FACULTAT
DE CIÈNCIES, TECNOLOGIA
I ENGINYERIES

UVIC | UVIC·UCC

Treball de Fi de Grau de Desenvolupament

ESTUDI PRÀCTIC DE L'ENTORN MATLAB-SIMULINK I EL ROSNI

Marc Suñen Collell

Grau en Enginyeria Mecatrònica

Tutor/a: Moisès Serra Serra

Co-tutor: Sergi Grau Carrión

Vic, Juny de 2023

Agraïments

Agraïments als docents i tutors del treball, per els coneixements i consells proporcionats durant la realització d'aquest treball.

A la meva família, en especial als meus pares i a la Sònia, la meva parella, que m'han ajudat i recolzat durant moltes tardes i dies que he estat centrat en el projecte.

A en Sergi de las Muelas i en Marc Sala, autors del robot ROSNI, pels consells i aclariments sobre l'arquitectura del robot.

Resum

Títol: *Estudi pràctic de l'entorn Matlab (Simulink) - ROSNI*
Autor: Marc Suñen Collell
Co-Tutors: Dr. Moisès Serra Serra (UVic) i Dr. Sergi Grau Carrión (UVic)
Data: Juny de 2023
Paraules clau: Matlab, Simulink, ROS, comunicació, robòtica, mètode.

Les plataformes de càlcul i programació, com ara el Matlab o Simulink han arribat a ser una eina molt potent i en nombrosos casos s'han convertit en imprescindibles per executar simulacions i prediccions de precisió en estudis d'àmbits molt diferents. Aquestes plataformes, sumades a la possibilitat d'enllaçar-les a un estàndard d'operació i execució per paquets, implementats a un sistema operatiu com pot ser el *Robot Operating System (ROS)*, resulta un entorn molt potent per a realitzar estudis i implementacions reals en l'àmbit de la robòtica industrial i robòtica mòbil.

El projecte resulta un estudi teòric i pràctic de l'entorn Matlab-Simulink comunicat amb el robot mòbil ROSNI, creat per dos ex-estudiants de la Universitat de Vic, Sergi de las Muelas i Marc Sala. L'objectiu principal és aconseguir detallar un procés d'execució i donar instruccions pràctiques per implementar una comunicació entre aquests dos entorns i permetre-hi desenvolupar qualsevol projecte relacionat amb la robòtica simulada i implementada.

Summary

Title: *Practical study of the Matlab (Simulink) – ROSNI*
Author: Marc Suñen Collell
Supervisor: Dr. Moisès Serra Serra (UVic) i Dr. Sergi Grau Carrión (UVic)
Date: June 2023
Keywords: Matlab, Simulink, ROS, communication, robotics, method.

The computing and programming platforms, such as Matlab or Simulink, have become a powerful tool and in many cases have become essential for carrying out precise simulations and predictions in studies from very different fields. These platforms, combined with the possibility of linking them to a standard operation and execution for packages, implemented on an operating system such as Robot Operating System (ROS), result in a very powerful environment for carrying out real studies and implementations in the field of industrial and mobile robotics.

The project results in a theoretical and practical study of the Matlab-Simulink environment connected with the mobile robot ROSNI, created by two ex-students of the University of Vic, Sergi de las Muelas and Marc Sala. The main objective is to detail execution process and provide practical instructions for implementing communication between these two environments and enable the development of any project related to simulated and implemented robotics.

Índex de Continguts

Llista de Taules

Llista de Figures

1. Introducció

La determinació de desenvolupar aquest projecte sorgeix d'una suma d'idees inicials relacionades amb la robòtica mòbil i industrial. Aquestes idees han portat a la detecció d'una necessitat de coneixements i estudis realitzats a la Universitat de Vic, sobre les possibilitats que té la plataforma Matlab en projectes relacionats en l'àmbit de la robòtica, com pot ser amb el robot mòbil ROSNI, creat per ex-alumnes de la Universitat.

Els coneixements treballats a les diferents assignatures del grau en Enginyeria Mecatrònica han aportat una visió dels punts necessaris a treballar en aquest projecte, i que podrien ser susceptibles d'aplicar-se en aquestes.

La motivació principal del projecte és dotar d'eines i obrir un camí inicial en l'estudi de la robòtica a la Universitat de Vic, mitjançant Matlab i aprofitant els entorns operacionals i de programació de la robòtica sobre el robot ROSNI.

El robot ROSNI, ens proporciona totes les funcionalitats necessàries per desenvolupar el treball i assolir els objectiu marcats d'aquest. El funcionament del robot es basa amb *Robot Operating System* (ROS), un sistema operatiu per a robots i amb el que permet realitzar les diferents implementacions amb el Matlab i Simulink tenint present l'entorn de treball de la Figura 1.

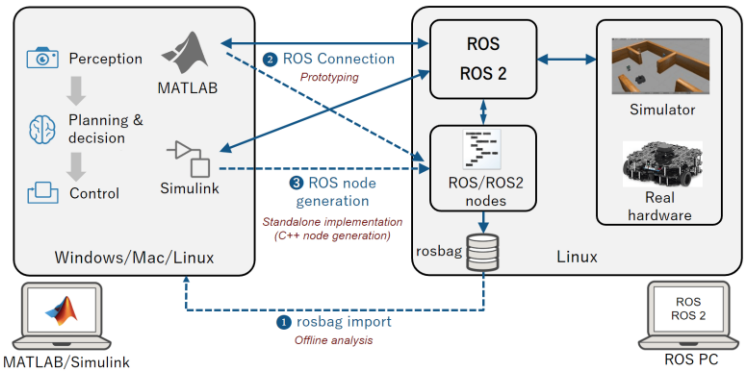


Figura 1. Visió general de l'entorn Matlab/Simulink i ROS.[1]

Actualment, l'ús de plataformes de càlcul i programació com Matlab-Simulink s'han convertit en una eina essencial per a la realització d'estudis en àmbits molt diferents. Aquest fet és degut a la seva gran capacitat per realitzar simulacions i prediccions.

En aquest sentit, la comunicació entre el Matlab-Simulink i el ROS proporciona un entorn molt potent per a realitzar estudis i implementacions reals en l'àmbit de la robòtica mòbil i industrial, fets que procedirem a demostrar en aquest treball.

[1] Font de la Figura 1: https://es.mathworks.com/help/ros/

1.1. Objectius

Els objectius principals del projecte són recollir diferents mètodes i funcions de la comunicació entre Matlab-Simulink i ROS, acabant amb varis exemples implementats per a poder consolidar la recerca desenvolupada durant aquests mesos.

L'objectiu és controlar i modelar, a través de Matlab-Simulink, el robot mòbil ROSNI creant un node entre ells que permeti controlar diferents funcions del robot.

Els objectius específics d'aquest projecte són:

- Obtenir i seleccionar la informació necessària referent a la comunicació entre plataformes.
- Detallar el procediment seguit per configurar la comunicació entre plataformes, donant les eines necessàries per implementar qualsevol exemple.
- Crear un model publicador i subscriptor entre plataformes simple i didàctic.

1.2. Estructura de la memòria

Aquesta memòria s'estructura per capítols, organitzant la informació obtinguda amb la finalitat de facilitar la interpretació i resum d'aquesta. Seguidament es detallen els diferents capítols mitjançant una breu descripció.

- Capítol 1, introducció i objectius generals del projecte.
- Capítol 2, Estat de l'art del projecte on es detalla la documentació referent al Matlab i les seves dependències, introducció del ROS, característiques a controlar del hardware (ROSNI), juntament amb altre generalitats que siguin necessàries detallar en aquest apartat.
- Capítol 3, es detalla el procediment emprat per tal de configurar la comunicació entre plataformes.
- Capítol 4, implementació del exemples avaluats durant el projecte.
- Capítol 5, conclusions i futures millores per a futurs projectes.

2. Estat de l'art

En els darrers anys, la branca de la Mecatrònica s'ha consolidat com una disciplina fonamental per al desenvolupament de projectes innovadors en àrees com la robòtica industrial, la robòtica mòbil, l'automatització i el control de processos. En aquest sentit, les plataformes de programació i simulació com Matlab-Simulink i els sistemes operatius especialitzats amb robòtica, com pot ser el ROS, han adquirit una gran importància ja que permeten la simulació, el control i la supervisió de sistemes mecatrònics d'una forma eficaç. El flux de les dades definit mitjançant tòpics entre el robot i Matlab, vegeu la Figura 2, també és un dels punts claus d'aquest projecte i que es detallen en aquest capítol.

A l'Estat de l'art, analitzem les característiques i les funcionalitats dels components que es volen utilitzar en el projecte, així com les aplicacions i les tendències més destacades en aquest entorn. Aquest estudi permet establir les bases per obtenir una visió extensa de les diferents eines disponibles.

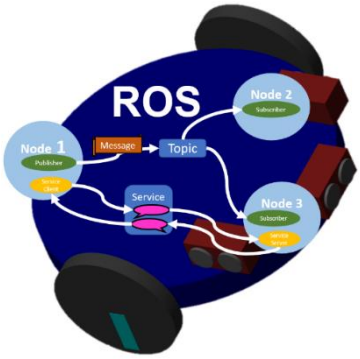


Figura 2. Estructura del flux de dades d'un robot mòbil amb ROS.[2]

2.1. Matlab i Simulink

Inicialment, la plataforma Matlab era un llenguatge de programació que permetia realitzar càlculs matricials amb facilitat. La primera versió comercial va ser llançada el 1984 per la companyia MathWorks. Imatge de la companyia reconeguda per el logotip representat a la Figura 3.

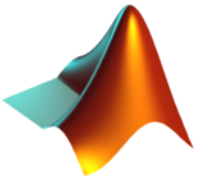


Figura 3. Logotip de Matlab i Mathworks.

[2] Font de la Figura 2: https://es.mathworks.com/matlabcentral/fileexchange/118630-matlab-and-simulink-ros-tutorials

Amb el temps, Matlab ha anat evolucionant afegint noves funcions. L'any 1986 s'hi va afegir la *Toolbox* de Control, que permetia dissenyar i simular sistemes de control. Posteriorment, es van afegir altres *Toolbox* per a diferents àmbits com ara el processament de senyals, el processament d'imatges i estadística.

El 1994 es va llançar Matlab 4, que va ser la primera versió de Matlab en incorporar Simulink, la plataforma de simulació de sistemes dinàmics, i que s'utilitzarà en aquest projecte en qüestió. A partir d'aquest moment, Matlab i Simulink es van convertir en eines imprescindibles per a la simulació i el disseny de sistemes complexos en diverses àrees com la indústria, l'automoció i la robòtica.

Últimament, ha seguit evolucionant i millorant, incorporant noves funcionalitats i facilitant-ne el seu ús. Actualment, Matlab i Simulink són àmpliament utilitzats en la investigació i la indústria, en l'enginyeria, la física, la biotecnologia i en molts altres àmbits.

Si aprofundim més en aquesta eina, Simulink representa una eina gràfica de simulació i disseny de sistemes dinàmics i que permet dissenyar, modelar i simular sistemes complexos. La simulació permet fer des dels sistemes més senzills fins a sistemes més avançats de control, comunicacions, processament de senyals, sistemes de potència, sistemes mecànics, entre altres.

Simulink està basat en un entorn gràfic que utilitza blocs per crear models de sistemes dinàmics. Aquest entorn gràfic facilita la seva utilització per a les persones que no tenen un gran coneixement en programació, ja que no requereixen de la creació de codi, segons el cas.

També disposa d'eines d'anàlisi i simulació avançades que permeten provar el comportament del sistema, modificar el disseny i ajustar els paràmetres per aconseguir els resultats desitjats. Això permet que sigui una eina molt útil pels enginyers i tecnòlegs que necessiten dissenyar i provar sistemes complexos.

Referent a les comunicacions, la plataforma permet comunicar-se amb altres programes externs com ara programes de CAD i C++. Aquest fet, provoca que sigui una eina molt flexible alhora d'integrar-la amb altres eines de disseny i simulació.

La plataforma permet treballar amb codi a alt nivell i ambdues direccions de treball, és a dir, tant es pot generar codi des de Matlab o Simulink i executar-lo, o bé, generar-lo des d'una d'aquestes plataformes i descarregar-lo a l'entorn ROS al qual s'estigui connectat.

Anteriorment, s'ha mencionat l'exemple de la *Toolbox* de Control, però per cada projecte a realitzar es requereixen certes *Toolbox* a instal·lar dintre la plataforma i que cal avaluar en cada cas. Bàsicament, les *Toolbox* de Matlab i Simulink són conjunts de funcions i eines que amplien la funcionalitat d'aquestes plataformes. Aquestes eines proporcionen funcions especialitzades per a una àmplia gamma d'aplicacions, com ara processament de

senyals, control, *Computer vision*[3], estadística, optimització, comunicacions, entre d'altres. Cada *Toolbox* conté funcions i recursos específics que s'adapten a les necessitats dels possibles usuaris i aplicacions.

Per tal de poder desenvolupar el projecte, es requerirà de la instal·lació de les següents *Toolbox*:

- *ROS Toolbox*
- *Robotic System Toolbox*
- *Navigation Toolbox*

2.1.1. ROS Toolbox

Una de les llibreries de Matlab utilitzada en aquest treball és la llibreria *ROS Toolbox*. És una eina que permet connectar l'entorn de Matlab i Simulink amb l'entorn ROS desitjat i amb una comunicació bidireccional, és a dir, configurant els blocs de la llibreria com a subscriptors o com a publicadors a través de nodes ROS. El mètode de comunicació que utilitza la llibreria *ROS Toolbox* per a connectar els nodes és a través del protocol TCP/IP, més endavant detallarem aquest protocol.

A més de permetre la creació de nodes entre Simulink i ROS, la llibreria ofereix suport per a crear simulacions i anàlisis de dades que ja es troben integrades en el propi entorn.

Aquestes funcions es troben dintre l'entorn i disponibles amb format de blocs gràfics, vegeu la Figura 4, que faciliten la seva utilització i configuració. Més endavant, s'observarà el model implementat a Simulink definit en aquest treball mitjançant els blocs que s'ofereix en aquesta llibreria.

