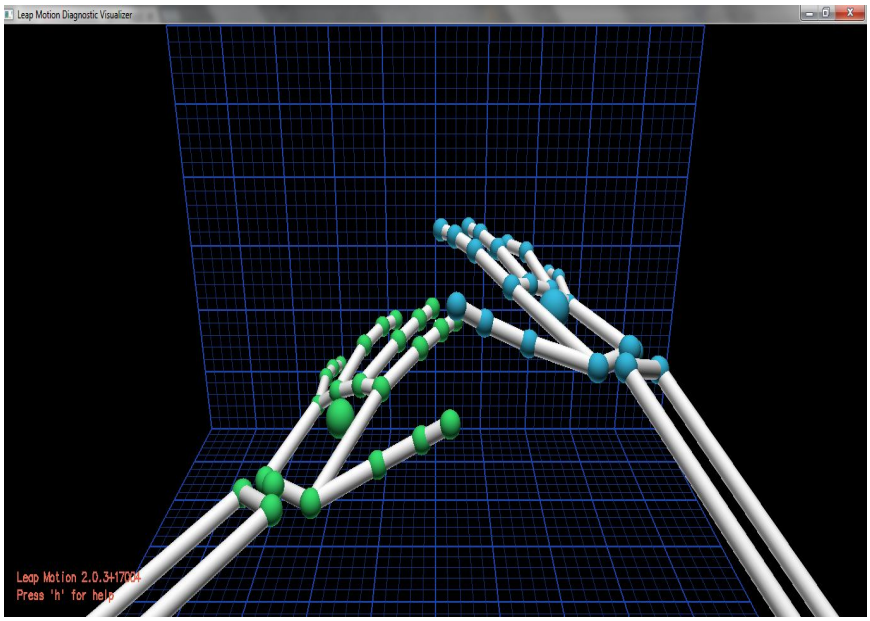

Figura 5.6. Leap Motion visualizer mostrant les mans llegides

Ara ja tenia tot el sistema amb la certesa que funcionava i només quedava programar els nodes necessaris pel correcte funcionament. Per això vaig crear un node que s'executava des de l'ordinador el qual llegia la informació del Leap Motion, processava les dades necessàries i enviava a l'Arduino els missatges pel control de moviment de la plataforma a través del cable USB. Per altra banda, a l'Arduino vaig fer el programa per fer funcionar els motors segons el missatge rebut per l'ordinador i alhora llegir la informació que proporcionaven els sensors ultrassons.

Així el sistema funcionava i havia complert la major part dels objectius, però tenia una limitació molt gran, la plataforma només podia funcionar si estava connectada amb un cable USB a un ordinador. Per això vaig pensar que amb la Raspberry Pi amb una connexió wifi podia solucionar aquest problema. Aquí és on vaig tenir més problemes, primer perquè amb la Raspberry Pi no vaig ser capaç d'instal·lar-hi ROS, fins a l'aparició de la Raspberry Pi 2 no ho vaig aconseguir. Un cop ja tenia la Raspberry Pi 2 amb ROS funcionant van aparèixer nous problemes, segurament pel fet que no sóc expert en informàtica ni en Linux

(no hi havia treballat mai), poder establir connexió de forma remota a través de l'ordinador amb la Raspberry.

Un cop vaig aconseguir que la Raspberry i l'ordinador es pugessin entendre vaig decidir que l'ordinador només havia de servir per llegir la informació del Leap Motion i publicar-la a través del paquet de ROS que havia instal·lat, deixant tots els nodes de control corrent a la Raspberry. Com que connectar els sensors i realitzar el programa amb la Raspberry era senzill vaig decidir treure'ls de l'Arduino, amb això vaig aconseguir que cada sensor treballés amb un node independent i que no depengués un de l'altre. El problema va venir amb els motors, com que ja disposava de poc temps per dedicar-hi i veient la complexitat de fer-los funcionar amb la Raspberry vaig decidir mantenir l'Arduino com a controladora dels motors (que ja funcionava) i connectar l'Arduino a la Raspberry amb un cable USB per poder-se comunicar i enviar les comandes dels motors.

6. Conclusions

Un cop realitzat el projecte, comprovat el seu funcionament i analitzat tot el que es pretenia aconseguir puc dir que estic molt satisfet dels resultats obtinguts.

- Introducció al sistema ROS (Robotic Operating System)

S'ha intentat realitzar una descripció clara i entenedora, sense entrar molt en detall per no carregar massa al lector, ja que es pretenia aconseguir poder entendre el que significa sense ser molt tècnic per facilitar-ne la comprensió.

- Instal·lació i funcionament de les eines ROS

En aquest aspecte tampoc s'ha volgut entrar molt en detall, i com que la wiki de ROS està molt ben explicat i detallat s'han facilitat els enllaços perquè el lector els pugui seguir en cas de voler instal·lar i provar el sistema, d'aquesta manera s'ha evitat sobrecarregar d'informació el treball.

- Construcció i programació plataforma mòbil

La construcció de la plataforma ha sigut relativament senzilla, ja que des d'un bon principi s'havia pensat en una plataforma del tipus *"turtlebot"* i al mercat hi ha molts models, al final s'ha optat per la més econòmica que es va trobar. Pel que fa a la programació, en un principi s'havia desenvolupat tot el sistema sense la Raspberry Pi i es connectava l'Arduino directament a l'ordinador amb un cable USB. Tot funcionava de manera correcte, però només hi havia un problema, la limitació d'haver d'estar connectat amb cable a la plataforma mòbil, el que la privava de llibertat de moviment i la complexitat d'haver-la de seguir si la longitud del cable era massa curta.

- Controlar plataforma utilitzant Leap Motion

Es va decidir utilitzar aquest dispositiu per controlar la plataforma per donar-li el toc innovador al projecte, ja que després de conèixer la seva existència em va fascinar la possibilitat de treballar amb un dispositiu d'aquestes característiques, el seu preu assequible també va facilitar la decisió. Els únics problemes que em va originar a l'inici van ser causa de compatibilitat dels drivers amb un ordinador massa antic, un cop solventat aquest aspecte no van aparèixer més problemes.

- Buscar sensors més adients per evasió d'obstacles

En un principi havia pensat en utilitzar sensor làser o càmeres de profunditat, però el seu preu era molt elevat i la programació complicada, per això

finalment vaig optar per utilitzar sensors d'ultrassons, ja que són econòmics i la programació és senzilla.

- Comunicació Wifi entre ordinador i plataforma mòbil

Per tal de poder-ho aconseguir es va decidir intentar integrar-hi la Raspberry Pi per aconseguir la comunicació sense fils i no tenir limitacions de moviment, mentre la plataforma i l'ordinador tinguin la suficient cobertura en la xarxa que estant connectats. Encara que sembli una qüestió trivial el fet d'acoblar la Raspberry quan el sistema ja funcionava, la veritat és que ha sigut més complex del que m'esperava i fins i tot en algun moment frustrant. El primer problema va ser trobar un sistema operatiu que pogués funcionar a la Raspberry i que permetés la instal·lació de ROS. Les primeres proves es fan fer amb el sistema operatiu per defecte anomenat *Raspbian*, però la instal·lació de ROS no va ser possible. La sort va ser que amb l'aparició de la Raspberry Pi 2 s'hi podia instal·lar Ubuntu Trusty (14.04) amb el qual si que era fàcil la instal·lació de ROS.

Un cop ja es tenien les dues màquines amb la mateixa versió d'Ubuntu i de ROS funcionant el problema va ser poder establir comunicació entre ambdós. Després de fer vàries cerques vaig deduir que la millor manera era utilitzar l'eina *ssh*, però el fet de no haver-la utilitzat mai va ser molt complex poder-ho configurar de manera correcte, però després de moltes hores invertides es va poder solucionar.

Un altre dels problemes que em vaig trobar a l'utilitzar la Raspberry Pi és que en un principi volia reutilitzar els nodes que havia escrit quan tenia l'Arduino connectat directament a l'ordinador, l'inconvenient era que estaven escrits amb llenguatge C++ i la Raspberry l'havia de programar en Python, ja que les llibreries que havia trobat per poder accedir als pins *GPIO* estaven escrites amb Python, per tant això va fer que hagués de tornar a reescriure els codis dels nodes. Al ser un llenguatge nou per a mi (Python) també em va suposar perdre o dedicar-hi més hores del compte, ja que primer havia d'aprendre l'estructuració i l'estil de programació. Però finalment ho vaig aconseguir i el repte més gran, que era poder controlar la plataforma mòbil sense fils ho vaig aconseguir.