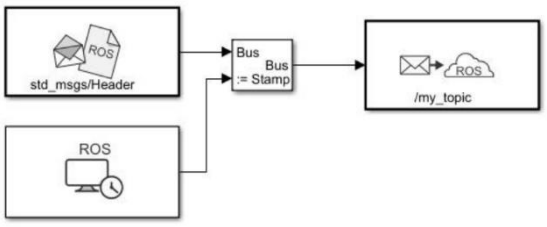


Figura 4. Exemple model Simulink amb blocs de la llibreria *ROS Toolbox*.[4]

Algunes de les funcions clau dels blocs i les comandes existents en aquesta llibreria són les següents:

[3] https://ca.wikipedia.org/wiki/Visi%C3%B3_artificial

[4] Font de la Figura 4: https://es.mathworks.com/help/ros/ug/time-stamp-a-ros-message-using-current-time-in-simulink.html

- Blocs de connexió: aquests blocs permeten connectar-se a un entorn ROS, establir una comunicació amb altres nodes de ROS, publicar i subscriure's a missatges de ROS.
- Blocs de transformació: aquests blocs permeten transformar punts, vectors i marcs de referència entre diferents sistemes de coordenades en un entorn ROS.
- Blocs de visualització: aquests blocs permeten visualitzar dades de sensor i altres informacions rellevants en un entorn ROS, com ara imatges, núvols de punts i models 3D.
- Blocs de control: aquests blocs permeten controlar robots en un entorn ROS, definir i executar tasques de moviment, planificar rutes i evitar col·lisions.
- Blocs de simulació: aquests blocs permeten simular robots i sistemes en un entorn ROS, proporcionant funcions per simular sensors, actuadors i dinàmica del robot.

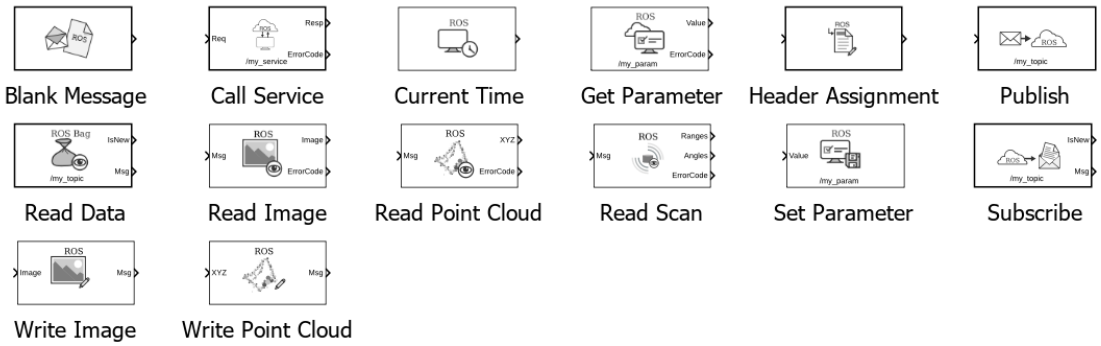


Figura 5. Blocs disponibles de ROS Toolbox a Simulink.

2.1.2. Robotic System Toolbox

L'eina *Robotic System Toolbox* de Matlab es basa en un conjunt de funcions i eines per al disseny, simulació i control de sistemes robòtics. Aquesta *Toolbox* s'integra amb ROS (Robot Operating System), un conjunt de llibreries i eines per al desenvolupament de software de robòtica.

Entre les seves característiques, la *Robotic System Toolbox* de Matlab ofereix visualització de robots en 3D, càlcul de cinemàtica directa i inversa, disseny i implementació de controladors de moviment, simulació de robots i interfícies amb ROS per a la comunicació amb robots mòbils.

Aquesta *Toolbox* permet als usuaris simular i validar el comportament dels seus sistemes robòtics, així com comunicar-se amb robots mòbils a través de ROS. Això fa que sigui una eina molt útil pels investigadors i desenvolupadors de robòtica que treballen amb comunicacions entre Matlab i robots mòbils amb ROS.

Les característiques més importants d'aquesta eina són:

- Visualització de robots: ofereix recursos per a la visualització de robots en 3D, incloent opcions per mostrar trajectòries, punts d'interès i altres detalls del robot.
- Cinemàtica directa i inversa: ofereix funcions per al càlcul de la posició i orientació d'un robot en funció dels angles (cinemàtica inversa) i pel càlcul dels angles en funció de la posició i orientació del robot (cinemàtica directa).
- Control de moviment: ofereix funcions per al disseny i implementació de controladors de moviment per a robots, incloent PID, LQR i control adaptatiu.
- Simulació de robots: inclou eines per a la simulació de robots, permetent als usuaris provar i validar el comportament dels seus sistemes robòtics.
- Interfícies amb ROS: la *Robotic System Toolbox* inclou funcions per a la comunicació amb ROS, fet que permet als usuaris enviar i rebre dades des de ROS cap a Matlab i al contrari.

2.1.3. Navigation Toolbox

La *Navigation Toolbox* de Matlab és una eina per al disseny, simulació i implementació de sistemes de navegació en vehicles autònoms i altres aplicacions de robòtica mòbil. Aquesta eina inclou funcions per a la planificació de rutes, la localització i la creació de mapes.

Entre les seves característiques, la *Navigation Toolbox* ofereix diverses opcions per a la planificació de rutes basada en objectius, de trajectòries i en visió. Aquesta eina, també ofereix eines per a la localització en temps real, que permeten als robots determinar la seva posició amb alta precisió, incloent el filtratge Kalman i algoritmes de localització, com l'algoritme Montecarlo[5] per exemple.

La *Navigation Toolbox* també té disponibles funcions per a la creació de mapes d'entorns del robot en qüestió, amb l'ús de sensors i dades GPS per a la creació de mapes de l'entorn en temps real. El tractament dels senyals provinents dels sensors amb les funcions d'aquesta eina permeten als robots crear i actualitzar mapes precisos de l'entorn alhora que es desplacen.

Una altra característica important de la *Navigation Toolbox* és la seva capacitat per a la simulació de sistemes de navegació autònoma. Això permet als usuaris provar i validar els seus sistemes de navegació en un entorn controlat, sense la necessitat de desplegar el robot en un entorn real. Aquest fet, pot ajudar a reduir el temps i el cost de desenvolupament, així com els riscos associats a les proves inicials de sistemes en entorns reals.

[5] L'algorisme Montecarlo és una tècnica estadística no determinista que s'utilitza per aproximar expressions matemàtiques complexes i costoses d'avaluar amb precisió.
https://ca.wikipedia.org/wiki/M%C3%A8tode_de_Montecarlo

Les funcionalitats bàsiques i blocs que conté aquesta llibreria són:

- Funcions de localització: aquestes funcions permeten estimar la posició i orientació d'un robot en un entorn desconegut utilitzant sensors com ara GPS, IMU o càmeres.
- Funcions de mapeig: aquestes funcions permeten construir mapes d'un entorn a partir de dades de sensors, com ara núvols de punts, imatges o escanejos làser.
- Funcions de planificació de trajectòries: aquestes funcions permeten generar trajectòries òptimes per al moviment d'un robot en un entorn complex, tenint en compte factors com ara la geometria de l'entorn i les restriccions del robot.
- Blocs de control de robots: aquests blocs permeten controlar el moviment d'un robot en temps real, utilitzant dades de sensors i algorismes de planificació de trajectòries.
- Blocs de simulació: aquests blocs permeten simular el comportament d'un robot en un entorn virtual, utilitzant dades de sensors simulades i algorismes de control de robots.

2.2. ROS (Robot Operating System)

El robot mòbil ROSNI, sobre el que desenvoluparem el nostra projecte, conté un entorn ROS (*Robot Operating System*) instal·lat a un Raspberry Pi (Linux), per tant en aquest apartat tractarem de recollir un coneixement general sobre aquest sistema dissenyat per a robots mòbils.

ROS, per tant, és un entorn de desenvolupament de software de codi lliure per a robots industrials i mòbils. Aquest sistema operatiu, creat el 2007 per un laboratori d'intel·ligència artificial de Stanford, va sorgir amb la finalitat de dotar als desenvolupadors del món de la robòtica d'un entorn homogeni i amb serveis estàndards.

Aquests serveis estàndards del sistema operatiu, controlen a baix nivell de programació els processos d'enviament i rebuda de missatges, mitjançant paquets encarregats de crear els nodes de comunicació amb altres plataformes. La forma de comunicar-se amb nodes, descriu la seva arquitectura basada amb *grafs*[6].

El software de l'entorn ROS, s'orienta per operar sobre sistemes Linux (Ubuntu), però existeixen altres sistemes operatius com ara Windows, Mac OS o Debian que també poden acceptar ROS, tot i que es consideren experimentals.

Aquest treball s'orientarà en operar l'entorn ROS sobre una versió Linux (Ubuntu), on es detallarà més endavant. Tenint en compte la informació consultada, la versió entre plataformes a instal·lar, reben una gran importància alhora de generar una comunicació entre aquestes.

[6] Objecte matemàtic que permet modelar una xarxa.

Precisament, per a comunicar-se entre altres sistemes, ROS accepta diferents protocols de comunicació però el més comú és el TCP/IP. Per altra banda, per comunicar-se amb dispositius com ara sensors, actuadors o motors, permet la connexió amb interfícies USB, serial i Ethernet.

2.2.1. Flux de missatges i estructura

En un entorn ROS, la comunicació dels nodes es produeix a través de tòpics. Un tòpic és un canal de comunicació en què els nodes publicadors envien dades i els nodes subscriptors les reben. Per exemple, un node publicador pot enviar dades que genera un sensor, com ara la informació d'una càmera o la de un sensor LIDAR, a través d'un tòpic. Els nodes subscriptors es poden subscriure al mateix tòpic per rebre-les. Així, la informació es pot distribuir i processar de manera eficient en una xarxa de nodes.

El flux de missatge és el procés de transferència de dades d'un node a un altre a través d'un tòpic. Un node publicador envia dades a través del tòpic, que es distribueixen a tots els nodes subscriptors que estan connectats al mateix tòpic. Cada node subscriptor rep una còpia de les dades enviades pel node publicador. El flux de missatge és bidireccional, ja que un node també pot ser al mateix temps publicador i subscriptor.

Els subscriptors i publicadors són les interfícies que permeten als nodes interactuar amb els tòpics. Un node publicador publica missatges en un tòpic, mentre que un node subscriptor s'inscriu en un tòpic per rebre missatges. Quan un node publicador envia dades a través d'un tòpic, aquestes dades són enviades a tots els nodes subscriptors que estan connectats al mateix tòpic. Els nodes subscriptors poden processar les dades rebudes i prendre decisions basades en aquestes. Vegeu la Figura 6.

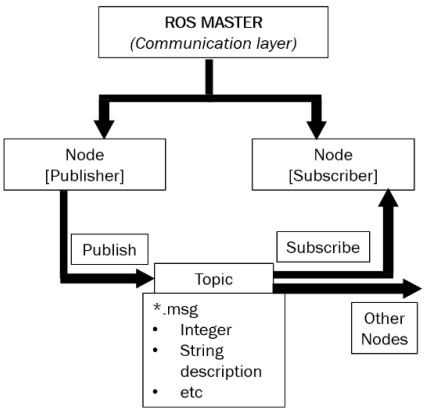


Figura 6. Diagrama de flux d'un tòpic ROS.[7]

[7] https://www.researchgate.net/figure/ROS-flowchart-of-nodes-in-publishing-and-subscribing_fig2_336935818

2.3. ROSNI

El robot ROSNI és un robot mòbil dissenyat i construït per ex-estudiants de la Universitat de Vic, en concret, per en Sergi de las Muelas i en Marc Sala. El robot mòbil en qüestió, valorant el seu conjunt des d'una perspectiva més genèrica, la seva arquitectura es pot englobar a una gamma de robots equivalent als *TurtleBot* que es poden trobar actualment al mercat.

En aquest apartat, es resumiran els aspectes més importants a tenir en compte per tal de poder utilitzar el robot ROSNI durant el desenvolupament d'aquest projecte.

Figura 7. Imatge del primer ROSNI.

El primer ROSNI va ser creat el 2022, i actualment existeixen dos robots més, és a dir, 3 robots ROSNI disponibles a la Universitat de Vic.

El robot ROSNI està fabricat majoritàriament amb peces fabricades mitjançant impressora 3D. Es poden consultar els materials utilitzats a la memòria *ROS Educational Robot Kit* (2022) i les peces en format CAD al repositori del GitHub del ROSNI que trobareu a la bibliografia d'aquesta memòria.

Cal remarcar, que aquest robot, té una limitació sobre la navegació autònoma ja que segons un dels autors del robot, actualment aquests paquets no s'han acabat de desenvolupar al 100% i és una de les funcions que actualment el robot no permet realitzar-la correctament.

2.3.1. Descripció dels nodes i funcionalitats del ROSNI

En aquest apartat, detallarem tots els paquets de ROS i nodes que conté el ROSNI, un dels aspectes més importants a conèixer per el projecte. En el robot, existeixen paquets que han set creats per part dels autors i d'altres que són paquets ja creats, descarregats de la xarxa. Segons la memòria dels autors del ROSNI, els paquets existents en el robot són compatibles amb els missatges que pugui enviar un dels seus homòlegs com pot ser el *TurtleBot*, per exemple.

```
/base_link_to_laser4
/joy_node
/matlab_global_node_47916
/robot_dif_controller
/robot_state_publisher
/robot_teleop_joy
/rosout
/slam_gmapping
/socket_node
/start_slam_nav_node_1150_1655306225464
/ydlidar_node
```

Figura 8. Llistat de *rosnode list.*

- *Robot description*
  El fitxer URDF (fitxer de descripció universal del robot) on s'especifica quants nodes té el robot i on es poden trobar. Aquest paquet també conté altres característiques del robot, com ara les propietats de massa, inèrcia i el tipus de moviments permesos, entre d'altres.

- *Robot start up*
  Conté els fitxers de llançament que inicien tots els nodes necessaris per inicialitzar el robot (per exemple, Robot base controller, YDLidar ROS o el node SLAM). Aquest fitxer facilita l'inici de tots els nodes ROS a la vegada i obté el control del que està passant.

- *Robot teleop*
  Conté el node que transforma les dades del *joystick* en missatges Twist (Vx, Vy, ω), permetent que l'usuari controli a distància el robot.

- *Robot base controller*
  Es troben els nodes que controlen els missatges Twist (`/robot/cmd_vel`). També aplica la cinemàtica per generar la velocitat de les rodes i escriure-la als motors a través de l'Arduino Nano del robot.

- *Node controller*
  Serveis per iniciar o aturar el node per fer SLAM.

- *Node controller msgs*
  Descripció dels serveis personalitzats i missatges utilitzats en el paquet *Node controller*.

- *Socket Node*
  Conté el node del servidor que escolta les sol·licituds de l'aplicació.

- *YDLidar ROS*
  Es troben els fitxers per poder llegir i utilitzar el lidar amb ROS.

- *Joystick divers*
  Conté els arxius per poder obtenir les dades del joystick amb ROS. Una de les funcionalitats del robot és poder ser controlat per un comandament de Bluetooth. Aquest dispositiu està associat amb el robot i totes les dades enviades des dels joysticks són capturades per un tòpic ROS anomenat `/joy`. El node *Robot teleop* es subscriu a `/joy` i converteix les dades dels joysticks a les velocitats corresponents amb el Twist.

Una altra funcionalitat del robot, ja comentada anteriorment, és la possibilitat de realitzar SLAM (Simultaneous Localization And Mapping). Quan s'activa SLAM, el robot construeix un mapa mitjançant les dades de l'escaneig làser i l'odometria de l'encoder dels motors. Si es vol obtenir el mapa complet d'una habitació, el robot s'ha de moure per tota la zona desitjada.

Cal destacar que el *Socket node* rep qualsevol missatge enviat per l'aplicació, però actualment només hi han implementats aquells que tenen efecte sobre el robot com la petició d'iniciar o aturar l'SLAM i qualsevol altre missatge s'ignora.

2.4. SLAM i odometria

SLAM (*Simultaneous Localization and Mapping*), és un procés clau en la robòtica mòbil que permet a un robot moure's i navegar en un entorn desconegut i/o dinàmic. L'objectiu de SLAM és construir un mapa de l'entorn mentre el robot localitza la seva posició i orientació. A través de l'ús de sensors, com ara càmeres, escàners làser o sensors d'ultrasons recopilen informació sobre l'entorn del robot.

La informació recollida pels sensors es processa amb un algorisme SLAM, que utilitza mètodes de fusió de dades per crear i actualitzar el mapa de l'entorn, estimant la posició i orientació del robot en relació a aquest mapa. En altres paraules, l'algorisme SLAM combina les mesures dels sensors amb informació de la posició anterior per crear un mapa més precís de l'entorn i millorar la seva estimació de la posició actual d'aquest.

Els algorismes SLAM més comuns són els següents:

- FastSLAM: aquest algorisme utilitza un enfocament de partícules per construir un mapa de l'entorn i estimar la posició del robot. FastSLAM divideix el problema SLAM en problemes més petits i més senzills, que cada partícula pot resoldre de forma independent. Això fa que el FastSLAM sigui una opció molt ràpida i eficient.

- EKF-SLAM: l'algorisme EKF-SLAM, o *Extended Kalman Filter SLAM*, és un dels algorismes SLAM més antics i utilitzats. Utilitza un filtre Kalman ampliat per tal d'estimar la posició i la orientació del robot, així com per actualitzar el

mapa de l'entorn. L'EKF-SLAM és relativament senzill i eficaç, però és més susceptible a errors i pot no funcionar bé en entorns dinàmics.

- GraphSLAM: aquest algorisme intenta trobar la millor solució segons les mesures dels sensors i les estimacions de posició del robot. El GraphSLAM és un algorisme robust i pot funcionar bé en entorns molt dinàmics, però és més complex i requereix més temps de processament que altres algorismes SLAM.

- ORB-SLAM: aquest és un algorisme SLAM basat en l'ús de descriptors d'òrbites (Oriented FAST and Rotated BRIEF) per localitzar punts d'interès en una imatge i utilitzar-los per construir el mapa de l'entorn i estimar la posició del robot. ORB-SLAM és relativament senzill i eficient, i té bones prestacions en entorns amb poca il·luminació, però pot tenir dificultats per localitzar-se en entorns amb poques característiques.

La odometria, per la seva banda, és un mètode per mesurar el desplaçament del robot basat en la informació de les rodes del robot. És una de les formes més senzilles i econòmiques de mesurar la posició del robot, ja que els *encoders* de les rodes són relativament fiables i econòmics. No obstant això, la odometria és propensa a errors, ja que no té en compte factors com el lliscament de les rodes o els obstacles en el camí del robot. Per compensar aquests errors, es combina sovint amb altres mètodes de navegació, com l'SLAM. Això és degut a que SLAM pot proporcionar informació més precisa sobre la posició del robot en l'entorn, mentre que la odometria pot proporcionar informació útil per actualitzar la posició estimada del robot.

2.5. Comunicació TCP/IP i UDP

Les comunicacions entre plataformes i el seu flux de dades són un element clau d'aquest projecte, en aquest apartat ens centrem a conèixer els dos protocols que s'utilitzen en casos d'entorns ROS i amb els que també està dissenyat el robot ROSNI. Les comunicacions utilitzades són les TCP/IP i UDP, dos protocols de xarxa que es fan servir per transmetre dades a través d'Internet i altres xarxes. Tots dos protocols són part del model de referència OSI[8], un conjunt de normes que defineixen com les xarxes han de comunicar-se.

TCP/IP és un protocol de comunicació per paquets que s'orienta a la connexió. És a dir, quan hi ha un establiment de la connexió entre els dispositius de la xarxa es crea la transmissió de dades entre ells. El protocol TCP/IP és molt robust i assegura que les dades arribin al seu destí i en l'ordre correctament. Aquest protocol és ideal per a aplicacions que requereixen una transmissió fiable de dades, com ara el correu electrònic, la navegació web i la transferència de fitxers.

[8] https://ca.wikipedia.org/wiki/OSI

UDP, per altra banda, és un protocol de comunicació per paquets sense connexió. Això significa que no hi ha cap establiment de la connexió entre els dispositius de la xarxa abans de la transmissió de dades. Això fa que el protocol UDP sigui més ràpid i eficient que TCP/IP, però també és menys fiable. Les dades que es transmeten amb UDP poden perdre's o arribar amb un ordre incorrecte, i a més no existeix cap mecanisme de correcció d'errors. És a dir, permet que el protocol UDP sigui ideal per a aplicacions que requereixen una transmissió ràpida de dades, com ara la transmissió de vídeo i àudio en temps real.

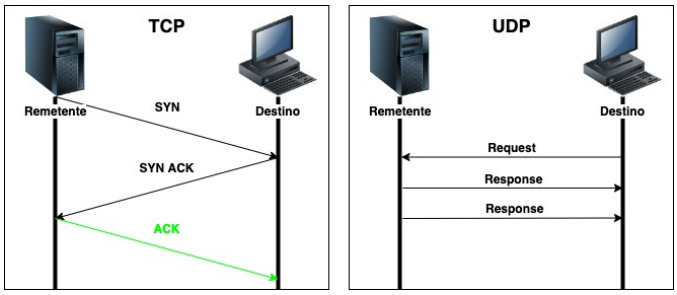


Figura 9. Diferències entre els protocols TCP i UDP.[9]

[9] https://dicionariotec.com/posts/portas-tcp-e-udp

3. Desenvolupament de projecte

En aquests capítol, es detallen els aspectes a tenir en compte alhora de configurar l'entorn de comunicació entre el robot ROSNI i el Matlab-Simulink. Entre els aspectes que es tractaran, reben gran importància i atenció les configuracions TCP/IP, les llibreries instal·lades a Matlab recollint les funcions més importants per el projecte, i com s'envien i s'inicialitzen els tòpics del robot ROSNI. Cal tenir en compte, que cada entorn PC-ROSNI amb el que es treballa té unes característiques variables per a cada usurari en el moment anterior a fer la comunicació, per tant en aquest capítol es detallaran els problemes i solucions que s'han obtingut per a realitzar el projecte, tenint present que en altres casos es pot necessitar alguna acció complementària.

Cal remarcar que aquest projecte s'ha realitzat amb la versió de *Matlab R2023a* i la versió de ROS Noetic instal·lat a la Raspberry Pi del ROSNI.

Per tenir present en tot moment quina serà l'arquitectura emprada en el desenvolupament d'aquest projecte s'ha creat la Figura 10 que representa el diagrama de comunicació entre Matlab i ROSNI.

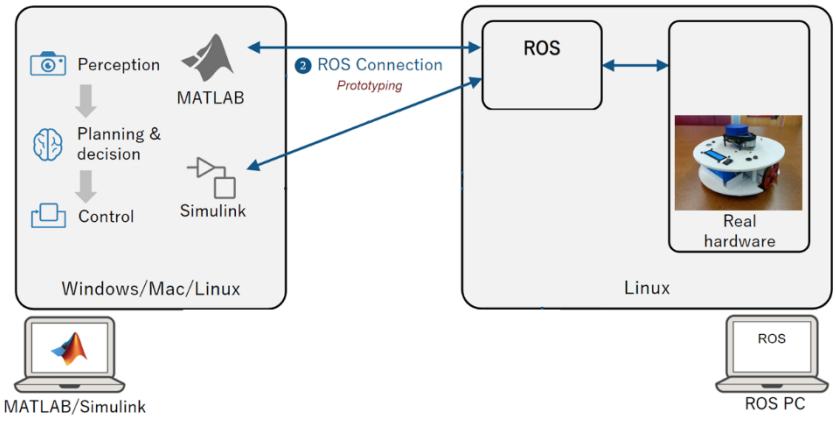


Figura 10. Representació de l'aquitectura del treball.

3.1. Configuració llenguatge C i C++

Com a primer pas, es comprova que l'entorn del Matlab pugui obtenir una versió compatible per interpretar el possible codi Python que pugui intervenir en els diferents blocs o línies de codi a implementar.

Per a la versió de *Matlab R2023a*, durant la primera comprovació realitzada, es detecta que el Matlab no reconeix la versió de Python instal·lada en aquell moment al PC d'usuari.

Cercant la solució a aquest defecte, s'arriba a la deducció que s'ha de tenir en compte que segons la versió de Matlab que es tingui instal·lada, aquest accepta un rang de versions de Python específiques. A la pàgina de MathWorks es pot trobar la documentació de Matlab *Versions of Python Compatible with MATLAB Products by Release*, i consultar la relació de versions entre programaris.

Release	MATLAB Interface MATLAB Engine	MATLAB Compiler SDK	MATLAB Production Server Client Library
R2023a	3.8, 3.9, 3.10	3.8, 3.9, 3.10	3.8, 3.9, 3.10
R2022b	2.7, 3.8, 3.9, 3.10	2.7, 3.8, 3.9, 3.10	3.8, 3.9, 3.10
R2022a	2.7, 3.7, 3.8, 3.9	2.7, 3.8, 3.9	2.7, 3.8, 3.9
R2021b	2.7, 3.7, 3.8, 3.9	2.7, 3.7, 3.8, 3.9	2.7, 3.7, 3.8, 3.9
R2021a	2.7, 3.7, 3.8	2.7, 3.7, 3.8	2.7, 3.7, 3.8

Taula 1. Relació de versions Matlab-Python.

Observant la *Taula 1*, vegeu que per la versió de *Matlab R2023a* les versions de Python compatibles són la 3.8, 3.9 i 3.10. Per tant, es va detectar que la versió que es tenia instal·lada en aquell moment a l'ordinador personal, i amb el qual s'estava treballant, era la versió 3.7 i aquest fet provocava l'error en la detecció de l'entorn Python.

La versió de Python instal·lada per a *Matlab R2023a* és la versió 3.9. S'ha cregut oportú instal·lar aquesta versió per assegurar amb el rang de versions especificades, i un cop instal·lat i comprovat segons Figura 11, s'ha validat el procés de configuració del Python amb la documentació consultada.

```
Command Window
>> pyenv

ans =

  PythonEnvironment with properties:

         Version: "3.9"
      Executable: "C:\Program Files\Python39\python.EXE"
         Library: "C:\Program Files\Python39\python39.dll"
            Home: "C:\Program Files\Python39"
          Status: NotLoaded
   ExecutionMode: InProcess
```

Figura 11. Configuració de l'entorn Python al Matlab.

Alhora d'instal·lar el Python a l'ordinador personal, s'han de tenir en compte les característiques a instal·lar. Es recomana realitzar la instal·lació amb l'opció "Custom" ja que amb l'opció per defecte és probable que s'instal·li la versió reduïda i faltin funcions per a realitzar la configuració amb el Matlab correctament.

3.2. Comunicació PC-ROSNI

La comunicació PC-ROSNI detallada en aquest apartat és rellevant per l'estudi al ser un dels punts que ha portat més complicacions alhora de crear la correcte comunicació entre Matlab i el ROSNI. La qüestió és que aquest és un punt que no és obvi ni que es pugui tenir en compte amb facilitat.

El motiu d'aquest error de configuració, és que la comunicació entre ROSNI i Matlab quan aquest s'ha de subscriure i llegir els valors dels tòpics enviats pel robot no es visualitzen o el valor que s'obté d'aquest tòpic és sempre 0.

En canvi, s'observa que amb la direcció de comunicació inversa, si es publica el tòpic */robot/cmd_vel* de Matlab cap al ROSNI, aquest arriba sense problemes al robot i es desplaça segons els valors de velocitat descrits en el tòpic.

Per tant, la conclusió del problema és que, per defecte, el flux de comunicació entre el Matlab i el ROSNI és només en direcció de publicació des del Matlab cap al ROSNI.

El motiu per a que no es produeixi correctament la comunicació amb una de les direccions de comunicació és que la IP del robot ROSNI, l'ordinador personal en qüestió no detecta aquesta IP en el llistat de *hosts* de la comunicació TCP/IP. Per solucionar el problema de comunicació i visualització de les dades dels tòpics des del ROSNI cap al Matlab és la següent.

1- Anar a la carpeta: C:\Windows\System32\drivers\etc

2- Modificar l'arxiu "hosts" afegint la direcció IP del ROSNI seguit d'espai i el nom d'usuari de la Raspberry Pi que hi ha instal·lat el ROS del robot. Vegeu la Figura 12.

És recomanable modificar l'arxiu "*hosts*" editant-lo a l'escriptori de l'ordinador i tornar a afegir l'arxiu per tal d'evitar possibles problemes de permisos.

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host
10.42.0.1   raspi
# localhost name resolution is handled within DNS itself.
#     127.0.0.1       localhost
#     ::1             localhost
```

Figura 12. Modificació de l'arxiu *host*.

Cal esmentar que aquest procediment és aplicable a ordinadors amb sistema operatiu Windows, amb sistemes operatius MAC OS o Linux es desconeix si existeix aquest problema.

3.3. Descripció dels tòpics del robot ROSNI

En el flux de missatges d'un entorn ROS, com ja s'ha introduït anteriorment, el tòpic és l'element conductor que permet comunicar entre el publicador i el subscriptor el missatge a tractar. Conèixer quin format de missatge i dades transporta cada tòpic representa un punt de gran importància alhora de saber com tractar-les. A continuació, es detallaran els diferents tòpics del robot ROSNI i quin format de missatge transporta cada un d'aquests, vegeu la Figura 13.