- Programar sistema SLAM (Simultaneous Localization And Mapping)

Pel que fa als objectius plantejats es podria dir que els he complert tots excepte el de programar un sistema *SLAM* (*Simultaneous Localization And Mapping*). Sobretot per la falta de temps, ja que hauria de començar des de zero i entendre com funciona i què es necessita per tal de desenvolupar un sistema d'aquestes característiques, i després de dedicar tant temps a configurar el sistema de comunicació sense fils no disposava del temps necessari per poder entrar en matèria i com que era un objectiu secundari al final he decidit no portar-lo a la pràctica.

7. Futures aplicacions

Aquest projecte no està pensat per quedar-se en un simple treball final de grau i endreçar-lo a un armari, sinó que des d'un bon principi volia que em servís com a base per entrar al món de ROS, adquirir els coneixements bàsics per dissenyar un sistema de robòtica mòbil i explorar totes les possibilitats que ofereix i en un futur poder-ho aplicar en projectes personals. Després d'haver elaborat aquest projecte estic molt satisfet perquè tot el que he esmentat s'ha complert.

El proper objectiu és poder-hi instal·lar una càmera de profunditat, amb RGB (color), una IR (infrarojos) i micròfon (tipus Kinect, Asus Xtion Pro Live o la recent marca Orbbec) a la mateixa plataforma mòbil construïda per aquest projecte, per tal de complir l'objectiu no finalitzat de dissenyar un sistema *SLAM* i simulació de làser 2D amb la càmera de profunditat per fer el mapa de zones i navegació autònoma de la plataforma.

Com a objectiu final que em vaig marcar a l'hora de decidir desenvolupar aquest projecte va ser poder aplicar tots aquests coneixements en el disseny i construcció d'un

a cadira de rodes que pugui funcionar de manera autònoma, planificant la ruta a seguir, evasió d'obstacles i fins i tot incorporar una pantalla tàctil que mostri el mapa de la zona on es troba i poder seleccionar el punt on es vol anar i que l'usuari no se n'hagi de preocupar del control. Aquest és un projecte personal a llarg termini, però estic molt il·lusionat i amb moltes ganes de poder-lo dur a terme.

8. Pressupost

A l'hora d'elaborar el pressupost del projecte només s'ha tingut en compte la part del hardware. Les hores invertides en desenvolupar el software i a muntar la plataforma no s'hi han comptat, han sigut moltes i en molts dies, ja que el fet de compaginar la feina amb els estudis hi he anat dedicant el temps que he pogut a les estones lliures i no n'he portat un control. En la figura 8.1 es pot veure de manera detallada el que ha costat cada element utilitzat, la quantitat usada de cada un, el cost total per element i al final hi ha la suma total de tots els components utilitzats.

Descripció	€/unitat	Quantitat	Total
Leap Motion	39,73€	1	39,73€
Plataforma mòbil	32€	1	32€
Motor	40€	2	80€
Sensor ultrassons	1,04€	4	4,16€
Arduino Mega	42,98€	1	42,98€
Raspberry Pi 2	31,98€	1	31,98€
Bateries	5,59€	4	22,36€
Placa Arduino Motors	4€	1	4€
Placa Distribució Corrent	2€	1	2€
		TOTAL	**258,21€**

Figura 8.1. Pressupost del material utilitzat

9. Annex 1. Programes dels nodes

9.1. Node *leap_teleop.py*

Aquest node és l'encarregat de llegir les dades que proporciona el node del Leap Motion i a partir de les dades més rellevants decidir si fa anar la plataforma mòbil endavant, enrere, dreta, esquerra o atura. Per saber la direcció (endavant/enrere) de totes les dades que proporciona el Leap Motion, després de vàries proves, s'ha decidit agafar la component Z de la normal, que dóna valors entre -1 i 1, segons la inclinació de la mà (avall o amunt), en el programa es multiplica per 100 per tenir més resolució. Per diferenciar el sentit de gir, s'ha escollit la component X de la normal, que també dóna valors entre -1 i 1 (segons si es gira la mà cap a la dreta o esquerra) i també es multiplica el valor proporcionat per 100. Tots els valors entre -30 i 30 es considera que la plataforma s'ha de parar o si es rep el mateix valor de forma consecutiva (desapareix la mà de sobre el controlador) també es pararà. Un cop gestionat el moviment es publica un missatge de tipus *geometry_msgs/Twist* modificant la component X de la variable *linear*, pel moviment rectilini (*linear.x = 1*-> endavant; *linear.x = -1*-> enrere; *linear.x = 0* -> parat) i la component Z de la variable *angular*, pel sentit de gir (*angular.z = 1* -> gir a l'esquerra; *angular.z = -1* -> gir a la dreta; *angular.z = 0* -> no giris).