```
/diagnostics
/joint_states
/joy
/joy/set_feedback
/map
/map_metadata
/odom
/robot/cmd_vel
/rosout
/rosout_agg
/scan
/slam_gmapping/entropy
/tf
/tf_static
```

Figura 13. Llistat de *rostopic list*.

- */diagnostics*
  Missatge de tipus *diagnostics_msgs/DiagnosticArray*. Publicació del tòpic des del node */joy_node*. Les dades del missatge descriuen diferents estats de processos que pot executar el robot.

- */joint_states*
  Missatge de tipus *sensor_msgs/JointStates*. Publicació del tòpic des del node */robot_dif_controller* i subscrit per defecte pel node */robot_state_publisher*. Les dades del missatge descriuen diferents la velocitat i posició de les rodes dreta i esquerra del robot.

- */joy*
  Missatge de tipus *sensor_msgs/Joy*. Publicació del tòpic des del node */joy_node* i subscrit per defecte pel node */robot_teleop_joy*. Les dades del missatge descriuen diferents la velocitat i posició de les rodes dreta i esquerra del robot. Visualització de dades del *joystick*, només visibles quan s'utilitza aquest.

- */joy/set_feedback*
  Missatge de tipus *sensor_msgs/JoyFeedbackArray*. No conté cap publicador del tòpic, subscrit per defecte des del node */joy_node*. Amb aquest tòpic no s'observen dades.

- */map*
  Missatge de tipus *nav_msgs/OccupancyGrid*. Publicació del tòpic des del node */slam_gmapping*. Permet obtenir les coordenades per la confecció del mapa de l'entorn del robot.

- */map_metadata*
  Missatge de tipus *nav_msgs/MapMetaData*. Publicació del tòpic des del node */slam_gmapping*. Permet obtenir les coordenades per la confecció del mapa de l'entorn del robot.

- */odom*
  Missatge de tipus *nav_msgs/Odometry*. Publicació del tòpic des del node */robot_dif_controller*. Les dades del missatge s'utilitzaran per a realitzar estudis d'odometria del robot.

- */robot/cmd_vel*
  Missatge de tipus *geometry_msgs/Twist*. Publicació del tòpic des del node */robot_teleop_joy*, subscrit per defecte pel node */robot_dif_controller*. Aquest tòpic permet publicar missatges de velocitat per tal de realitzar moviments lineals i angules amb el robot.

- */rosout*
  Missatge de tipus *rosgraph_msgs/Log*. Publicació del tòpic des de tots els nodes del robot. Aquest tòpic és utilitzat per tancar l'entorn ROS.

- */rosout_agg*
  Missatge de tipus *rosgraph_msgs/Log*. Publicació del tòpic des del node */rosout*. Aquest tòpic és utilitzat per tancar l'entorn ROS.

- */scan*
  Missatge de tipus *sensor_msgs/LaserScan*. Publicació del tòpic des del node */ydlidar_node*, subscrit per defecte des del node /slam_gmapping. Les dades d'aquest missatge s'utilitzen per a realitzar SLAM.

- */slam_gmapping/entropy*
  Missatge de tipus *std_msgs/Float64*. Publicació del tòpic des del node */slam_gmapping*. Dades utilitzades per a realitzar SLAM.

- */tf*
  Missatge de tipus *tf2_msgs/TFMessage*. Publicació del tòpic des dels nodes */robot_state_publisher*, */base_link_to_laser4*, */robot_dif_controller* i */slam_gmapping*. Es troba subscrit per defecte des del node */slam_gmapping*. Les dades d'aquest missatge permeten obtenir les transformades de translació i rotació del robot a temps real.

- */tf_static*
  Missatge de tipus */robot_state_publisher*. Publicació del tòpic des del node */robot_state_publisher*. Es troba subscrit per defecte des del node */slam_gmapping*. Es troba subscrit per defecte des del node */slam_gmapping*. Les dades d'aquest missatge permeten obtenir les transformades de translació i rotació del robot.

3.4. Llibreries Matlab instal·lades

Aquest apartat, es detallen les llibreries que s'han hagut d'instal·lar per tal de desenvolupar les diferents parts del projecte, veure la Figura 14.

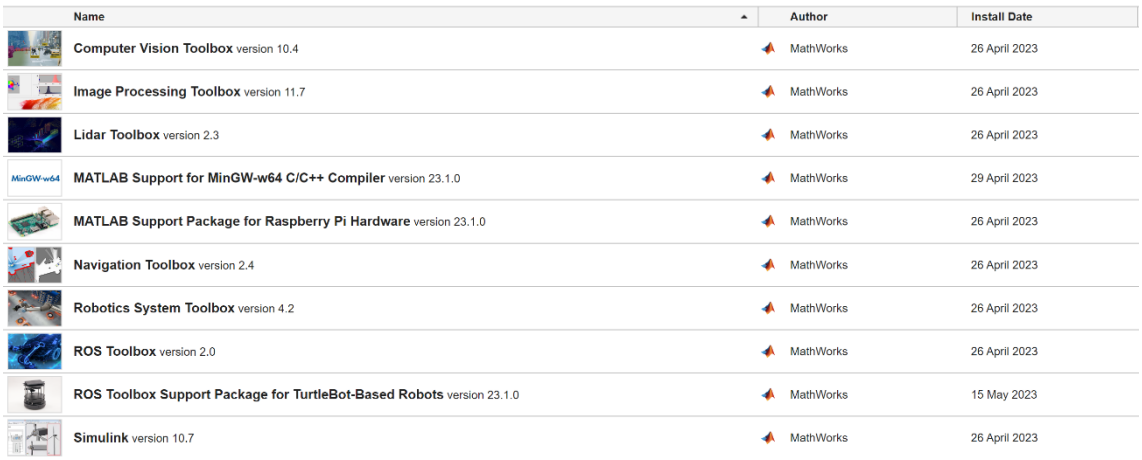


Figura 14. *Toolbox* de Matlab instal·lades.

Per el desenvolupament del projecte, un dels primers passos és conèixer quines funcions tenim disponibles i per a què serveixen alhora de solucionar problemes en els models o codi que s'estigui creant. A continuació, es detallen un seguit de funcions de les llibreries instal·lades i que són d'interès i utilitat per el treball.

3.4.1. Funcions de ROS Toolbox

- *rosinit*
  Inicia la connexió entre Matlab i el ROS Master. El ROS Master és el nucli del sistema ROS que gestiona la comunicació entre nodes, tòpics i serveis. Utilitzant aquesta funció crea una sessió de ROS, que es manté activa mentre Matlab estigui en execució i s'utilitza per connectar-se a altres nodes d'una xarxa ROS.

```
rosinit(URI)
```

- *rosshutdown*
  Desconnecta Matlab i el ROS Master alliberant els recursos utilitzats per la sessió ROS. S'utilitza per tancar correctament la sessió ROS i per prevenir possibles problemes si es vol crear una nova connexió.

```
rosshutdown
```

- *rossubscribe*
  Crea una subscripció a un tòpic de ROS. Quan es crea una subscripció, s'informa al ROS Master que esta esperant rebre missatges d'un tòpic en concret. Quan s'envia un missatge, el ROS Master el reenvia al node de Matlab que ha creat la subscripció mitjançant aquesta funció.

```
sub = rossubscriber(topicname,"DataFormat","struct")
```

- *rospublisher*
  Crea el publicador d'un tòpic de ROS. Quan es crea un publicador, s'informa al ROS Master que el node de Matlab està interessat en publicar missatges en un tòpic determinat. Quan el node de Matlab envia un missatge a aquest tòpic, el ROS Master el reenvia a tots els nodes que s'han subscrit.

```
pub = rospublisher(topicname,Name,Value)
```

- *rosmessage*
  S'utilitza per crear objectes de missatge que es poden enviar o rebre a través de ROS. Cada tipus de missatge té el seu propi format i contingut específic. Amb *rosmessage*, es pot crear objectes de qualsevol tipus de missatge definits a la xarxa ROS.

```
msg = rosmessage(messagetype)
```

- *rosnode*
  Permet crear i configurar nodes ROS des de Matlab. Els nodes són els elements bàsics del sistema ROS i es comuniquen entre ells a través de tòpics i serveis. Amb *rosnode*, es poden configurar les propietats del node i establir la connexió amb altres nodes.

```
rosnode list

rosnode info nodename

rosnode ping nodename
```

- *rostopic*
  Obté la informació sobre els tòpics ROS i manipular-los des de Matlab. Es poden llistar els tòpics disponibles, obtenir informació sobre el tipus de missatge que es transmet, i publicar o subscriure's.

```
rostopic list

rostopic echo topicname

rostopic info topicname

rostopic type topicname
```

- *rosservice*
  Permet crear i utilitzar serveis de ROS des de Matlab. Els serveis són una forma de comunicació entre nodes que permet sol·licitar o proporcionar informació o serveis específics.

```
rosservice list

rosservice info svcname

rosservice type svcname

rosservice uri svcname
```

3.4.2. Funcions de Robotics System Toolbox

- *importrobot*
  S'utilitza per importar un model de robot en format URDF (*Unified Robot Description Format*) des d'un arxiu extern. Un cop importat, el model es converteix en un objecte que es pot utilitzar per realitzar simulacions, planificar trajectòries, dissenyar controladors, entre d'altres.

```
robot = importrobot(URDFtext)
```

- *inverseKinematics*
  S'utilitza per calcular la posició i orientació de les articulacions del robot a partir d'una posició i orientació desitjada. Això es fa utilitzant l'algorisme de la cinemàtica inversa, que és útil per planificar les trajectòries del robot i per controlar les seves posicions.

```
ik = inverseKinematics(Name,Value)
```

- *binaryOccupancyMap*
  Aquesta funció s'utilitza per crear un mapa d'ocupació binari, a partir de les dades del sensor, com ara el LIDAR. Això permet al robot detectar les zones on hi ha obstacles i planificar les trajectòries per evitar-los.

```
map = binaryOccupancyMap(width,height,resolution)
```

- *lidarScan*
  S'utilitza per crear una representació en forma de núvol de punts de les dades del sensor LIDAR. Això és útil per visualitzar l'entorn del robot i detectar els obstacles.

```
scan = lidarScan(ranges,angles)

scan = lidarScan(cart) # Coordenades cartesianes
```

- *plot*
  Permet visualitzar els objectes en un entorn 3D. Això és útil per visualitzar el model del robot, el mapa d'ocupació, les trajectòries planificades, entre d'altres.

```
plot(scanObj,Name,Value)
```

- *transformScan*
  Permet transformar les dades del sensor LIDAR en funció de la posició i orientació del robot.

```
transScan = transformScan(scan,relPose)
```

- *controllerPurePursuit*
  S'utilitza per dissenyar un controlador de seguiment de trajectòries per al robot. El controlador Pure Pursuit és un algorisme de control en bucle tancat que permet al robot seguir una trajectòria predeterminada de manera precisa i eficient.

```
controller = controllerPurePursuit(Name,Value)
```

3.4.3. Funcions de Navigation Toolbox

- *odometryMotionModel*
  S'utilitza per modelar el moviment del robot basat en les dades d'odometria. Això és útil per a la navegació del robot i la planificació de trajectòries.

```
omm = odometryMotionModel
```

- *poseGraph*
  S'utilitza per crear un *graf* que representa les relacions espacials entre les diferents posicions del robot en el temps. Això és útil per la localització del robot i la construcció del mapa.

```
poseGraph = poseGraph('MaxNumEdges',maxEdges,'MaxNumNodes',maxNodes)
```

- *buildMap*
  Aquesta funció s'utilitza per crear un mapa de l'entorn a partir de les dades del sensor. Això és útil per planificar les trajectòries del robot i evitar obstacles.

```
map = buildMap(scans,poses,mapResolution,maxRange)
```

- *getMapData*
  S'utilitza per obtenir les dades del mapa generat per la funció *buildMap*. Permet al robot conèixer les característiques de l'entorn i planificar les trajectòries en funció d'aquestes dades.

```
mapData = getMapData(map)
```

- *addScan*
  Aquesta funció s'utilitza per afegir les dades del sensor a un mapa existent. Això és útil per actualitzar el mapa de l'entorn a mesura que el robot es mou.

```
addScan(slamObj,currScan)
```

- *lidarSLAM*
  Utilitzada per construir un mapa de l'entorn i localitzar simultàniament el robot utilitzant les dades del sensor LIDAR. Això és útil per a la navegació autònoma del robot en entorns desconeguts.

```
slamObj = lidarSLAM(mapResolution,maxLidarRange,maxNumScans)
```

3.5. Inicialitzar el ROS node entre Matlab, Simulink i ROSNI

Per inicialitzar el node ROS entre la Raspberry Pi i el Matlab, o Simulink, s'han de realitzar les indicacions que es presenten a continuació. Hi han dues formes de crear aquest node, segons es treballi amb línies de comanda a través d'un *Script* o la *Command Window* de Matlab, o bé, des de Simulink.

Independentment des d'on s'executin aquestes accions, el primer pas es realitzarà sobre la Raspberry Pi del ROSNI. Un cop es troba engegat el robot, podem procedir a entrar via *ssh* a la Raspberry Pi.

Consultar l'Annex A1 per veure el procediment d'encesa i connexió via *ssh*[10] del robot ROSNI.

Un cop encès el robot i connectats a la Raspberry Pi del robot mitjançant aquest protocol, podrem introduir les dues comandes que necessitem indicar per acabar d'inicialitzar tots els serveis de ROS necessaris.

Primerament, com a comprovació inicial de que ROS s'hagi inicialitzat correctament, es pot escriure la següent comanda a la sessió *ssh* que s'ha inicialitzat:

```
pi@raspi:~$ rostopic list
```

El resultat esperat:

```
/diagnostics

/joint_states

/joy

/joy/set_feedback

/odom

/robot/cmd_vel

/rosout

/rosout_agg

/scan

/tf

/tf_static
```

Després de fer aquesta comprovació, la comanda a introduir és la següent:

[10] https://ca.wikipedia.org/wiki/Secure_Shell

```
pi@raspi:~$ export ROS_IP=10.42.0.1
```

Aquesta comanda serveix per exportar la IP del robot i que pugui ser visible per altres plataformes que vulguin crear un node de comunicació. La IP que s'exporta representarà la IP que s'anomenarà ROS MASTER per a qualsevol altre plataforma amb la qual es vulgui realitzar el node d'enllaç.

Aprofitant la mateixa finestra de comandes oberta amb *ssh*, inicialitzem el servei de ROS indicat amb la següent comanda.

```
pi@raspi:~$ rosservice call /start_slam_nav "slam"
```

Resultat esperat:

```
ok: true
```

Amb aquest servei de ROS inicialitzat podrem veure que ens han aparegut tòpics addicionals que permeten aportar dades necessàries alhora de realitzar SLAM.

Si s'introdueix una altre vegada la comanda *rostopic list*, haurem d'observar la següent llista de tòpics:

```
/diagnostics

/joint_states

/joy

/joy/set_feedback

/map

/map_metadata

/odom

/robot/cmd_vel

/rosout

/rosout_agg

/scan

/slam_gmapping/entropy

/tf

/tf_static
```

Es pot observar que amb la funció utilitzada anteriorment per inicialitzar el servei ROS "slam", aquest ens afegeix 3 tòpics addicionals que aporten dades per a la creació dels mapes amb l'aplicació *Rviz*.

Els passos que s'han detallat s'apliquen a l'inici de la comunicació, cada vegada que s'engega el robot i independentment de si es vol treballar amb la Matlab o Simulink. A continuació, observareu que segons s'esculli treballar amb Matlab o Simulink les accions a realitzar amb el node de comunicació entre aquests entorns i el ROSNI tenen un procediment diferent.

3.5.1. ROS node amb Matlab

El sistema de treball de Matlab és amb línies de comanda, a continuació es detallaran les comandes necessàries per poder realitzar el ROS node correctament. Per inicialitzar aquest node, es pot utilitzar una *Command Window* o bé un *Live Script*.

També, puntualitzar que un cop inicialitzat el ROS node amb el robot ROSNI, no cal tornar-lo a iniciar durant la sessió, excepte si es realitza la comanda d'aturar el servei del ROS node, `rossshutdown`.

La comanda per crear el ROS node és:

```
rosinit("http://10.42.0.1:11311","NodeHost","10.42.0.233");
```

Recorda que has d'estar connectat a la xarxa Wifi del ROSNI corresponent. També, vegeu que la primera IP que s'indica per crear és la IP del robot ROSNI juntament amb el port de comunicació d'aquest i l'altra que s'indica és la IP de l'ordinador de l'usuari amb el que s'estigui connectat a la xarxa i executant el Matlab en aquell instant.

Missatge esperat:

Initializing global node /matlab_global_node_99999 with NodeURI http://COMPUTER _IP/ and MasterURI http://10.42.0.1:11311.

Per comprovar que s'ha realitzat la comunicació correctament, es pot llançar la següent comanda:

```
rostopic list
```

El resultat d'aquesta comanda hauria de ser el següent, compartint el mateix nombre de tòpics que s'ha visualitzat a la terminal de la Raspberry Pi del ROSNI:

```
/diagnostics
/joint_states
/joy
/joy/set_feedback
/map
/map_metadata
/odom
/robot/cmd_vel
/rosout
/rosout_agg
/scan
/slam_gmapping/entropy
/tf
/tf_static
```

Si el resultat d'aquests passos detallats és satisfactori, la comunicació entre Matlab i el ROSNI ja està apunt per a implementar les comandes i exemples que es podran observar amb més detall al següent Capítol 4 d'aquesta memòria.

3.5.2. ROS node amb Simulink

Simulink, amb un concepte de treball diferent al Matlab, té un sistema de funcionament per blocs detallat a l'anterior capítol de l'Estat de l'art d'aquesta memòria. En aquest cas, si només es volen treballar les accions del ROSNI des del Simulink, no es requereix inicialitzar el ROS node de la mateixa forma que l'indicat a l'apartat anterior. Alhora d'inicialitzar un projecte amb blanc, si es defineixen els paràmetres que es detallen a continuació, no es requereix de cap més acció. És a dir, un cop configurat al Simulink, cada cop que s'inicialitzi el model (i existeixi connexió a la xarxa del ROSNI corresponent) no caldrà iniciar el *ROS node* perquè ja existirà l'enllaç per defecte.

Seguidament detallem els passos a seguir per configurar la comunicació entre Simulink i ROSNI.

1- Primerament, s'ha d'indicar que es treballarà amb un entorn ROS. Observant la Figura 15 seleccionarem la opció de *Robot Operating System (ROS)*. Un cop seleccionat, escollirem l'última opció ja que el ROS s'executa sobre una Raspberry Pi del ROSNI, vegeu Figura 16.

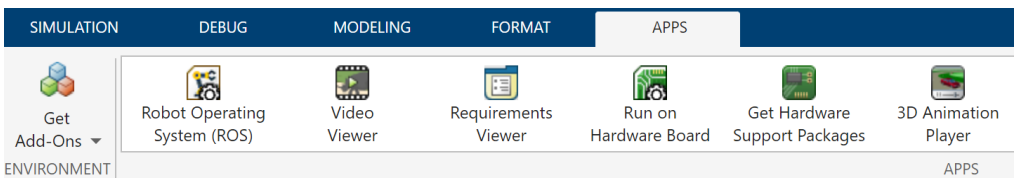


Figura 15. Pas inicial en el model Simulink.

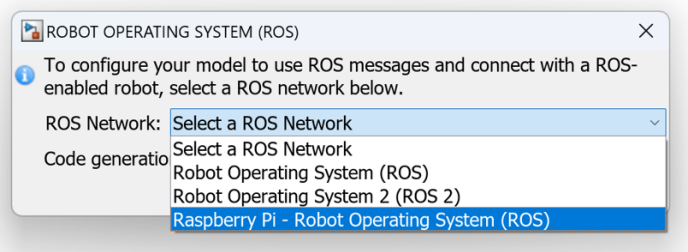


Figura 16. Selecció de la xarxa ROS.

2- Seguidament, haurem d'anar a la pestanya ROS. D'aquesta pestanya s'hauran de configurar les dades de connexió de l'apartat CONNECT < Deploy to < Manage Remote Device. Vegeu la Figura 17 i Figura 18.

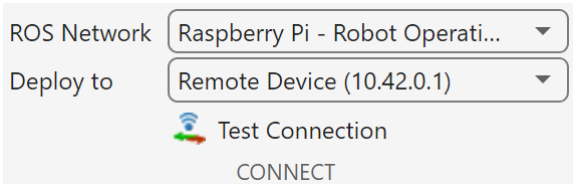


Figura 17. Pestanya per configurar la connexió Simulink-ROS.

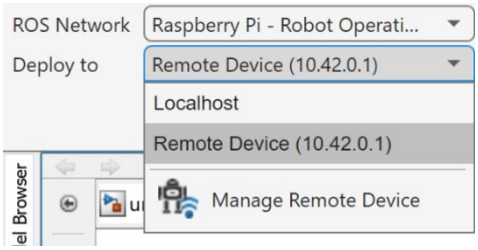


Figura 18. *Manage Remote Device*.

A dins el Manage Remote Device, indicarem la IP del robot ROSNI, el nom d'usuari d'aquest (raspi) i la contrasenya habitual (1234) quan s'inicia una sessió *ssh*. Si observem la Figura 19, també indicarem el directori de treball ROS i el *catkin workspace* existent a la Rasperry Pi del robot.

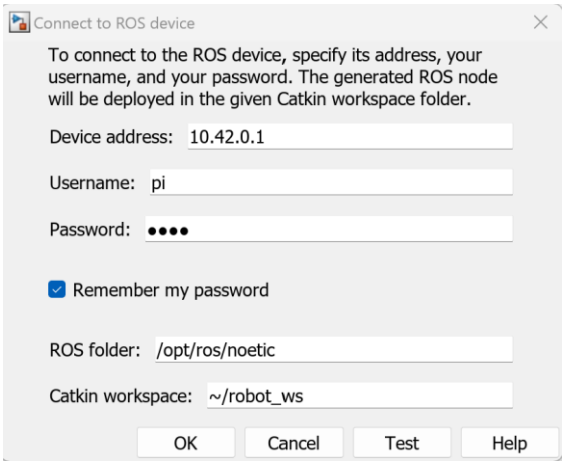


Figura 19. Indicació de les dades del dispositiu ROS.

Per comprovar que hem configurat les dades correctament, podem executar un *Test connection* a través de la icona de la Figura 17 i es necessitaran veure els següents missatges de la Figura 20 a la finestra de diagnòstics:

```
1. Pinging device '10.42.0.1'.
Successfully pinged device.
---
2. Establishing connection to device '10.42.0.1' at SSH port 22. Connecting with username 'pi'.
Successfully connected to device.
---
3. Testing privilege level of user 'pi'.
The user has administrative privileges (sudo requires password). Both single-tasking and multi-tasking models can be deployed to the device.
---
4. Detecting installed ROS distribution.
Checking in folder '/opt/ros/noetic'.
Found ROS distribution 'NOETIC' in folder '/opt/ros/noetic'.
---
5. Verifying Catkin workspace in '~/robot_ws'.
Folder exists.
Folder is writable.
Folder contains a valid Catkin workspace.
---
6. Disconnecting from device '10.42.0.1'.
Done with connection test.
```

Figura 20. Test de connexió.

3.6. Màquina virtual i recursos d'utilitat

En aquest apartat, es detallarà una forma alternativa i addicional per tal de comunicar-se amb el robot ROSNI i poder inicialitzar l'aplicació *Rviz* que pot ser útil alhora de contrastar les dades visualitzades a través de Matlab i Simulink.

Rviz (*Robot Visualization*) és una interfície gràfica clau en el desenvolupament i la depuració de sistemes robòtics, ja que proporciona una representació visual dels diferents components d'un robot o d'un entorn simulat. Amb aquesta aplicació, els desenvolupadors poden monitoritzar i inspeccionar de forma detallada els diversos aspectes d'un sistema robòtic simulat.

Una de les característiques més rellevants de *Rviz* és la seva capacitat per visualitzar models tridimensionals. Això significa que els usuaris poden importar models CAD o generar models virtuals per representar el robot, els sensors, l'entorn i altres objectes relacionats. A més, *Rviz* permet ajustar les propietats dels models, com ara la mida, el color o la transparència, per adaptar-se a les necessitats visuals específiques. Aquestes funcions inclouen la visualització de dades de sensors en temps real, com ara les lectures de càmeres, escàners làser o acceleròmetres. També és possible mostrar informació sobre la planificació de trajectòries, com ara camins planificats, punts de referència o àrees prohibides.

Seguidament, detallarem un procediment substitutiu a l'utilitzat a l'assignatura de Robòtica Mòbil. En aquesta assignatura, que s'imparteix a 4rt curs del grau d'Enginyeria Mecatrònica, es treballa sobre el robot ROSNI comunicant-se a través d'un altre entorn ROS mitjançant UDP (per veure el procediment consultar la documentació de l'assignatura).

En tot cas, l'objectiu d'aquest apartat és detallar la configuració trobada en aquest projecte per a poder aplicar aquesta comunicació d'una forma més simplificada.

Els passos a seguir per a obtenir la màquina virtual són els següents:

1- El primer pas per preparar el sistema operatiu Windows del nostre ordinador és configurar les opcions necessàries per obtenir una màquina virtual en el mateix sistema. Obrirem la finestra de *Característicasde Windows*, vegeu la Figura 21.

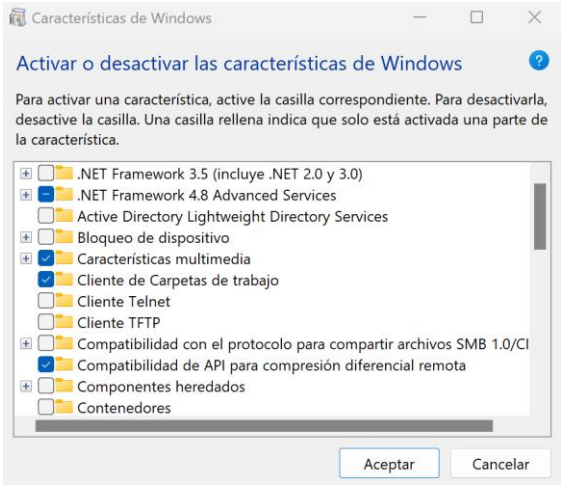


Figura 21. Característiques de Windows.

2- Un cop oberta la finestra de *Característicasde Windows*, activarem les opcions Hyper-V, Plataforma de màquina virtual i Subsistema de Windows para Linux. Podeu veure les característiques marcades amb groc a la Figura 22.

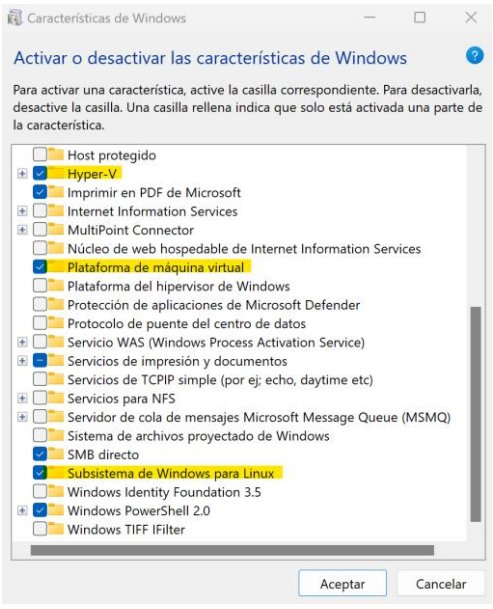


Figura 22. Activació de les característiques de Windows.

3- Amb les característiques detallades anteriorment configurades, el sistema operatiu estarà preparat per instal·lar les dues aplicacions necessàries per obtenir un entorn Linux en concret. Les aplicacions a instal·lar són Windows Subsystem for Linux (vegeu Figura 23) i Ubuntu 20.04.6 LTS (vegeu Figura 24).

Figura 23. Aplicació *WSL*.

Figura 24. OS Ubuntu 20.04

4- Seguidament, amb les dues aplicacions instal·lades, instal·larem l'entorn ROS en el que ens descarregarem els paquets del repositori GitHub i comunicarem amb el ROSNI mitjançant el protocol UDP. La instal·lació i configuració de l'entorn ROS, és a dir el *catkin_ws*, s'ha procedit segons els enllaços detallats a continuació.

http://wiki.ros.org/noetic/Installation/Ubuntu

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

5- Com a darrer pas, descarreguem des del repositori GitHub del ROSNI els paquets mitjançant la següent comanda.

```
ubuntu@DESKTOP-QOQVB0O:~$ git clone https://github.com/ROSNI-
educational-robot/rosni_ros_packages.git
```

Per sincronitzar correctament els paquets descarregats, és necessari realitzar la comanda *catkin_make*. Per més informació bàsica sobre el funcionament de ROS, consultar la documentació ROS Wiki que podeu trobar a la Bibliografia.

3.7. Inicialització de l'aplicació *Rviz*

L'objectiu d'aquest apartat és detallar els passos i l'ordre a seguir alhora d'inicialitzar l'aplicació *Rviz* per a la visualització del robot ROSNI. Per obtenir informació complementària es pot consultar la documentació *ROSNI Assignment* (2022) existent de l'assignatura de Robòtica Mòbil del Grau en Enginyeria Mecatrònica.

1- Primerament, necessitarem exportar el ROS MASTER URI indicant la IP i el port del ROSNI obrint una terminal del sistema operatiu Ubuntu, que ens haguem instal·lat a la màquina virtual, i inserint la següent línia de comanda:

```
ubuntu@DESKTOP-QOQVB0O:~$ export ROS_MASTER_URI=http://10.42.0.1:11311
```

2- Seguidament, si encara no s'ha realitzat l'exportació de la IP del robot i s'ha inicialitzat el servei de ROS referent a SLAM, s'hauran d'inserir a una de les terminals del robot les següents comandes que ja s'han vist anteriorment amb passos per comunicar-se al Matlab:

```
pi@raspi:~$ export ROS_IP=10.42.0.1
```

```
pi@raspi:~$ rosservice call /start_slam_nav "slam"
```

3- Tornant a la terminal d'Ubuntu, anirem al directori de l'espai de treball *catkin_ws* que s'ha creat anteriorment i on hem guardat els paquets del ROSNI descarregats del GitHub. Seguidament realitzarem un *catkin_make*, funció que ens permet administrar les dependències i compilar els paquets ROS assegurant que aquests estiguin ben enllaçats en l'entorn de treball.