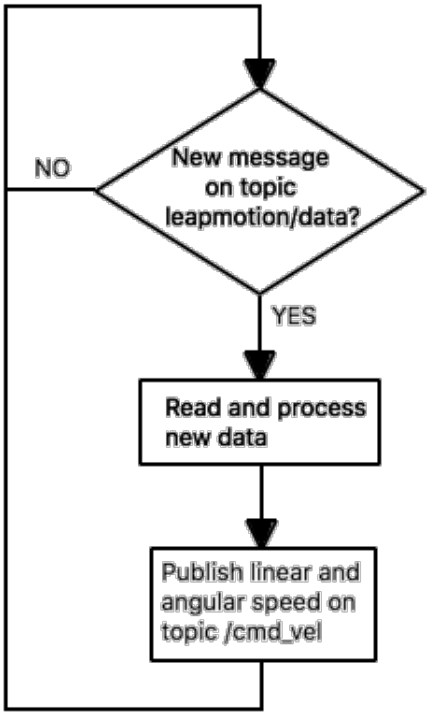

Figura 9.1.1. Diagrama de flux del node *leap_teleop.py*

```
#!/usr/bin/env python
__author__ = 'guiu'

import rospy
from leap_motion.msg import leap
from leap_motion.msg import leapros
from geometry_msgs.msg import Twist

class leap_teleop():
        def __init__(self):
            rospy.init_node('move_test', anonymous=False)

            rospy.on_shutdown(self.shutdown)

            self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=20)

            rospy.Subscriber("leapmotion/data", leapros, self.callback_leap)
            self.twist = Twist()
            rate = rospy.Rate(10)
            self.old_x = 0.0
            self.old_z = 0.0
            self.dirty = False

            while not rospy.is_shutdown():
```

```
                if(self.dirty):
                        self.cmd_vel_pub.publish(self.twist)
                        rospy.loginfo('\nPublish twist.linear.x = %f' % self.twist.linear.x +
                        '\ntwist.angualr.z = %f' % self.twist.angular.z)
                        self.dirty = False
                rate.sleep()

    def callback_leap(self, data):
        linear_ = 0
        angular_ = 0
        diff_x = data.normal.x-self.old_x
        diff_z = data.normal.z-self.old_z
        if((data.normal.z*100)>30 and diff_z!=0.0):
                #** MOVE FORWARD **#
                linear_ = 1.0
                self.dirty = True
                self.old_z = data.normal.z
                self.old_x = data.normal.x
        elif((data.normal.z*100)<-30 and diff_z!=0.0):
                #** MOVE BACKWARD **#
                linear_ = -1.0
                self.dirty = True
                self.old_z = data.normal.z
                self.old_x = data.normal.x
        elif((data.normal.x*100)>30 and diff_x!=0.0):
                #** TURN LEFT **#
                angular_ = 1.0
                self.dirty = True
                self.old_x = data.normal.x
                self.old_z = data.normal.z
        elif((data.normal.x*100)<-30 and diff_x!=0.0):
                #** TURN RIGHT **#
                angular_ = -1.0
                self.dirty = True
                self.old_x = data.normal.x
                self.old_z = data.normal.z
        else:
                #** STOOOOOP **#
                linear_ = 0.0
                angular_ = 0.0
                self.dirty =  True
                self.old_x = data.normal.x
                self.old_z = data.normal.z
        self.twist.linear.x = linear_
        self.twist.angular.z = angular_

def shutdown(self):
    self.cmd_vel_pub.publish(Twist())
    rospy.sleep(1)
```

```
if __name__ == '__main__':
        try:
          leap_teleop()
          rospy.spin()
        except rospy.ROSInterruptException:
                rospy.loginfo('Move Base Test finished...')
```

9.2. Node *tf_br_robot.py*

La funció d'aquest node és la d'enviar la posició de la base respecte el (0, 0, 0) i la orientació (0, 0, 0, 1), en quaternions, del mapa. Així com la posició dels sensors respecte la base, així es pot utilitzar l'eina visual de ros *RVIZ* i veure la informació que proporcionen els sensors en temps real i la posició i orientació de la plataforma mòbil.

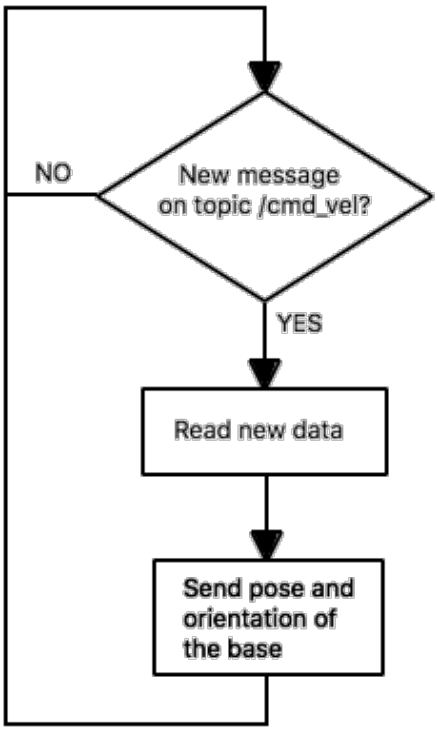

Figura 9.2.1. Diagrama de flux del node *tf_br_robot.py*

```
#!/usr/bin/env python
# -*- coding: latin-1 -*-
import os, sys

__author__ = 'guiu'

import rospy
```

```
import tf
import math
from geometry_msgs.msg import Twist
from math import pi

class TfBroadcaster():
        def __init__(self):
          rospy.init_node('tf_br', anonymous=True)
          rospy.on_shutdown(self.shutdown)

          rospy.Subscriber("cmd_vel", Twist, self.callback_update_tf)

          br = tf.TransformBroadcaster()
          rate = rospy.Rate(20) #Hz
          self.pose_base = [0.0, 0.0, 0.0]
          self.rpy_base = [0.0, 0.0, 0.0]
          rpy_fr_sensor = [0.0, 0.0, 0.0]
          rpy_ri_sensor = [0.0, 0.0, -40.0*pi/180.0]
          rpy_le_sensor = [0.0, 0.0, 80.0*pi/180.0]
          rpy_ba_sensor = [0.0, 0.0, pi]
          quaternion_fr_sensor =
tf.transformations.quaternion_from_euler(rpy_fr_sensor[0], rpy_fr_sensor[1],
rpy_fr_sensor[2])
          quaternion_ri_sensor =
tf.transformations.quaternion_from_euler(rpy_ri_sensor[0], rpy_ri_sensor[1],
rpy_ri_sensor[2])
          quaternion_le_sensor =
tf.transformations.quaternion_from_euler(rpy_le_sensor[0], rpy_le_sensor[1],
rpy_le_sensor[2])
          quaternion_ba_sensor =
tf.transformations.quaternion_from_euler(rpy_ba_sensor[0], rpy_ba_sensor[1],
rpy_ba_sensor[2])

          while not rospy.is_shutdown():
                    quaternion_base =
tf.transformations.quaternion_from_euler(self.rpy_base[0], self.rpy_base[1],
self.rpy_base[2])
                    br.sendTransform(self.pose_base, quaternion_base, rospy.Time.now(),
"tf_br", "map")
                    br.sendTransform((0.085, 0.035, 0.025), quaternion_fr_sensor,
rospy.Time.now(), "front_us_sensor", "tf_br")
                    br.sendTransform((0.075, -0.065, 0.035), quaternion_ri_sensor,
rospy.Time.now(), "right_us_sensor", "tf_br")
                    br.sendTransform((0.075, 0.065, 0.025), quaternion_le_sensor,
rospy.Time.now(), "left_us_sensor", "tf_br")
                    br.sendTransform((-0.13, 0.0, 0.025), quaternion_ba_sensor,
rospy.Time.now(), "back_us_sensor", "tf_br")
                    rate.sleep()
```

```
    def callback_update_tf(self, data):
      if(data.linear.x==1):
            self.pose_base[0] += (0.05*math.cos(self.rpy_base[2]))
            self.pose_base[1] += (0.05*math.sin(self.rpy_base[2])) #FORWARD
      elif(data.linear.x==-1):
            self.pose_base[0] -= (0.05*math.cos(self.rpy_base[2]))
            self.pose_base[1] -= (0.05*math.sin(self.rpy_base[2])) #BACWARD
      else:
            self.pose_base[0] = self.pose_base[0]
            self.pose_base[1] = self.pose_base[1] #STOP
      if(data.angular.z==1):
            self.rpy_base[2] += 0.1 #TURN LEFT
      elif(data.angular.z==-1):
            self.rpy_base[2] -= 0.1 #TURN RIGHT
      else:
            self.rpy_base[2] = self.rpy_base[2]

    def shutdown(self):
      rospy.sleep(1)

if __name__ == '__main__':
    try:
      TfBroadcaster()
    except rospy.ROSInterruptException:
            pass
```