```
ubuntu@DESKTOP-QOQVB0O:~$ cd catkin_ws
```

```
ubuntu@DESKTOP-QOQVB0O:~/catkin_ws$ catkin_make
```

4- Com a últim pas, a la mateixa terminal que s'han executat les comandes anteriors, ja estarem preparats per poder inicialitzar l'aplicació *Rviz*. Per tal d'inicialitzar-la utilitzarem la següent comanda:

```
ubuntu@DESKTOP-QOQVB0O:~/catkin_ws$ rosrun rviz rviz
```

4. Resultats i discussió

En aquest capítol, es detallen diversos exemples de Matlab i Simulink que permeten gestionar i controlar les dades del robot ROSNI.

Primerament es presenta un exemple de com visualitzar el model del robot amb 3D a Matlab mitjançant un dels paquets disponibles al repositori GitHub del robot ROSNI. En aquest cas, les dades són preestablertes pels autors del robot, es a dir no es tracten les dades en temps real que pugui estar generant el robot.

Per avaluar el potencial que té la plataforma Matlab i Simulink, també es detallen dos exemples, els dos amb procediments diferents, però aconseguint un resultat final igual en ambdós casos.

La demostració que la plataforma Matlab i Simulink és una eina molt potent per al control i gestió de dades en robòtica, és la base d'aquest capítol. Cal destacar, que davant la gran quantitat de variants que hi pot haver, s'han escollit els exemples més representatius i didàctics per a poder treballar amb el robot ROSNI en un futur.

4.1. Importar model del robot

Conèixer sobre quin model s'està treballant i obtenir la informació necessària és un dels punts més importants per a poder desenvolupar projectes sobre robòtica. A continuació, es detallaran els passos per poder visualitzar el model del robot 3D mitjançant les dades que proporcionen els autors amb un dels paquets que proporcionen els autors en el seu repositori de GitHub del robot ROSNI[11]. Aquesta funció disponible a Matlab és de gran utilitat per a casos on es vulgui saber de quin model de robot es tracta abans de tenir-lo físicament, per exemple.

El primer pas és cercar l'arxiu que conté la informació del robot ROSNI. Aquest arxiu el trobarem al respositori de GitHub del robot ROSNI, vegeu la següent Figura 25.

https://github.com/ROSNI- educational-robot/rosni_ros_packages

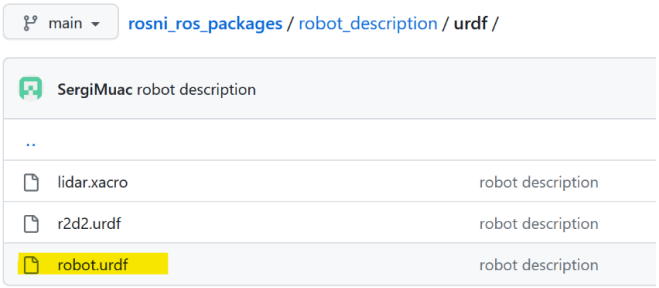


Figura 25. Arxiu *robot.urdf*

[11] https://github.com/ROSNI-educational-robot/rosni_ros_packages

L'arxiu amb la informació del robot, *robot.urdf*, s'insereix a la ubicació de la carpeta d'instal·lació de Matlab, per aquest projecte la ubicació de la carpeta d'arxius és la següent.

C:\Program Files\MATLAB\R2023a\toolbox\robotics\robotmanip\robotModels\roboturdf

Per poder importar el model i el Matlab pugui mostrar el model del robot s'han d'inserir les següents línies de comandes a la *Command Window*, o bé a un *Live Script* si es desitja guardar aquesta acció.

```
robot = importrobot('C:\Users\Usuario\Desktop\rosni_ros_packages-main\robot_description\urdf\robot.urdf');
show(robot)
```

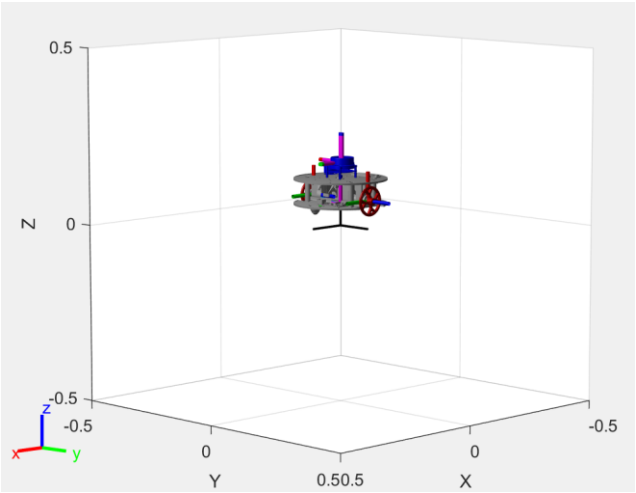


Figura 26. Representació del model *robot.urdf* importat.

A la Figura 26, noteu que aquesta funció requereix indicar la ubicació de l'arxiu *robot.urdf*, si l'arxiu que conté la descripció del robot està ubicat a un altre directori haurem d'indicar la ruta d'aquest.

4.2. Implementació exemple "Explore Basic Behavior of the TurtleBot" (Matlab)

En aquest apartat, s'exposarà com implementar l'exemple "*Explore Basic Behavior of the TurtleBot*" [12] disponible a la documentació de MathWorks. Com ja s'ha comentat anteriorment en aquesta memòria, un dels homòlegs del robot ROSNI és el robot mòbil *TurtleBot*, compartint la majoria de característiques. Aquest fet permet tenir a disposició una gran quantitat d'exemples a implementar amb el ROSNI, tenint en compte que

[12] https://es.mathworks.com/help/ros/ug/explore-basic-behavior-of-the-turtlebot.html

direccions IP i altre puntualitzacions s'han d'adaptar a cada model amb el qual es desenvolupa.

L'objectiu d'aquest exemple és comprovar que adaptant l'exemple disponible a la documentació de MathWorks, és possible visualitzar les dades d'escaneig a través de Matlab d'una manera ràpida i simplificada.

4.2.1. Explicació del codi pas a pas

Inicialment, es crea el node ROS seguint el procediment detallat anteriorment en aquesta memòria. També s'inicialitza un publicador que permetrà enviar missatges de velocitat comunament anomenat Twist, que permetrà dotar de moviment lineal i angular al robot ROSNI.

```
rosinit("http://10.42.0.1:11311","NodeHost","10.42.0.233").

robot = rospublisher("/robot/cmd_vel","DataFormat","struct").
velmsg = rosmessage(robot).
```

Amb el tractament de les dades d'escaneig provinents del sensor LIDAR com a objectiu principal, es crea una subscripció al tòpic */scan* que permetrà obtenir les dades necessàries. També podem observar l'estructura de dades que conté la variable *scan* creada en el següent codi.

```
laser = rossubscriber("/scan","DataFormat","struct").
scan = receive(laser)


figure
rosPlot(scan).
```

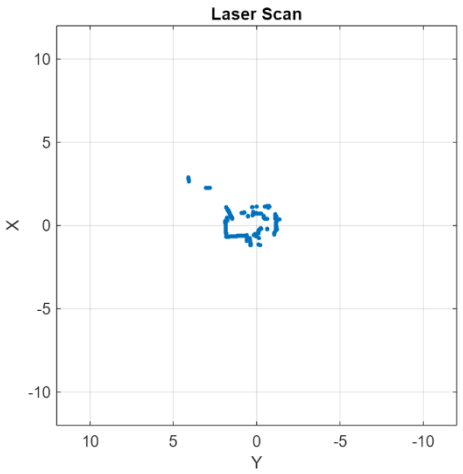


Figura 27. Gràfic *Laser Scan 1.*

A la Figura 27, observem el primer gràfic representat amb els punts obtinguts en un instant de temps a través del tòpic `/scan`. En tot cas, aquest primer gràfic, com a usuaris no aporta una informació actualitzada, al mostrar la informació d'un instant de temps passat i on l'entorn ja ha canviat. Per obtenir la informació actualitzada de l'escaneig afegirem una funció `while` que durant 10 segons, s'actualitzaran les dades de l'escaneig i es mostraran mitjançant el gràfic de la Figura 28.

```
tic.
while toc < 10
  scan = receive(laser,3).
  rosPlot(scan).
end
```

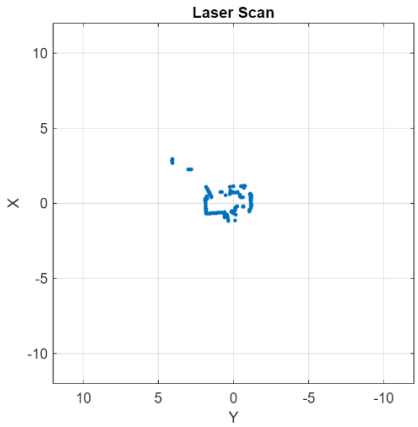


Figura 28. Gràfic *Laser Scan 2*.

Per completar aquest exemple, en aquesta part de codi es dota de moviment al robot i es tracten les dades d'escaneig durant 20 segons, temps que pot ser modificat segons cas. L'objectiu final és obtenir el control del robot detectant els obstacles, indicant una distància mínima a avaluar i actuar conseqüentment segons aquesta distancia envers l'obstacle més pròxim.

Per l'escaneig de dades, en aquest apartat s'hi afegeix la funció `rosReadCartesian(scan)` pel tractament de dades que encara no s'havia utilitzat a l'exemple. Aquesta funció converteix les coordenades polars dels punts de l'escàner a coordenades cartesianes.

Seguidament, es calcula la distància dels obstacles més propers utilitzant la fórmula de la distància euclidiana. Es calcula la distància per a cada punt detectat i utilitzant la mínima d'aquestes.

Segons el valor mínim de la distància respecte al llindar de distància definit amb la variable `distanceThreshold`, es defineixen les diferents accions per controlar el robot:

- Si la distància mínima és inferior al llindar de la distància definida, significa que el robot està a prop d'un obstacle. En aquest cas, s'estableix la velocitat angular `velmsg.Angular.Z` com a `spinVelocity` per girar el robot i la velocitat lineal `velmsg.Linear.X` com a `backwardVelocity` perquè el robot es mogui enrere lleugerament i eviti l'obstacle.
- Si la distància mínima és igual o superior al llindar de distància, el robot continua amb la seva direcció actual i sense girar. S'estableix la velocitat lineal `velmsg.Linear.X` com a `forwardVelocity` i la velocitat angular `velmsg.Angular.Z` com a zero.

Durant l'execució d'aquesta funció `while`, en la que s'ha definit que dura 20 segons, s'imprimeix a la pantalla de forma actualitzada el gràfic de l'escaneig on s'observa segons el moviment que realitza el robot tal com es representa a la Figura 29.

```
spinVelocity = 0.6.       % Angular velocity (rad/s)
forwardVelocity = 0.1.    % Linear velocity (m/s)
backwardVelocity = -0.02. % Linear velocity (reverse) (m/s)
distanceThreshold = 0.6.  % Distance threshold (m) for turning

  tic.
  while toc < 20

      % Collect information from laser scan
      scan = receive(laser).
      rosPlot(scan).
      data = rosReadCartesian(scan).
      x = data(:,1).
      y = data(:,2).
      % Compute distance of the closest obstacle
      dist = sqrt(x.^2 + y.^2).
      minDist = min(dist).
      % Command robot action


      if minDist < distanceThreshold

          % If close to obstacle, back up slightly and spin
          velmsg.Angular.Z = spinVelocity.
          velmsg.Linear.X = backwardVelocity.
      else

          % Continue on forward path
          velmsg.Linear.X = forwardVelocity.
          velmsg.Angular.Z = 0.
      end

      send(robot,velmsg).

end
```

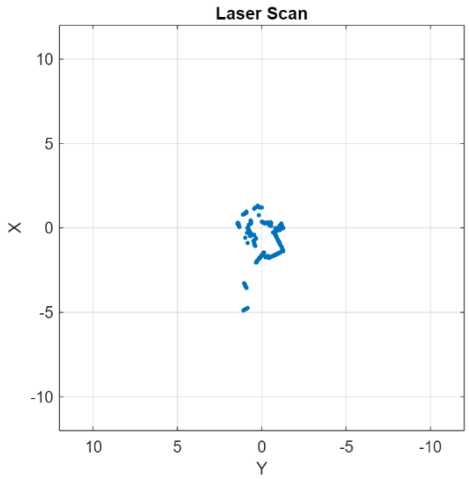


Figura 29. Gràfic *Laser Scan 3.*

4.3. Implementació exemple "Plot TurtleBot Odom" (Matlab)

La implementació de l'exemple *"Plot TurtleBot Odom"* de Matlab conté la mateixa estructura que l'exemple anterior. En aquest cas, l'exemple permet obtenir les coordenades del robot i mostrar el gràfic de la trajectòria guardada, permetent així, veure la traçada que ha realitzat el robot.

4.3.1. Explicació del codi pas a pas

En aquest exemple es pot observar una altra forma de connectar-se al ROSNI, ja que la funció `turtlebot` de la llibreria *ROS Toolbox Support Package for TurtleBot-Based Robots* permet realitzar la connexió més simplificada, com observarem a continuació.

La primera línia de codi defineix la variable `ipaddress` amb l'adreça IP del teu robot, que en aquest cas és '10.42.0.1'.

A la següent línea es crea un objecte `turtlebot` indicant l'adreça IP i el port 11311 de la Raspberry Pi del ROSNI. Aquest objecte l'anomenem `rosni` i s'utilitzarà per interactuar amb el nostra robot.

S'estableix el nom del tòpic per a la velocitat del robot mitjançant la propietat `Velocity.TopicName`, amb el qual definirem missatge de velocitats a través del tòpic */robot/cmd_vel*.

```
ipaddress = '10.42.0.1'. % IP address of your robot
rosni = turtlebot(ipaddress,11311).
rosni.Velocity.TopicName = '/robot/cmd_vel'.
```

A continuació, s'obté la odometria del robot utilitzant la funció `getOdometry` i s'emmagatzema a la variable `odom`. Després, per assegurar que es comença des de zero, reinicialitza la odometria utilitzant la funció `resetOdometry`.

Seguidament, s'estableix la velocitat lineal del robot a 0.2 durant 2 segons utilitzant la funció `setVelocity` amb els paràmetres *Time* i *Duration*. Després d'això, s'obté de nou la odometria del robot i s'emmagatzema a la variable `odom`.

```
odom = getOdometry(rosni)
resetOdometry(rosni)
setVelocity(rosni,0.2,'Time',2)
odom = getOdometry(rosni)
```

El següent pas, prèviament a inicialitzar el bucle `for` de captura de dades, s'inicialitza una matriu buida `odomList` de mida 20x2 per emmagatzemar les posicions del robot. Tornem a reiniciar la odometria del robot utilitzant `resetOdometry`.

A continuació, s'executa el bucle `for` que itera de l'índex 1 al 20. A cada iteració, s'obté l'odometria del robot i s'emmagatzema a la variable `odom`. Les coordenades x i y de la posició actual s'afegeixen a la matriu `odomList`.

Durant l'execució del bucle, avaluem el nombre d'iteracions que s'han dut a terme. S'avalua en termes que quan l'índex de la iteració és menor a 10, es defineix la velocitat del robot a 0.2 i un gir de 1.5 utilitzant la funció `setVelocity`. Si el nombre d'iteracions és major a 10 la velocitat serà de 0.2 i un gir de -1.5, és a dir, el robot girarà al revés. Entre cada iteració, s'espera 1 segon utilitzant la funció `pause`.

Un cop finalitzat el bucle, es defineix la velocitat del robot a 0 per aturar-lo utilitzant `setVelocity(rosni,0,0)`.

Finalment, es realitza una representació gràfica de les posicions guardades a `odomList` utilitzant la funció `plot`. S'estableixen els límits dels eixos x i y utilitzant `xlim` i `ylim`, respectivament, per tal de centrar la trajectòria representada amb la zona específica de l'entorn on s'ha mogut el robot, vegeu la Figura 30.

```
odomList = zeros(20,2);
resetOdometry(rosni)


for i = 1:20
    odom = getOdometry(rosni);
    odomList(i,:) = [odom.Position(1) odom.Position(2)];

    if i < 10
        setVelocity(rosni,0.2,1.5)
    else
```

```
        setVelocity(rosni,0.2,-1.5)
    end
    pause(1);
end
setVelocity(rosni,0,0)
plot(odomList(:,1),odomList(:,2))


xlim([-2 2])
ylim([-2 2])
```

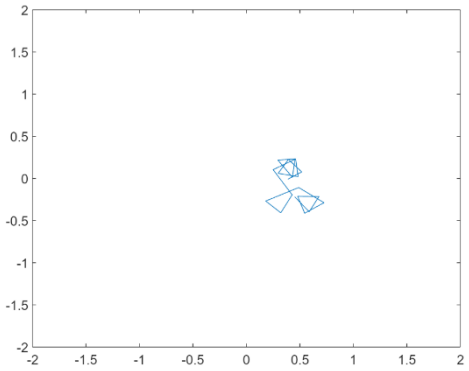


Figura 30. Representació de la trajectòria del ROSNI

4.4. Escaneig i moviment del ROSNI mitjançant Simulink

L'objectiu d'aquesta implementació és crear un model simple que permet moure el robot i visualitzar diferents dades del robot, que ja s'han tractat amb els exemples anteriors, però en aquest cas mitjançant els blocs disponibles que proporcionen les llibreries detallades en el capítol 2 de l'Estat de l'art d'aquesta memòria.

La documentació existent de MathWorks referent a models d'exemple de Simulink no és d'aplicació tant directe com els exemples que s'hagin pogut veure anteriorment. En conseqüència, en aquest apartat s'ha realitzat una adaptació a un model creat amb informació obtinguda de les documentacions de MathWorks i que pugui representar les funcions i capacitats d'aquesta eina.

Seguidament, detallem un exemple creat que pretén ser un model introductori per familiaritzar-nos en treballar amb l'entorn de blocs que ens proporciona el Simulink.

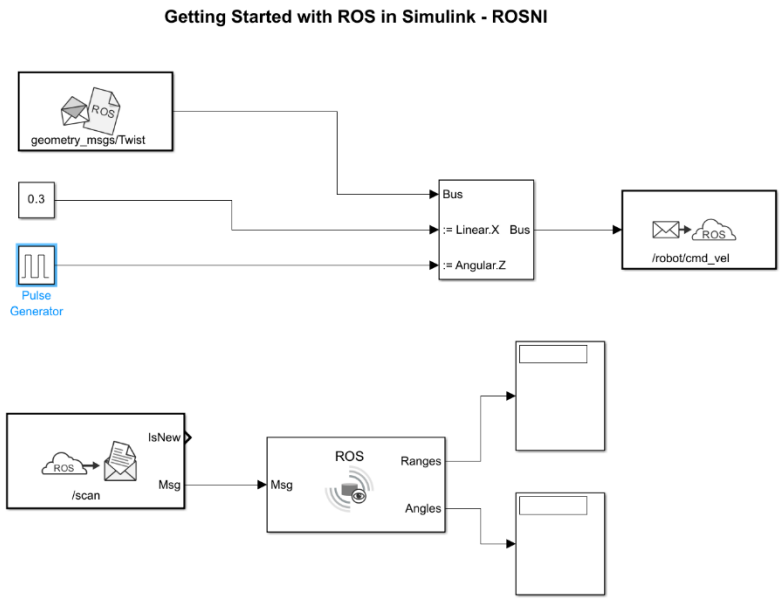


Figura 31. Visió general del model introductori a Simulink.

Observant la Figura 31, la implementació realitzada en aquest model es basa en practicar la dinàmica del flux de dades entre Simulink i ROS, creant un publicador i subscriptor utilitzant dos dels tòpics, `/scan` i `/robot/cmd_vel`, que ja s'han treballat en exemples anteriors.

Primerament, si ens centrem en els blocs del publicador, vegeu Figura 32, observem que amb el bloc *Blank Message*, Figura 33, es relaciona amb el format de missatge a enviar, `geometry_msgs/Twist`, que permet dotar al robot de les velocitats necessàries per dotar de moviment al robot.

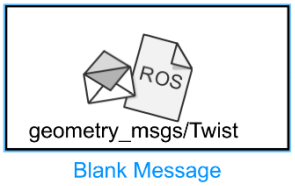


Figura 32. Bloc *ROS Blank Message*.

Mitjançant un bloc *Bus Assignment*, vegeu Figura 33, realitzarem la configuració per poder crear dues entrades que representaran la velocitat lineal i angular del robot, amb una sola sortida de *Bus Assignment*. A les entrades configurades al *Bus* s'hi afegeixen constants o altres senyals per definir les velocitats, i conseqüentment influeixen en el comportament del robot.

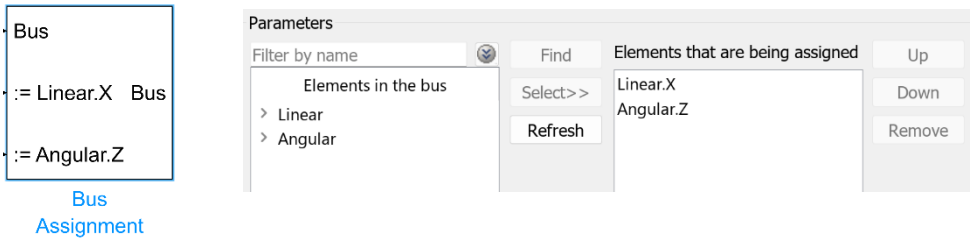


Figura 33. Bloc Bus Assignment.

Com a bloc final, configurem un bloc *ROS Publish*, vegeu Figura 34, per a poder enviar el missatge provinent del *Bus* a través del tòpic `/robot/cmd_vel`.

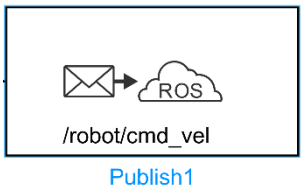


Figura 34. Bloc *Ros Publish*.

Com a segona part del model, si tornem a observar la Figura 31 ens centrem en comentar els blocs que ens permeten obtenir les dades d'escaneig provinents del tòpic `/scan`. En aquest cas, el primer bloc a enllaçar és el bloc *ROS Subscribe*, vegeu la Figura 35, i al que s'hi defineix el nom exacte del tòpic del que volem obtenir les dades.

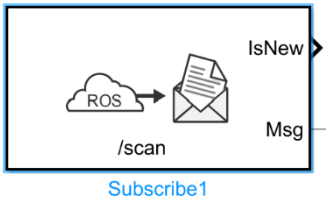


Figura 35. Bloc *ROS Subscribe*.

Per poder recollir les dades directament i mostra-les, el Simulink ens dota d'un bloc que realitza la funció d'obtenir les dades i seleccionar els *Ranges* i *Angles* útils per l'escaneig. El bloc que permet realitzar aquesta funció és el *ROS Read Scan* vegeu Figura 36.

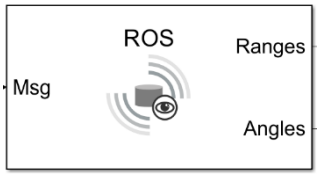


Figura 36. Bloc *ROS Read Scan*.

Un cop enllaçats els blocs de la Figura 35 i 36, connectant dos Displays a les sortides del bloc *ROS Read Scan* ens permet observar les dades obtingudes.

4.5. Implementació exemple "*RobotController.slx*" de Simulink

Avançant un pas més amb els coneixements del control del robot ROSNI mitjançant el Simulink, és destacable detallar l'exemple *RobotController.slx* ja que és un exemple significativament complert i consolida els coneixements de l'apartat anterior, vegeu la Figura 36.

Aquest exemple es basa en dotar de moviment al robot utilitzant el model primer model de la Figura 30. A través de la lectura de l'odometria del tòpic `/odom` provinent de l'entorn ROS, en aquest cas del ROSNI, obtindrem les coordenades del robot. Amb aquestes coordenades, en definirem unes altres, en aquest cas les desitjades per l'usuari i que es convertiran en l'objectiu del robot.

Amb un control PID com a bloc central, tractarem les coordenades i velocitats en tot moment per tal de que el robot es desplaci fins a la posició desitjada i un cop complert l'objectiu aquest s'aturi.

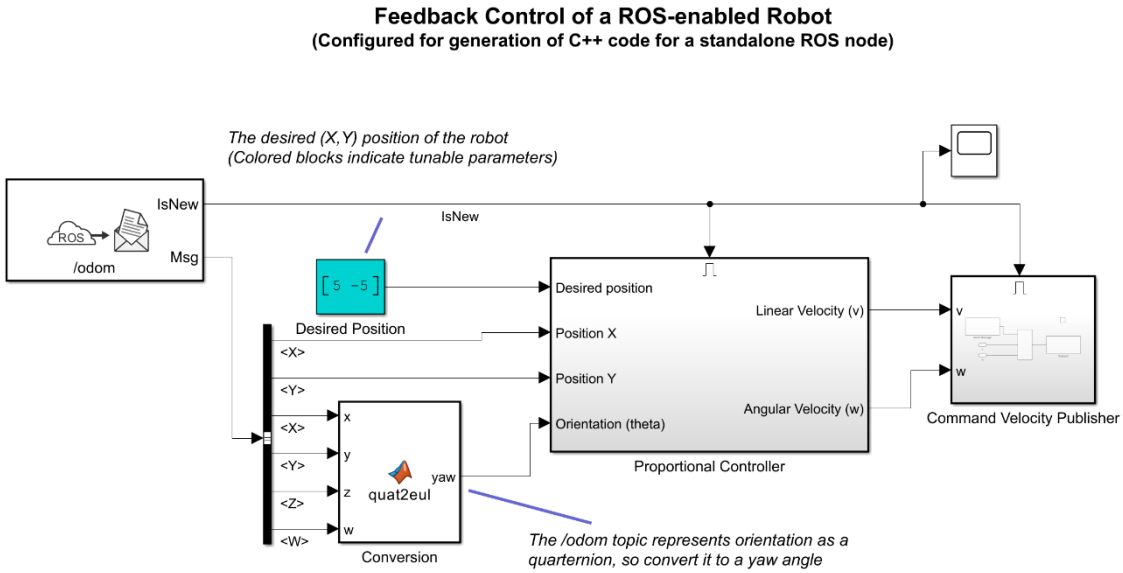


Figura 37. Visió general de l'exemple *RobotController.slx*.

El primer bloc que avaluem és el bloc ROS Subscribe que trobem a l'esquerra de tot de la Figura 37. Aquest bloc permet establir una subscripció del tòpic `/odom` enviat pel ROSNI.

El tòpic `/odom`, recordem, s'utilitza per publicar les dades d'odometria que descriu el moviment del robot en termes de posició i orientació en l'espai. Cada vegada que es

publiquen dades d'odometria en el tòpic `/odom`, el bloc ROS Subscribe les captura i actualitza les sortides associades. La seva configuració resulta segons s'observa a la Figura 38.

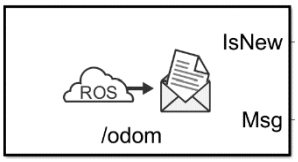


Figura 38. Suscripció al tòpic */odom* en Simulink.

La següent funció a comentar, és el bloc de Matlab Function del Simulink que converteix un quaternió, les dades que es troben en el tòpic `/odom`, en un angle de gir (yaw) en format d'àngles d'Euler (zyx). Si vegeu la Figura 38, la funció `quat2eul` té cinc paràmetres d'entrada: `x`, `y`, `z` i `w`, que representen els components del quaternió.

A continuació, es declara una variable `eul` que emmagatzema els angles d'Euler calculats a partir del quaternió. La funció `quat2eul` és cridada amb els components del quaternió com a paràmetres.

Finalment, s'assigna el valor de l'angle de gir (`yaw`) com a primer element de la variable `eul`.


```
function yaw  = quat2eul(x,y,z,w)
%#codegen

% Convert the quaternion to a yaw angle (zyx euler angle)
eul = quat2eul([w,x,y,z]);
yaw = eul(1);

end
```

Figura 39. Conversió de coordenades amb *Matlab Fuction*.

El bloc "Proportional Controller", de la Figura 40, és la part més complexa de l'exemple. Aquest bloc implementa un controlador proporcional per a robots diferencials i s'utilitza per generar una senyal de control basada en l'error entre una referència desitjada i una variable de retroacció.

Si observem les entrades del bloc, conté un total de 4 entrades que les podem separar en tres canals diferents. A la entrada de Desired position, tal com indica el nom, amb aquesta entrada hi indiquem les coordenades amb x i y que desitgem que el robot es desplaci. Després, observem les entrades *Position X* i *Position Y*, aquestes estan enllaçades amb les dades de posició lineal que provenen directament d'un *Bus Assignme*nt connectat a les dades del tòpic `/odom`. Com a tercer canal de dades, s'enllacen les dades d'orientació del robot tractades amb el bloc de la Figura 38.

La funció bàsica del controlador proporcional és comparar i Calcular la diferència entre la ubicació actual i la ubicació desitjada. Si la distància és inferior a un llindar, atura el robot. Si la distància és superior al llindar, estableix l'angle de gir per orientar el robot cap a la ubicació desitjada i avança cap a aquesta.

Amb els blocs de color turquesa que s'observen a la Figura 40, podem modificar el llindar d'error de la distància, la velocitat lineal del robot mitjançant una constant i aplicant un guany si s'escau, i regular el guany que influeix en la velocitat angular de sortida.

Les sortides del bloc són les velocitats lineals i angular resultants del controlador. Aquestes sortides estan prèviament enllaçades amb dos *Swicth* que permeten modificar a 0 les velocitats en cas que el robot s'hagi desplaçat a les coordenades desitjades.

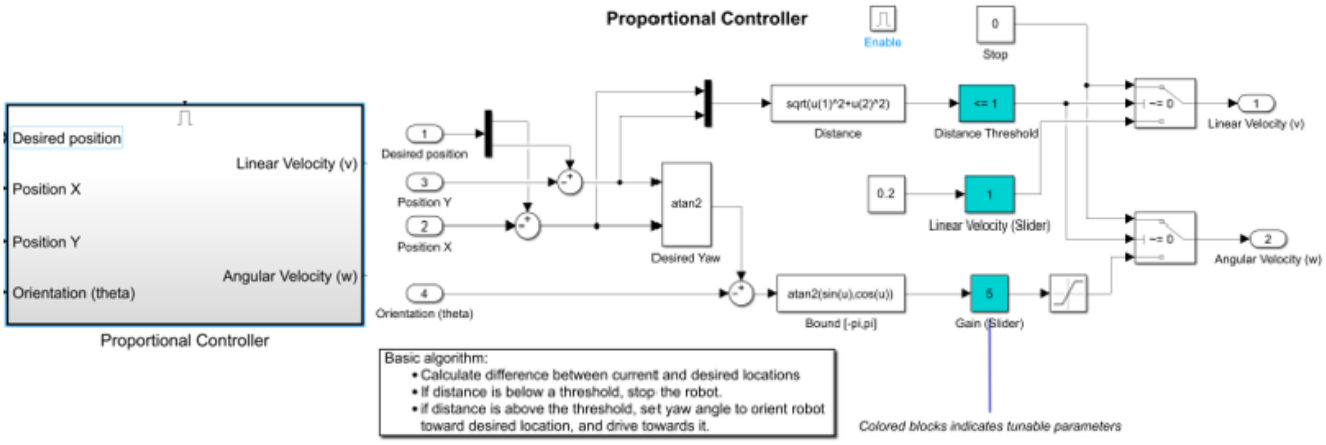


Figura 40. Vista general del *Propotional Controller*.

Per concloure l'exemple, observant la part final de l'exemple amb la Figura 41, vegem que a través de les velocitats 1 i 2 definides amb les sortides del controlador de la Figura 39, i utilitzant el mateix sistema comentat anteriorment amb la Figura 31, generem el missatge a través del tòpic `/robot/cmd_vel` per dotar de moviment al robot.

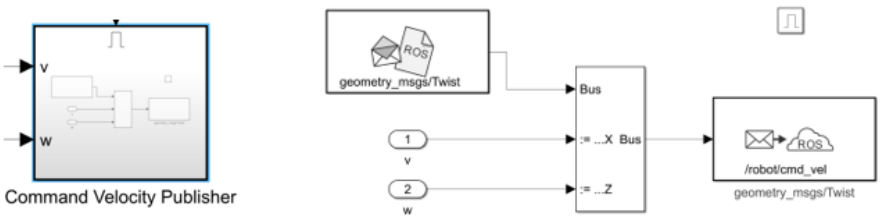


Figura 41. Publicador de velocitats a través del controlador.

5. Conclusió

Com a conclusió final d'aquest projecte, si valorem l'objectiu principal, crec que s'ha assolit d'una forma satisfactòria ja que utilitzant les eines proporcionades per la Universitat de Vic s'ha pogut acabar realitzant una comunicació entre plataformes, captant dades i iniciant-nos amb el tractament d'aquestes. Podem afirmar que, s'ha complert l'objectiu d'una forma eficaç ja que s'ha aconseguit crear un node entre les plataformes de tres formes diferents, tal i com es detalla als Capítols 3 i 4, controlar els moviments del robot mitjançant Matlab i Simulink i modelar parts de codi per a la implementació de les accions sobre el ROSNI.

La gran quantitat de documentació i possibilitats de Matlab i Simulink alhora de desenvolupar models relacionats amb ROS i la robòtica, ha set un dels factors a valorar amb més èmfasi, amb la dificultat de saber acotar bé els límits de desenvolupament del projecte.

Tot i aquestes dificultats, crec que la documentació seleccionada serà de gran utilitat per a la Universitat de Vic i aplicable a les diferents assignatures i projectes que es puguin desenvolupar en un futur a la facultat.

Sobre els objectius específics, s'han assolit amb l'excepció d'un d'aquests. Si ens centrem amb l'objectiu de seleccionar la informació referent a la comunicació entre plataformes, aquest s'ha complert ja que a més de detallar la informació necessària s'han afegit puntualitzacions que s'han trobat com a millora d'aquesta. Alhora de seleccionar els exemples i detallar un procediment de configuració didàctics també s'han assolit els objectius, ja que poden bons exemples per utilitzar-se a l'assignatura de Robòtica Mòbil del Grau. El punt que no s'ha pogut completar, és assolir el detall de totes les eines per poder realitzar qualsevol implementació ja que al haver-hi tanta documentació i possibilitats de generar codi hi ha molts aspectes que no s'han pogut testejar i documentar.

Per concloure, crec que després d'haver desenvolupat aquest treball, podem argumentar que l'entorn Matlab i Simulink és una eina amb un gran potencial i d'aplicació en l'àmbit de la robòtica.

I personalment, la feina realitzada crec que ha set bona i enriquidora, ja que m'ha permès consolidar coneixements sobre la comunicació entre plataformes i practicar en termes de programació i models de Matlab.

5.1. Limitacions i millores a realitzar en projectes futurs

Les limitacions més importants experimentades durant el desenvolupament del projecte han sigut la poca experiència tinguda inicialment en desenvolupar entorns ROS i Matlab. Sobretot en coneixements de ROS, els quals es tenia la base mínima, obtinguda gràcies a l'assignatura de Robòtica Mòbil del Grau en Enginyeria Mecatrònica. A un altre nivell, però, també s'ha requerit llegir molta documentació de Matlab i funcions d'aquesta plataforma que es desconeixien inicialment. Per altra banda, configuracions que no estaven detallades a la documentació del robot ROSNI han afectat a agilitzar el desenvolupament inicial del projecte.

Les millores que poden sorgir en un futur sobre el hardware poden afectar amb alguns aspectes treballats en aquest projecte, per tant, cal tenir en compte quines poden ser aquestes millores. Si ens centrem amb el ROSNI com a hardware, una de les futures millores ha de ser acabar d'implementar la navegació autònoma del robot. Aquesta millora permetrà obtenir més opcions de treball envers els temes estudiats.

Per altra banda, el pas del temps comportarà l'adaptació al sistema operatiu i les funcions de ROS2 ja que recentment s'ha produït la finalització del desenvolupament amb noves versions de ROS, les quals a partir d'ara aquestes només seran desenvolupades amb ROS2.

Sobre les millores a realitzar en futurs projectes, crec que s'haurien d'enfocar principalment en desenvolupar un model complert amb Simulink, o Matlab, i que pugui realitzar diferents funcions treballades en aquest projecte. A més, crec que seria interessant poder acabar de generar mapes de l'entorn i un model PID potent pel control del robot.

6. Bibliografia i webgrafia

Coulter, R. (1990). *Implementation of the Pure Pursuit Path Tracking Algorithm* (Report No. CMU-RI-TR-90-01). Carnegie Mellon University, Pittsburgh, Pennsylvania.

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2023). *Real-Time Loop Closure in 2D LIDAR SLAM*. In 2016 IEEE International Conference on Robotics and Automation (ICRA).

De las Muelas Acosta, S., & Sala Codina, M. (2022). ROS Educational Robot Kit.

Olivares Alarcos, A. (2022). ROSNI Assignment.

Wiki, ROS. (2023). Documentació de ROS Topic. Recuperat de *http://wiki.ros.org/rostopic*

Wiki, ROS. (2023). Tutorial de configuració de l'entorn ROS. Recuperat de *http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment*

Wiki, ROS. (2023). Tutorials de ROS. Recuperat de *http://wiki.ros.org/ROS/Tutorials*

Wiki, ROS. (2023). Instal·lació d'Ubuntu de ROS Noetic. Recuperat de *http://wiki.ros.org/noetic/Installation/Ubuntu*

MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/help/ros/index.html?s_tid=CRUX_lftnav*

MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/help/instrument/tcpclient.html*

MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/help/nav/ug/localize-turtlebot-using-monte-carlo-localization.html*

MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/hardware-support/raspberry-pi-matlab.html*

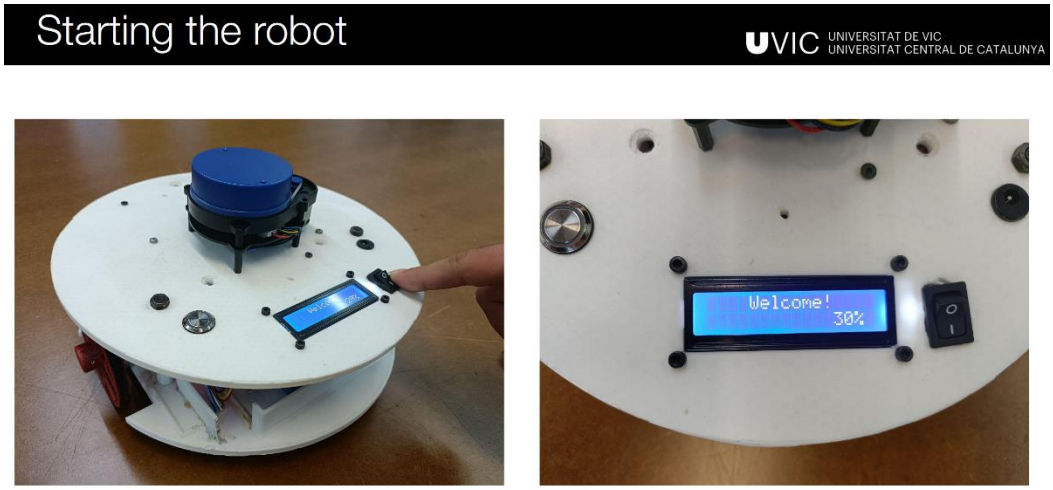
MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/help/ros/ug/generate-a-standalone-ros-node-from-simulink.html*

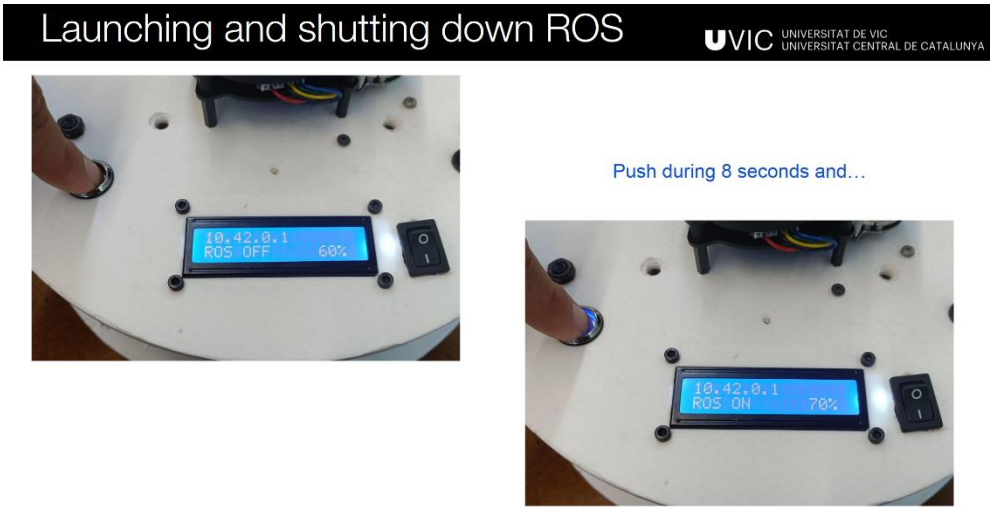
MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/help/ros/ug/explore-basic-behavior-of-the-turtlebot.html*

MathWorks. (2023). Documentació de MATLAB. Recuperat de *https://es.mathworks.com/help/supportpkg/turtlebotrobot/ug/plot-turtlebot-odometry.html*

Annex A

A.1. Encesa i connexió amb *ssh* del robot ROSNI





Hotspot connection

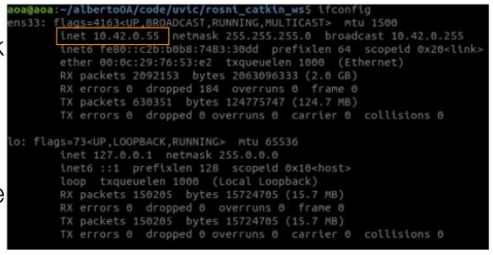
UVIC UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL DE CATALUNYA

ROSNI launches a hotspot so that you can connect to it. First, you should connect to the network *'Rosni-0X'* (X will be the ID of your robot, use password: 12345678).

In order to check that we are actually connected to the robot's network, we will check if we are getting an IP from it, in the ubuntu terminal write the following:

```
$ ifconfig
```

The result should look like the image, check the value of your IP. If it is not similar to the robot's IP, you might need to go to the VM menu 'Virtual Machine > Network Adapter' and select 'Bridged'. Wait some time before using 'ifconfig' again.



Podem connectar-nos al robot utilitzant ssh, el qual ens donarà accés al terminal ubuntu del robot.

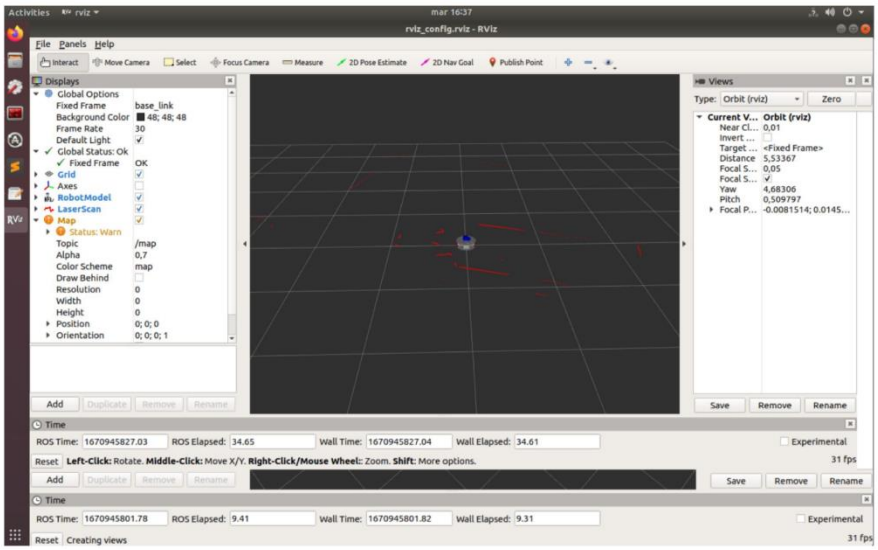
```
$ ssh pi@10.42.0.1
```

La contrasenya que se us demanarà introduir és: 1234. Això és tot, proveu de veure els rostopics o rosservices que s'estan executant al robot.

A.2. Visualització de l'aplicació Rviz

Personalized rviz configuration

UVIC UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL DE CATALUNYA



Visualizing SLAM

UVIC UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL DE CATALUNYA