9.3. Node *test_hc_sr04_X.py*

Aquest node és el que llegeix la informació dels sensors d'ultrassons i publica la distància mesurada al missatge de tipus *sensor_msgs/Range* en metres. La *X* del títol significa que és vàlid pels quatre sensors utilitzats (F, B, R, L; inicials en anglès de davant, darrere, dreta i esquerra), la única diferència són els pins per enviar i llegir senyal (*TRIG* i *ECHO*), el nom del node i el *topic* al qual publiquen el missatge.

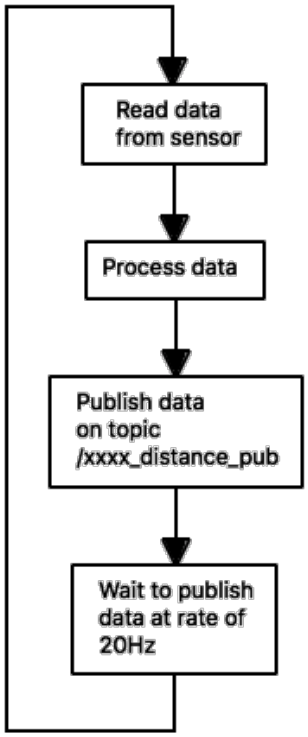

Figura 9.3.1. Diagrama de flux del node test_hc_sr04_X.py

```
#!/usr/bin/env python
# -*- coding: latin-1 -*-
import os, sys

__author__ = 'guiu'

import rospy
import tf
import RPi.GPIO as GPIO
from sensor_msgs.msg import Range
from math import pi

class UltrasonicSensor():
        def __init__(self):
          self.TRIG = 23
          self.ECHO = 24
          GPIO.setmode(GPIO.BCM)
          GPIO.setup(self.TRIG, GPIO.OUT)
          GPIO.setup(self.ECHO, GPIO.IN)
          self.distance = Range()

          rospy.init_node('front_us_sensor', anonymous=False)
          rospy.on_shutdown(self.shutdown)
```

```
        pub = rospy.Publisher('front_distance_pub', Range, queue_size=10)

        rate = rospy.Rate(20) # 10hz

        self.distance.radiation_type = 0
        self.distance.field_of_view = 20*pi/180 #20º passats a radians
        self.distance.min_range = 0.02 #2cm
        self.distance.max_range = 4 #4m --> 400cm
        self.distance.range = 0.0
        self.distance.header.frame_id = '/front_us_sensor'
        while not rospy.is_shutdown():
                self.measure()
                pub.publish(self.distance)
                rate.sleep()

    def measure(self):
        pulse_start = 0.0
        pulse_end = 0.0
        pulse_duration = 0.0
        GPIO.output(self.TRIG, False)
        rospy.loginfo('Waiting for sensor to settle')
        rospy.sleep(0.1)
        GPIO.output(self.TRIG, True)
        rospy.sleep(0.00001)
        GPIO.output(self.TRIG, False)
        while GPIO.input(self.ECHO)==0:
                pulse_start = rospy.get_time()
        while GPIO.input(self.ECHO)==1:
                pulse_end = rospy.get_time()
        pulse_duration = pulse_end - pulse_start
        self.distance.range = pulse_duration*17150/100
        rospy.loginfo('******************************')
        rospy.loginfo('Distance = %f' % self.distance.range + ' m')
        rospy.loginfo('******************************')

    def shutdown(self):
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        UltrasonicSensor()
    except rospy.ROSInterruptException:
            pass
```

9.4. Fitxer *launch*

Aquest tipus de fitxers s'han de guardar dins d'una carpeta amb el nom *launch* dins el paquet perquè ROS els pugui localitzar i executar sense problemes. Estant escrits en format XML. En sistemes més complexes permet la

inicialització de paràmetres i variables, però en aquest cas només s'utilitza per executar els diferents nodes de la Raspberry Pi 2 per facilitar la posada en marxa. Així per definir l'execució d'un node s'ha d'introduir el nom del paquet al qual pertany (*pkg*), el nom que se li ha donat quan s'ha creat o amb el que s'ha guardat (*type*) i finalment el nom que se li vol donar al node quan s'executi (*name*).

```
<launch>
	<node pkg="testing_leap" type="tf_br_robot.py" name="tf_br_robot"/>
	<node pkg="testing_leap" type="test_hc_sr04_F.py" name="front_sensor"/>
	<node pkg="testing_leap" type="test_hc_sr04_B.py" name="back_sensor"/>
	<node pkg="testing_leap" type="test_hc_sr04_R.py" name="right_sensor"/>
	<node pkg="testing_leap" type="test_hc_sr04_L.py" name="left_sensor"/>
	<node pkg="testing_leap" type="leap_teleop_2.py" name="leap_teleop"/>
</launch>
```

9.5. Node Arduino

Aquest és el programa descarregat a l'Arduino, el qual llegeix el missatge publicat pel node *leap_teleop.py* per saber com s'ha de moure i accionar el o els motors que toca en cada cas.

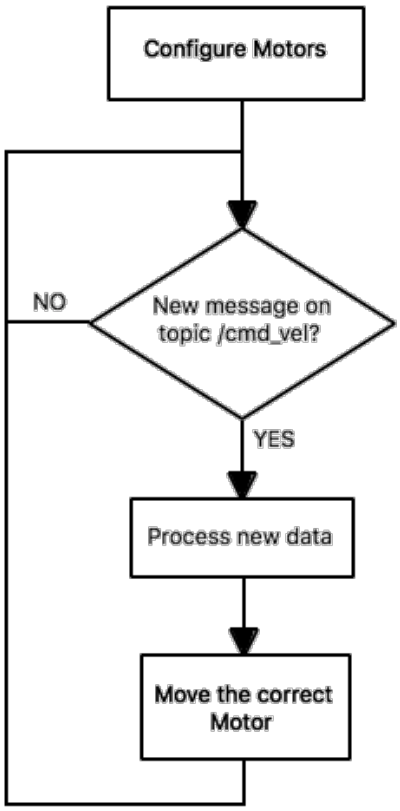

Figura 9.5.1. Diagrama de flux del node de l'Arduino

```
#include <DynamixelSerial1.h>

#include <ros.h>
#include <geometry_msgs/Twist.h>

int currentTime = millis();
int lastTime = millis();
int now =millis();
int front = 1;
int Stop = 0;
int back = 2;
int right = 3;
int left = 4;
int RWh = 1;
int LWh = 2;

ros::NodeHandle nh;
```

```
/*****************************/
/**  ROS message callback  **/
/*****************************/
void messageCb(const geometry_msgs::Twist& vel_msg) {
  if(vel_msg.linear.x==1) {
    motorControl(front);
  } else if(vel_msg.linear.x==-1) {
    motorControl(back);
  } else if(vel_msg.angular.z==-1) {
    motorControl(right);
  } else if(vel_msg.angular.z==1) {
    motorControl(left);
  } else {
    motorControl(Stop);
  }
  lastTime = millis();
}

ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", messageCb);

void setup() {
  Dynamixel.begin(1000000, 2);
  Dynamixel.ledStatus(RWh, ON);
  Dynamixel.ledStatus(LWh, ON);
  nh.initNode();
  nh.subscribe(sub);
}

void loop() {
  currentTime = millis();
  nh.spinOnce();
}

void motorControl(int mode) {
  switch(mode) {
      case 0: /** Stop Motors **/
```

```
  Dynamixel.turn(RWh, RIGTH, 0);
  Dynamixel.turn(LWh, RIGTH, 0);
  Dynamixel.ledStatus(RWh, ON);
  Dynamixel.ledStatus(LWh, ON);

  break;

case 1: /** Run Forward **/
  Dynamixel.setEndless(RWh, ON);
  Dynamixel.setEndless(LWh, ON);
  Dynamixel.ledStatus(RWh, OFF);
  Dynamixel.ledStatus(LWh, OFF);
  Dynamixel.turn(RWh, RIGTH, 200);
  Dynamixel.turn(LWh, LEFT, 200);

  break;

case 2: /** Run Backward **/
  Dynamixel.setEndless(RWh, ON);
  Dynamixel.setEndless(LWh, ON);
  Dynamixel.ledStatus(RWh, OFF);
  Dynamixel.ledStatus(LWh, OFF);
  Dynamixel.turn(RWh, LEFT, 200);
  Dynamixel.turn(LWh, RIGTH, 200);

  break;

case 3: /** Turn Right **/
  Dynamixel.setEndless(RWh, ON);
  Dynamixel.setEndless(LWh, ON);
  Dynamixel.ledStatus(RWh, OFF);
  Dynamixel.ledStatus(LWh, OFF);
  Dynamixel.turn(RWh, RIGTH, 0);
  Dynamixel.turn(LWh, LEFT, 200);

  break;
```

```
    case 4: /** Turn Left **/
      Dynamixel.setEndless(RWh, ON);
      Dynamixel.setEndless(LWh, ON);
      Dynamixel.ledStatus(RWh, OFF);
      Dynamixel.ledStatus(LWh, OFF);
      Dynamixel.turn(RWh, RIGTH, 200);
      Dynamixel.turn(LWh, LEFT, 0);

      break;

    default:
      Dynamixel.turn(RWh, RIGTH, 0);
      Dynamixel.turn(LWh, RIGTH, 0);
      Dynamixel.ledStatus(RWh, ON);
      Dynamixel.ledStatus(LWh, ON);

      break;
  }
}
```

10. Bibliografia

http://savageelectronics.blogspot.com.es/2011/01/arduino-y-dynamixel-ax-12.html
https://www.leapmotion.com/product/desktop
https://en.wikipedia.org/wiki/Leap_Motion
http://wiki.ros.org/ROS/StartGuide
http://mediawiki.isr.ist.utl.pt/images/0/0f/SCRF2013_intro1.pdf
http://wiki.ros.org/ROSberryPi/Setting%20up%20Hydro%20on%20RaspberryPi
http://seeg.mae.carleton.ca/installing-ros-hydro-on-the-raspberry-pi/
http://wiki.ros.org/rosserial_arduino/Tutorials/
https://www.leapmotion.com/setup/linux
https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit
http://wiki.ros.org/ROS/Tutorials/
http://www.crustcrawler.com/motors/AX12/index.php?prod=63
http://www.dfrobot.com/index.php?route=product/product&product_id=65#.Vy9ZvjbwpPw
http://wiki.ros.org/Parameter%20Server
http://wiki.ros.org/Messages
http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv#Creating_a_srv
http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams
http://wiki.ros.org/catkin/CMakeLists.txt#msgs_srvs_actions
http://wiki.ros.org/srv
http://wiki.ros.org/action/show/rosmsg?action=show&redirect=rossrv
http://wiki.ros.org/Services
http://wiki.ros.org/rqt_graph
http://wiki.ros.org/Topics
http://wiki.ros.org/Nodes
https://en.wikipedia.org/wiki/Peer-to-peer
http://wiki.ros.org/ROS/Introduction
http://wiki.ros.org/Courses
https://cse.sc.edu/~jokane/agitr/
https://en.wikipedia.org/wiki/Robot_Operating_System
https://en.wikipedia.org/wiki/Ros

http://wiki.ros.org/tf/Tutorials
https://help.ubuntu.com/lts/serverguide/openssh-server.html
http://wiki.ros.org/ROS/NetworkSetup
http://wiki.ros.org/ROS/Tutorials/MultipleMachines
http://www.tutorialspoint.com/python
https://wiki.ubuntu.com/ARM/RaspberryPi
http://raspberrypi.stackexchange.com/questions/7178/ssh-permission-denied-please-try-again
http://elinux.org//RPi_Setting_up_a_static_IP_in_Debian
http://support.robotis.com/en/product/dynamixel/ax_series/ax-12w.htm
http://www.oppedijk.com/robotics/control-dynamixel-with-raspberrypi
https://github.com/iandanforth/pydynamixel
https://canvas.harvard.edu/courses/7567/pages/getting-started-ros-turtlebot-sensors-and-code
http://wiki.ros.org/ROSNodeTutorialPython
https://www.modmypi.com/blog/tutorial-tactile-switch
http://www.modmypi.com/blog/how-to-install-the-gpio-python-library
http://www.modmypi.com/bloc/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi