GRADO EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Navegación Autónoma de un Vehículo en un Entorno Limitado

Autor: Javier González del Campo Artesero

Director: Juan Luis Zamora Macho

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Navegación Autónoma de un Vehículo en un Entorno Limitado

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

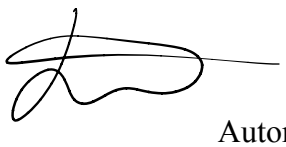
curso académico 2019/20 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Javier González del Campo Artesero Fecha: 27/ 08/ 2020



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Juan Luis Zamora Macho Fecha: 27/ 08/ 2020

GRADO EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Navegación Autónoma de un Vehículo en un Entorno Limitado

Autor: Javier González del Campo Artesero

Director: Juan Luis Zamora Macho

Madrid

A mi familia y amigos

Agradecimientos

Mi más sincero agradecimiento a Juan Luis Zamora Macho, director del proyecto, por su ayuda y dedicación durante todo el trabajo. Gracias por estar siempre dispuesto a ayudar y buscar huecos para reunirse conmigo incluso en vacaciones. Gracias de corazón.

Agradecer también a mi familia y amigos, por todo el apoyo recibido durante los años de carrera, y por mantenerme a flote, incluso en los momentos más difíciles. Una parte de este proyecto también es vuestra.

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO EN UN ENTORNO LIMITADO

Autor: González del Campo Artesero, Javier.
Director: Zamora Macho, Juan Luis.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El objetivo de este proyecto es la instrumentación electrónica de medida y actuación, y la adaptación del software de control de un vehículo para navegación autónoma en un entorno limitado. Para ello, se han implementado con éxito cuatro motores operados a través de dos tarjetas controladoras, una IMU, un LIDAR, unos pulsadores, una Raspberry Pi como ordenador de a bordo y una batería LiPo como única fuente de alimentación del vehículo.

Palabras clave: Vehículo autónomo, navegación autónoma, LIDAR, cámaras, MD25, IMU, MPU-6000, ROS, MATLAB, Simulink, Raspberry, driver.

1. Introducción

La industria de la robótica y la automática está en pleno auge, los procesos de producción en las fábricas se encuentran cada año más automatizados; y el avance en terrenos como el *Internet of Things* (*IoT*) o la Inteligencia Artificial se encuentran superando barreras que hace unos años eran inimaginables. El mundo se encuentra actualmente sumergido en una Cuarta Revolución Industrial, término que fue acuñado por Klaus Schwab, fundador del Foro Económico Mundial, en la edición de 2016 de este mismo Foro. Esta nueva Revolución se caracteriza por ser una fusión de tecnologías que busca difuminar las líneas entre lo físico, lo biológico y lo digital; marcada por avances tecnológicos en campos como la robótica, la Inteligencia Artificial, la nanotecnología, la computación cuántica, el IoT, la biotecnología, y los vehículos autónomos, entre otros.

En este ámbito, este proyecto se enmarca en la tarea de facilitar la instrumentación de vehículos de tamaño reducido, combinando entornos de software tan diferentes como Matlab – Simulink y ROS (*Robot Operating System*); facilitando así el estudio de diferentes algoritmos y procesos de Inteligencia Artificial que ayuden al desarrollo del campo del vehículo autónomo.

2. Definición del proyecto

El Proyecto presentado a continuación puede dividirse en los siguientes objetivos:

- Diseño y construcción del vehículo.
- Integración de la electrónica de medida y actuación del vehículo a través de drivers.
- Integración de los entornos software de Matlab – Simulink y ROS.

3. Descripción general del hardware del vehículo

Los diferentes componentes hardware integrados en el vehículo del proyecto se encuentran representados en la Ilustración 1; en la que se muestra un esquema de las conexiones de tensión (en rojo) y de transmisión de datos (en azul) entre los diferentes elementos hardware.

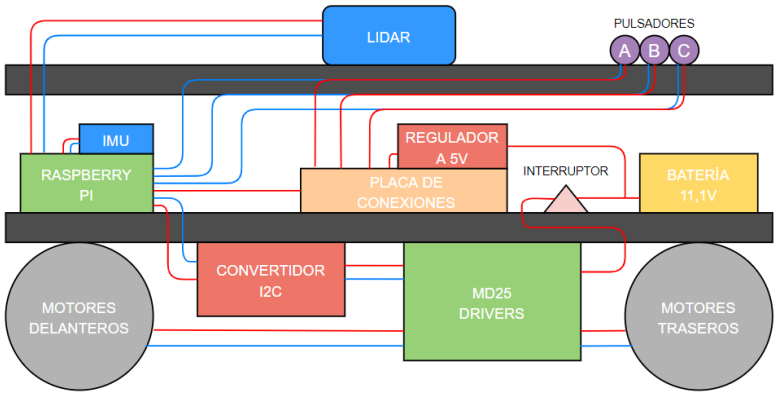


Ilustración 1 - Esquema del conexionado del vehículo.

4. Descripción general del software del vehículo

La parte referente al software del proyecto se encuentra dividida en 3 partes fundamentales: drivers en Matlab – Simulink, drivers en ROS, y pruebas ROS – Simulink.

4.1. Drivers en Matlab – Simulink

Para el presente proyecto se han realizado los drivers encargados de la instrumentación de la IMU *MPU-6000* y de dos MD25 encargados de la medida y actuación de cuatro motores de continua *EMG30*. La IMU ha sido configurada para la transmisión de datos a través del protocolo SPI; mientras que en los MD25 se ha optado por el uso del protocolo I2C, ligeramente más lento que el anterior. En el driver de la IMU se obtienen las medidas, calibradas y filtradas, procedentes de los acelerómetros y giróscopos de la *MPU-6000*. Para el control de los motores se ha desarrollado un driver que permite la lectura de los datos procedentes de los encoders, y el envío de los comandos de tensión que definirán la velocidad de los *EMG30*, todo ello realizado a través de las tarjetas MD25. En la Ilustración 2 pueden apreciarse la toma de medidas en los acelerómetros de la IMU durante un ensayo; mientras que la velocidad de avance y de rotación del vehículo propinada por los motores en otro ensayo diferente son mostradas en la Ilustración 3.

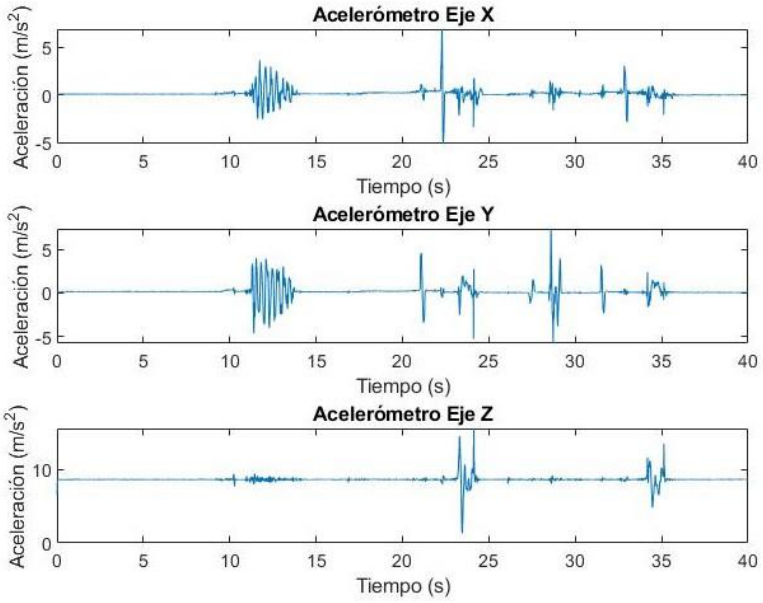


Ilustración 2: Ensayo de funcionamiento del driver de la IMU.

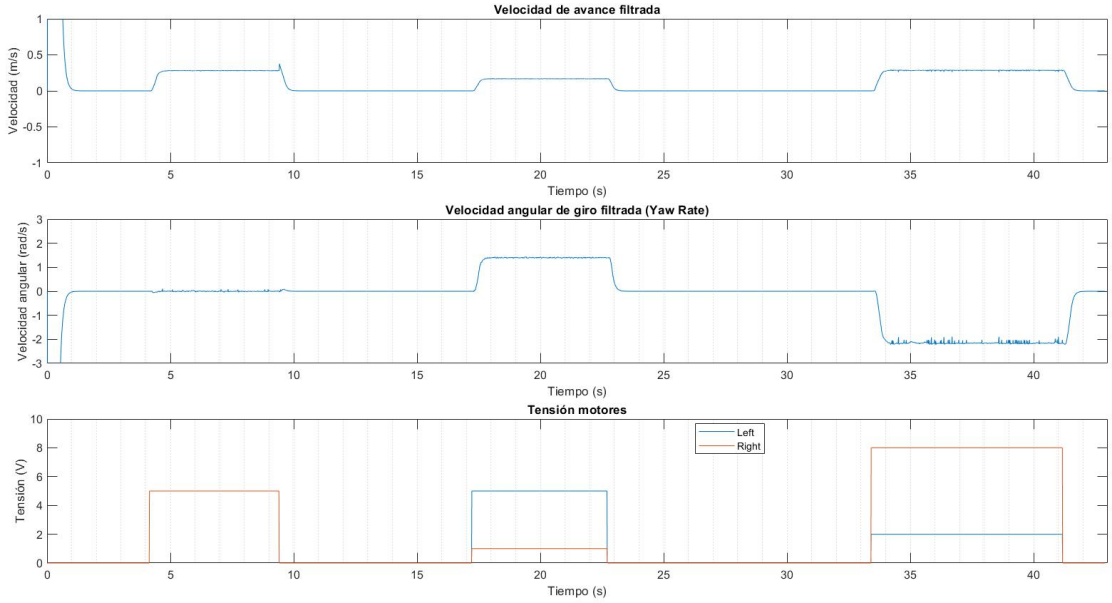


Ilustración 3: Ensayo de funcionamiento del driver de los MD25.

4.2. **Drivers en ROS**

El driver capaz de controlar la velocidad de rotación del LIDAR y obtener las coordenadas de los puntos referentes al entorno del vehículo del proyecto ha sido implementado en ROS. El driver es capaz de funcionar para todas las versiones de RPLIDAR A1, A2 y A3 a través de un puerto USB presente en un ordenador; que en el caso de este proyecto, se trata de la *Raspberry Pi 3 Model B+*. Además, se han realizado pruebas de funcionamiento de un algoritmo LIDAR SLAM capaz de ser visualizado en el entorno gráfico *RViz* de ROS. Los resultados de los ensayos realizados para la toma de medidas y del algoritmo LIDAR SLAM pueden ser visualizados en la Ilustración 4.

Ilustración 4: Resultados del driver LIDAR (izquierda), HECTOR SLAM (derecha).

4.3. **Pruebas ROS – Simulink**

Con la intención de solventar ciertos problemas que han ido surgiendo durante el desarrollo del proyecto, se han realizado numerosos ensayos y pruebas tratando de establecer los mejores métodos posibles para el desarrollo y adaptación de drivers en futuros proyectos robóticos. En concreto, se ha estudiado la forma de implementar diagramas de Simulink, con y sin paquete de soporte para Raspberry Pi, en nodos ejecutables en ROS; obteniendo resultados muy satisfactorios.

5. Conclusiones

Se han completado con éxito todos los objetivos fijados para este proyecto, en el que no solo se ha conseguido llevar a cabo la instrumentación y la adaptación software (driver) de los componentes hardware del vehículo; sino que ha sido posible también estudiar la adaptabilidad y generalización de los drivers en futuros proyectos.

El vehículo se encuentra totalmente operativo para poder investigar e implementar técnicas de control avanzadas basadas en LIDAR SLAM, completando así su función de vehículo autónomo.

Para futuros proyectos se recomienda el uso de un concentrador hardware tipo Arduino para realizar la conexión de ciertos sensores, como la IMU o los MD25, para transmitir los datos al ordenador de a bordo mediante comunicación en serie. Utilizando este método, los drivers diseñados en Simulink pueden ser compatibles con cualquier tarjeta SBC que haga uso del entorno ROS.

AUTONOMOUS VEHICLE NAVIGATION IN A LIMITED ENVIRONMENT

Author: González del Campo Artesero, Javier.
Supervisor: Zamora Macho, Juan Luis.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

The objective of this Project is the electronic instrumentation of measurement and actuation hardware alongside the development of the control software intended for an autonomous vehicle in a limited environment. The design of the vehicle is composed of four DC motors operated through two control cards, an IMU sensor, a LIDAR sensor, three push buttons, a Raspberry Pi as an on-board computer, and a LiPo battery as the vehicle's only power source.

Keywords: Autonomous vehicle, autonomous navigation, LIDAR, camera, MD25, IMU, MPU-6000, ROS, MATLAB, Simulink, Raspberry, driver.

1. Introduction

The automation and robotics industries are booming. The production processes in factories are becoming more automated every year, and the progress being made in areas such as the Internet of Things (IoT) or Artificial Intelligence are overcoming barriers that were unimaginable a few years ago. The world is currently immersed in a Fourth Industrial Revolution, a term coined by Klaus Schwab, founder of the World Economic Forum, in the 2016 edition of this event. This new Revolution is characterized by a fusion of technologies that seeks to blur the lines between the physical, biological, and digital worlds. It is heavily correlated with technological advances in fields such as robotics, Artificial Intelligence, nanotechnology, quantum computing, IoT, biotechnology, and autonomous vehicles, among others.

In this context, this Project looks to facilitate the instrumentation of small-sized vehicles combining software environments as different as MATLAB – Simulink and ROS (Robot Operating System), thus facilitating the study of numerous algorithms and Artificial Intelligence processes that can help the development of autonomous vehicles.

2. Project definition

The Project presented hereafter can be divided into the following main parts:

- Design and construction of the vehicle.
- Integration of the vehicle's measurement and actuation electronics through software drivers.
- Integration of MATLAB – Simulink and ROS software environments.

3. General description of the vehicle's hardware

The different hardware components integrated into the Project vehicle are shown in Figure 1, which shows a schematic of the voltage (in red) and data transmission (in blue) connections between the different hardware elements.

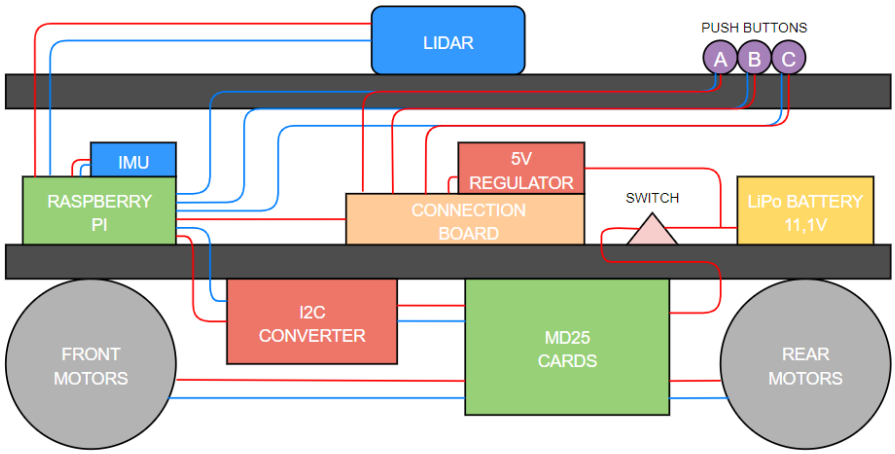


Figure 1: Schematic of the vehicle's connections.

4. General description of the vehicle's software

The software used in this Project can be divided into three fundamental parts: drivers in MATLAB - Simulink, drivers in ROS, and ROS - Simulink tests.

4.1. **Drivers in MATLAB – Simulink**

The drivers in charge of the instrumentation of the MPU-6000 IMU sensor and the two MD25 boards responsible for the measurement and performance of the four EMG30 DC motors have been successfully developed and integrated into the control system. The IMU has been configured to perform the duties of data transmission through the SPI protocol, while the I2C protocol – slightly slower than the previous one – has been selected to operate in the MD25 boards. The IMU driver provides calibrated and filtered measurements from the accelerometers and gyroscopes of the MPU-6000. For the control of the four motors installed a driver that operates through the MD25 boards has been successfully developed. The cited driver allows the reading of raw and filtered data form the encoders and the sending of the velocity commands that define the DC motors' speed. Figure 2 shows the measurements taken on the IMU accelerometers during a test, while Figure 3 shows the vehicle's forward and rotational speed driven by the engines carried out in a different test.

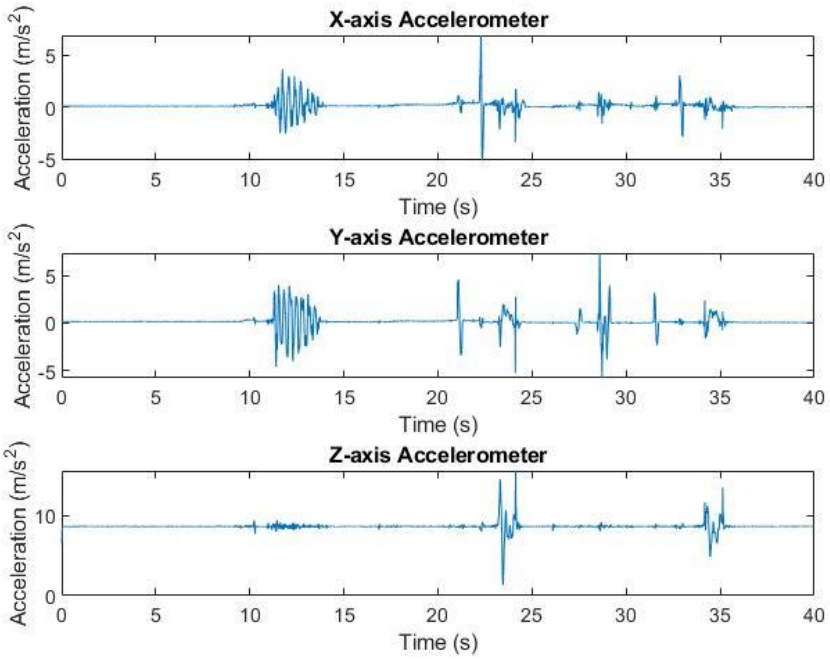


Figure 2: IMU driver performance test.

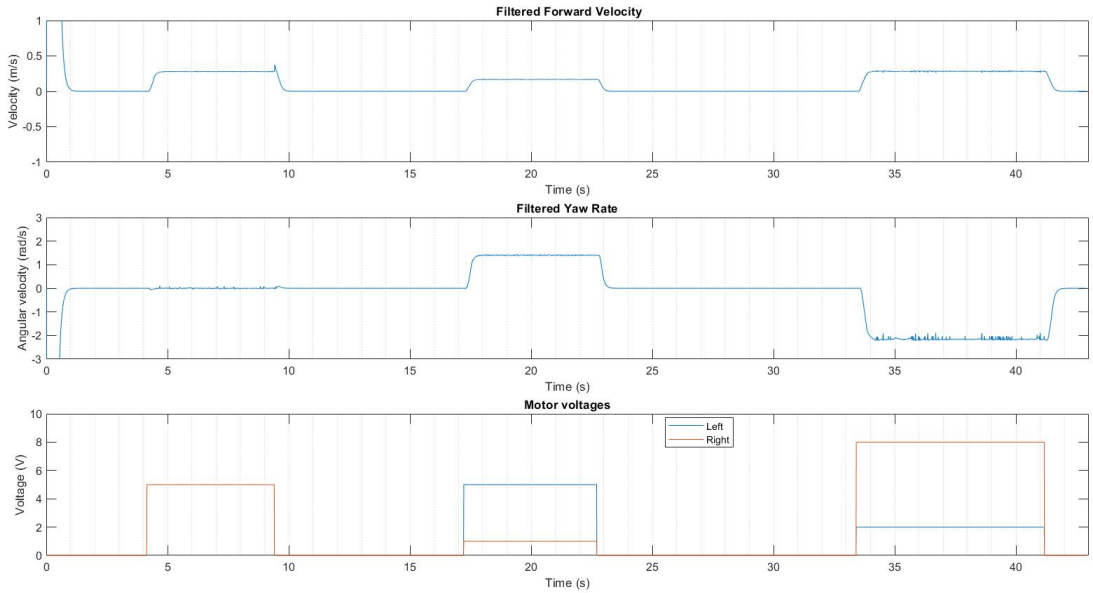


Figure 3: Md25 driver performance test.

4.2. **Drivers in ROS**

The driver capable of controlling the rotation speed of the LIDAR sensor and obtaining the coordinates of the point cloud representing the vehicle's environment has been implemented in ROS. The driver is compatible with all versions of the RPLIDAR A1, A2, and A3, through a USB port present in a computer, which in the case of this Project is the *Raspberry Pi 3 Model B+*. In addition, a LIDAR SLAM algorithm capable of being visualized in the *RViz* graphic environment of ROS has been tested. The results of the several tests carried out for the visualization of data measurement and the LIDAR SLAM algorithm are shown in Figure 4.

Figure 4: LIDAR driver results (left), HECTOR SLAM (right).

4.3. **ROS – Simulink**

To sort out certain problems that have arisen during the development of the Project, numerous trials and tests have been carried out trying to establish the best possible methods for the development and adaptation of drivers in future robotic projects. Specifically, a study has been carried out about how to implement Simulink diagrams – with and without support package for Raspberry Pi – in executable nodes in ROS, obtaining satisfactory results.

5. Conclusions

All the objectives set for this Project have been successfully completed, in which it has not only been possible to implement and adapt the drivers and control software for the vehicle's hardware components; but it has also been possible to study the adaptability and generalization of drivers in future projects as well.

The vehicle is fully operational to enable the investigation and implementation of advanced control techniques based on LIDAR SLAM, thus completing its autonomous vehicle function.

For future projects it is recommended to use an Arduino-type hardware hub to connect certain sensors, such as the IMU or the MD25, to enable the transmission of data to the on-board computer through serial communication. Using this method, drivers designed in Simulink can be compatible with any SBC card that makes use of a ROS environment.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1 INTRODUCCIÓN

El proyecto consiste en la implementación y diseño de un vehículo de escala reducida para el estudio y desarrollo de tareas de navegación autónoma en un entorno limitado. Para ello, es necesario llevar a cabo tanto el diseño hardware del vehículo, como la implementación electrónica de los sensores y actuadores al sistema de control del coche. La finalidad última del proyecto es la conformación de una base robusta y estable para el futuro estudio de algoritmos de navegación autónoma.

El nivel de autonomía de un vehículo es medido actualmente según la norma internacional J3016_201806, actualizada a su última versión en 2018 por SAE International (SAE – Society of Automotive Engineers), formalmente conocida como la Sociedad de Ingenieros de Automoción. Esta norma establece 6 niveles distintos de autonomía en la navegación, numerados del 0 al 5. [1]

Nivel 0: No Automatización

En este nivel de automatización, el conductor es responsable de la totalidad de las tareas de conducción. Se incluyen también en este grupo aquellos vehículos dotados de sistemas activos de seguridad tales como controles de estabilidad electrónica, sistemas de frenado automáticos de emergencia y algunos tipos de sistemas de asistencia a la conducción, como asistentes de seguimiento de carril.

Nivel 1: Asistencia a la Conducción

Este nivel de automatización agrupa aquellos vehículos capaces de realizar la ejecución sostenida y específica de un Dominio de Diseño Operacional (ODD) mediante un sistema de automatización de las tareas de conducción, ya sea del control de movimiento lateral o longitudinal del vehículo (pero no simultáneamente), con la expectativa de que el conductor realice el resto de las tareas de conducción.

Nivel 2: Automatización Parcial de la Conducción

Este nivel de automatización agrupa aquellos vehículos capaces de realizar la ejecución sostenida y especifica de un Dominio de Diseño Operacional (ODD) mediante un sistema de automatización de la conducción, tanto del control de movimiento lateral como longitudinal del vehículo, con la expectativa de que el conductor realice las labores de Detección y Respuesta frente a Objetos y Eventos (OEDR) y supervise el sistema de automatización de conducción.

Nivel 3: Conducción Autónoma Condicional

Este nivel de automatización agrupa aquellos vehículos capaces de realizar el desempeño sostenido de determinados Dominios de Diseño Operacional (ODD) de un Sistema de Conducción Autónoma (ADS) encargado de ejecutar todas las Dinámicas de Conducción (DDT). En este nivel, el conductor todavía debe estar preparado para responder a las solicitudes de intervención del Sistema de Conducción Autónoma (ADS); así como a posibles fallos en las Dinámicas de Conducción (DDT) de otros vehículos.

Nivel 4: Conducción Autónoma Alta

Este nivel de automatización agrupa aquellos vehículos capaces de realizar el desempeño sostenido de determinados Dominios de Diseño Operacional (ODD) de un Sistema de Conducción Autónoma (ADS) encargado de ejecutar todas las Dinámicas de Conducción (DDT). En este nivel, el Sistema de Conducción Autónoma (ADS) no realizará ninguna petición de respuesta al conductor.

Nivel 5: Conducción Autónoma Total

Este nivel de automatización agrupa aquellos vehículos capaces de realizar el desempeño sostenido e incondicional de cualquier Dominio de Diseño Operacional (ODD) de un Sistema de Conducción Autónoma (ADS) encargado de ejecutar todas las Dinámicas de Conducción (DDT) sin realizar ninguna petición de respuesta al conductor.

Como se puede observar, la diferencia con el anterior nivel de automatización es la capacidad del Sistema de Conducción Autónoma (ADS) de operar en cualquier entorno y bajo cualquier circunstancia. Este proyecto, según se ha explicado anteriormente, aspira a

alcanzar el nivel 4 de autonomía en un entorno limitado, como puede ser el diseñado en un laboratorio. Este proyecto, aparte de diseñar una base hardware sólida y totalmente funcional de un vehículo, también ahonda en el Sistema de Conducción Autónoma (ADS) del coche; encargado de procesar toda la información obtenida de los sensores y, mediante diferentes algoritmos de control, aplicar actuadores que respondan al comportamiento dinámico deseado.

1.2 MOTIVACIÓN

La motivación para realizar este proyecto nace de la infinidad de posibilidades que la actual tecnología ofrece para poder crear productos completamente nuevos y revolucionarios. Lógicamente, el desarrollo de este proyecto está a años luz de los desarrollos y algoritmos que manejan las grandes compañías; pero puede ser un comienzo práctico para comprender el funcionamiento de la toma de datos mediante sensores y su posterior procesamiento. Las aplicaciones y productos que se pueden crear con la tecnología actual son, tan solo, limitados prácticamente por la imaginación; y con una base sólida de comprensión de los procesos y controles básicos que actualmente se llevan a cabo, se puede adquirir la capacidad de dar forma a numerosas ideas.

El segundo aspecto más importante de este proyecto es que servirá como recurso a futuros estudiantes, que podrán hacer uso del vehículo instrumentado para comprender mejor las dinámicas de la teoría de control. Además, también servirá como base para poder realizar numerosos proyectos más avanzados que este, o para desarrollar con más precisión las ideas que en este se muestran.

1.3 OBJETIVOS

El objetivo de este proyecto es la instrumentación electrónica de medida y actuación, y la adaptación del software de control de un vehículo para navegación autónoma en un entorno limitado. El proyecto puede ser desglosado en los diferentes hitos:

1.3.1 DISEÑO Y CONSTRUCCIÓN DEL VEHÍCULO

El diseño del vehículo y de los múltiples sensores y actuadores que acompañan su estructura han sido diseñados de tal forma que su conjunto resulte elegante a la vez que accesible; por si fuera necesaria alguna reparación, o para facilitar futuras actualizaciones. El tamaño del vehículo es reducido, facilitando así su operación en entornos limitados; como laboratorios de prueba, que es el entorno para el cual ha sido diseñado.

1.3.2 INTEGRACIÓN DE LA ELECTRÓNICA DE MEDIDA Y ACTUACIÓN

Desarrollo y adaptación de los drivers de los diferentes componentes hardware del vehículo permitiendo la medición y actuación del control sobre los mismos. Todos los datos obtenidos por los diferentes sensores se integran en un filtro extendido de Kalman (EKF) que ha sido optimizado para poder obtener las mediciones más fiables posibles.

1.3.3 INTEGRACIÓN DE MATLAB – SIMULINK Y ROS

Se han realizado numerosas pruebas para establecer una pauta clara con la que poder integrar drivers procedentes de Simulink y de ROS en un mismo control ejecutado en Matlab y Simulink. También se han realizado varios ensayos para la conversión de diagramas de Simulink a nodos en ROS, ejecutables por tarjetas SBC sin la necesidad de que un PC con Matlab – Simulink corra dichos diagramas de forma externa.

1.4 RECURSOS EMPLEADOS

- Ordenador con el software de Matlab en su versión 2020a y Simulink con la instalación de las siguientes librerías:
  - Optimization Toolbox
  - System Identification Toolbox
  - Signal Processing Toolbox
  - DSP System Toolbox

  - Model Predictive Control Toolbox
  - Symbolic Math Toolbox
  - MATLAB Support Package for Raspberry Pi Hardware
  - Simulink Support Package for Raspberry Pi Hardware
  - ROS Toolbox
  - Matlab Coder
  - Simulink Coder
  - Embedded Coder

- Herramientas de microelectrónica habituales:
  - Soldador de estaño
  - Multímetro
  - Destornillador
- Estructura del vehículo
- Sensores, actuadores y microprocesadores:
  - Raspberry Pi 3 Model B+
  - 4 motores EMG30
  - 2 drivers MD25 para controlar sendos motores
  - IMU modelo MPU6000
  - Batería LiPo de 1300mAh y 11,1V de salida
  - Convertidor de tensión de 5V a 3,3V
  - Regulador de tensión de 12V a 5V
  - 3 pulsadores

- 1 interruptor
- 1 LIDAR modelo RPLIDAR A2M8

Capítulo 2. ESTADO DEL ARTE

En el estado actual de la industria, se pueden encontrar numerosos ejemplos de vehículos autónomos con un nivel de automatización alta, enmarcados en un nivel 4, según lo definido en el Punto 1.1. El ejemplo más reciente[1] del que data la industria, ha surgido a raíz de las complicaciones en el reparto de productos alimenticios y de primera necesidad en Estados Unidos debido al COVID-19. [2] Se trata del uso de los vehículos autónomos de baja velocidad de la empresa Nuro, concretamente de una flota de unos 5.000 vehículos del modelo R2; como el que se puede apreciar en la Figura 1. La Autoridad de Tráfico de Estados Unidos, la NHTSA (National Highway Traffic Administration), fue la que otorgó el permiso temporal que hizo posible la puesta en marcha de estos vehículos durante la crisis pandémica global. Con dicho número limitado de vehículos, Nuro está presente tan solo en las ciudades de Scottsdale, AZ y de Houston, TX. [3]

Figura 1: Vehículo R2 de Nuro.

Sin embargo, la industria de la conducción autónoma no solo agrupa a vehículos que circulan por las calles y carreteras; sino que también incluye otros vehículos capaces de desempeñar

[1] Información datada a fecha de 27 de abril de 2020.

funciones muy diversas. La conducción autónoma lleva estando presente desde hace varios años en productos como aspiradoras inteligentes capaces de mapear y procesar las dimensiones de varias habitaciones y programar rutas para limpiar de manera eficiente un domicilio. La mayor parte de este tipo de robot-aspirador hace uso, en cierta medida, de un sistema LIDAR o un sistema VSLAM para obtener las distancias vectoriales de los obstáculos del entorno que les rodea. El funcionamiento detallado de ambas tecnologías, junto con sus respectivas ventajas y desventajas se detallarán en el apartado 2.2.1.

Con el objetivo de poder analizar de una forma más estructurada el estado del arte del proyecto que se ha llevado a cabo se procederá a separar dicha exposición en dos bloques diferenciados: el hardware y el software; para terminar el desarrollo del presente capítulo con la presentación del vehículo que sirve como referencia y motivación para el desarrollo del proyecto.

2.1 HARDWARE

En este apartado, se procederá a presentar los diferentes sensores y actuadores más utilizados en las industrias de conducción autónoma y robótica.

2.1.1 ACELERÓMETROS Y GIROSCOPIOS

Los acelerómetros y giroscopios son sensores que llevan varias décadas en el mercado y son fundamentales para definir la orientación y la posición de objetos. Los acelerómetros son sensores capaces de medir la aceleración lineal a lo largo de un eje. Gracias a esta medida, la velocidad lineal y la posición pueden ser conocidas mediante simples integraciones. Los giroscopios que se suelen utilizar en robótica, y los que se van a utilizar en este proyecto, son sensores basados en sistemas micro-electro-mecánicos (MEMS) integrados en chips de silicio. Estos dispositivos son capaces de medir la velocidad angular sobre un eje; y como ocurre con los acelerómetros, son capaces de indicar la posición y la aceleración angular mediante integración y derivación.

Sin embargo, para poder medir de forma eficaz el desplazamiento y la orientación de un dispositivo, los acelerómetros y los giroscopios suelen ir integrados en unidades de medida inerciales (IMU) con una configuración de 3 giroscopios y 3 acelerómetros dispuestos en ejes ortogonales para su correcto desempeño. [4]

2.1.1.1 Magnetómetros

Los magnetómetros son dispositivos que sirven para cuantificar en fuerza o dirección una señal magnética. En robótica, los más comunes son los magnetómetros de efecto Hall, que consisten en un semiconductor que produce una tensión proporcional a la intensidad del flujo magnético que lo atraviesa, en una dirección normal a la corriente de flujo. [4]

Unidades de medida inerciales (IMU) más sofisticadas, comúnmente conocidas como IMUs de 9 ejes, suelen añadir a la configuración de giroscopios y acelerómetros anteriormente descrita, 3 magnetómetros dispuestos en ejes ortogonales. Esto es debido a que la configuración únicamente de acelerómetros y giroscopios suele llevar asociada errores de posición y orientación acumulativos. Estos suelen provenir de los giroscopios, que, al proporcionar medidas relativas sobre la velocidad de rotación en cada uno de sus ejes, necesitan de la integración para calcular los ángulos de giro de cada uno de ellos. El problema radica en la integración (método conocido como dead reckoning); ya que errores constantes en la medida de los giroscopios se acumularán de forma lineal tras dicha conversión, produciendo errores cada vez más grandes en la orientación. Añadiendo otro tipo de sensor, como los magnetómetros, permite al algoritmo encargado de interpretar las medidas de los sensores, compensar y corregir dichos errores acumulados en el tiempo y obtener así, unas medidas de posición y orientación mucho más precisas y robustas. [4] [5]

2.1.2 ENCODERS

Los encoders son dispositivos electromecánicos usados para obtener la posición lineal o angular de un eje a través de señales analógicas o digitales. Es posible encontrar una gran variedad de este tipo de sensores en el mercado debido a su alto número de aplicaciones en la industria. Fundamentalmente, según el tipo de movimiento para el cual estén diseñados para medir, los encoders se dividen en dos tipos: lineales y rotativos. Los encoders lineales

son frecuentemente utilizados en máquinas industriales que necesitan una gran precisión en los desplazamientos lineales de sus partes móviles; como por ejemplo, los ejes por los que se mueve el fusor de una impresora 3D. Por el contrario, los encoders rotativos son utilizados para obtener información sobre el desplazamiento angular de un eje; y son ampliamente utilizados en sistemas mecánicos y robótica. [6]

Los encoders son sensores que se pueden clasificar también según la tecnología que usen para obtener la información del movimiento que miden: ópticos, magnéticos y mecánicos. Este proyecto se hace uso de encoders magnéticos de efecto Hall, en los cuales se utilizan series de polos magnéticos opuestos (positivos y negativos) orientados de tal forma que un sensor magnético-resistivo varíe su tensión de salida de forma proporcional a las variaciones de campo magnético; obteniendo así información sobre el movimiento. [7] [8]

Por último, los encoders también pueden clasificarse en absolutos o incrementales, según la información que se precise obtener del movimiento. Para explicar la diferencia, se usarán los encoders rotativos debido a la implicación que tienen en el proyecto; siendo toda la información descrita a continuación extrapolable para los encoders lineales.

Los encoders absolutos rotativos son dispositivos que miden la posición angular absoluta de una rotación y mantienen la información sobre la misma al encender y apagar el sensor, sin la necesidad de volver a un punto de calibración. Esto es gracias a que la señal medida por el sensor al rotar produce un código único equivalente a una posición angular concreta, ver Figura 2.

Los encoders rotativos incrementales, por el contrario, son dispositivos capaces de medir únicamente las variaciones en la posición angular que sufre el movimiento; y no la posición absoluta del mismo. Esto es debido a que la señal obtenida del sensor resulta únicamente en un tren de pulsos, ver Figura 3. Gracias a esta medición, es sencillo obtener la velocidad de rotación del movimiento medido al dividir las variaciones de posición entre el tiempo transcurrido. Algunos encoders incrementales son capaces también de identificar el sentido de giro y la posición angular absoluta gracias a la inclusión de patrones más sofisticados en el disco del sensor. Sin embargo, en el caso de la posición angular absoluta, la información

se perderá al apagar y encender el sensor, teniendo este que volver a un punto de calibración para volver a realizar dicha medida. [6] [7]

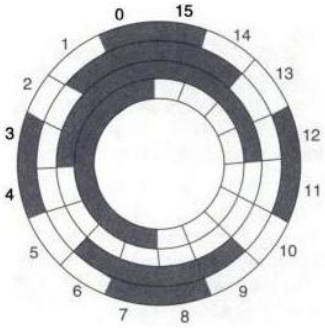


Figura 2: Encoder rotativo absoluto.

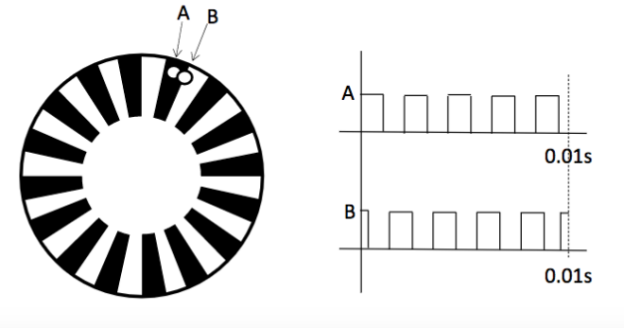


Figura 3: Encoder rotativo incremental.

2.1.3 SENSORES POR ULTRASONIDO

Los sensores por ultrasonido (*ultrasonic rangefinder*, en inglés) son dispositivos capaces de medir la distancia a un objeto utilizando ondas sonoras, ver Figura 4. Son sensores que trabajan libres de roces mecánicos y son capaces de detectar objetos a distancias que van desde pocos centímetros hasta varios metros. Dicha distancia es medida al emitir una señal sonora a una frecuencia ultrasónica y procesar la onda de vuelta (el eco de la onda inicial) que recibe el micrófono del sensor. Las altas frecuencias son usadas en estos sensores debido a su mejor desempeño en la medición de distancias cortas y a la obtención de una mayor precisión en las medidas. Son sensores ampliamente utilizados en máquinas industriales, en la industria del automóvil y en robótica; aunque quizás el ejemplo más frecuente se encuentre en los sensores de aparcamiento que incorporan ciertos vehículos.

La distancia entre el objeto y el sensor es calculada usando el tiempo transcurrido entre la emisión de la onda generada y el retorno de esta al sensor, considerando que la velocidad de propagación del sonido en el aire es de aproximadamente 343 m/s. [7]

El cálculo es sencillo, tal y como refleja la siguiente fórmula:

$$L = \frac{C * t}{2}$$

Ecuación 1: Distancia a un objeto. Principio de "tiempo de vuelo".

Donde *L* es la distancia medida, C es la velocidad de propagación del sonido en el aire, y *t* es el tiempo transcurrido. [9]

El entendimiento de la zona de detección de este tipo de sensores es muy importante a la hora de conseguir una correcta medición o detección de un objeto. La geometría del haz de onda emitida por el sensor es típicamente descrita como un cono de cierto ángulo. Este ángulo, describe el arco con el que la onda de ultrasonido emana del transductor (dispositivo capaz de convertir señales eléctricas en ultrasonido, y viceversa); y es esencial para determinar la zona de detección que poseerá la onda. Es posible el aumento del área de dicha zona de detección a través del uso de múltiples sensores orientados en diferentes ángulos. Ver Figura 5. Sin embargo, al adoptar esta última configuración, es necesario considerar el cruce y superposición de ondas a la hora de interpretar la información recibida. [7]

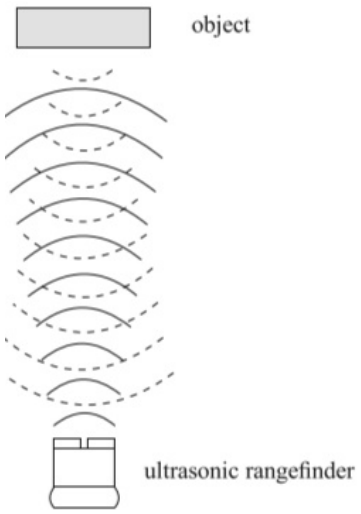


Figura 4: Sensor por ultrasonido.

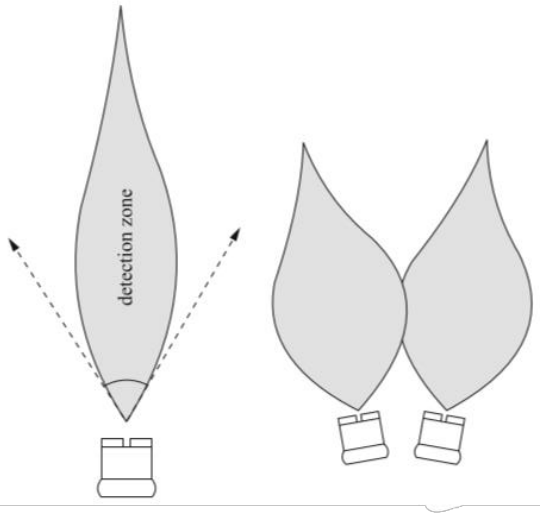


Figura 5: Zona de detección.

Los sensores por ultrasonidos pueden presentar fallos debidos a múltiples causas; algunas de las cuales se mencionarán a continuación. Las medidas más fiables de distancia se obtienen cuando el objeto que refleja la onda se encuentra localizado en el centro de la zona de detección del sensor. Sin embargo, esta simple regla no siempre se cumple; ya que también han de tenerse en cuenta factores como el tamaño, la composición, la forma, y la orientación de los objetos a la hora de evaluar la eficacia y la viabilidad de la medida. Ver Figura 6. Los objetos que se desean medir deben ser deflectores de sonido para poder reflejar la onda emitida por el sensor. Además, deben estar, aparte de en una orientación adecuada para la reflexión de la onda, a una distancia no demasiado alejada; ya que cuanto mayor sea la distancia, mayor será la dispersión de la onda, y menor la intensidad con la que retorne al sensor. [7] [10]

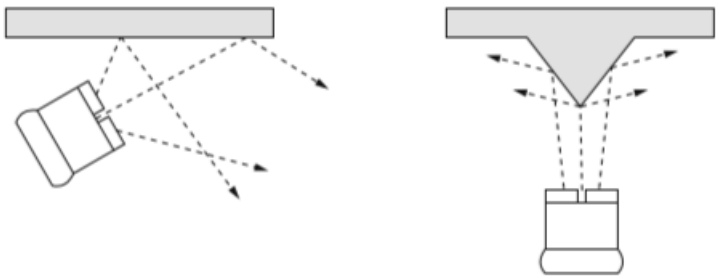


Figura 6: Posibles errores de un sensor por ultrasonido.

Este tipo de sensores presenta también ciertas ventajas respecto a sensores de otro tipo, como pueden ser los sensores por láser infrarrojo. Esto es debido a que, al contrario que estos últimos, los sensores por ultrasonido no se ven afectados por la luz infrarroja desprendida por el sol, haciéndolos idóneos para uso en condiciones de mucha luz. [11]

2.1.4 SENSORES LÁSER

Los sensores láser (*laser rangefinder*, en inglés) son dispositivos capaces de medir la distancia de un objeto utilizando un rayo láser. Son sensores que trabajan libres de roces mecánicos, y el tipo más común de sensor láser mide la distancia a un objeto basándose en el principio de "tiempo de vuelo". El principio es el mismo que el comentado anteriormente en los sensores por ultrasonidos, pero aplicado, en este caso, a la tecnología láser. La distancia a un objeto queda determinada midiendo el tiempo transcurrido entre la emisión de un pulso láser, desde el sensor, hasta la recepción del mismo tras haber sido reflejado en la superficie de dicho objeto. La formulación es la misma que la de los sensores por ultrasonidos, ya que se basan en el mismo principio. Ver Ecuación 1. [7] [9]

Otra posibilidad, a la hora de medir la distancia con el sensor, es la utilización de un cambio de fase de frecuencia múltiple. Esta técnica consiste en medir el cambio de fase de múltiples frecuencias tras reflejarse en un objeto y compararlas con una señal de referencia. Sin embargo, y sin olvidar los dos métodos anteriores, el procedimiento más preciso para la medición de variaciones en la distancia es la interferometría. [7] Véase Figura 7.

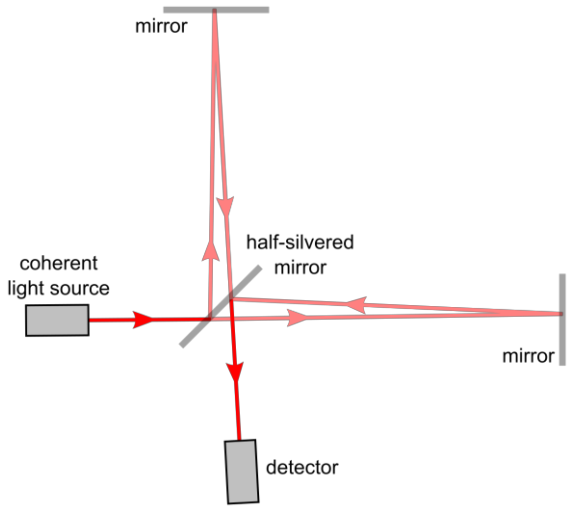


Figura 7: Estructura para la medición por interferometría.

En la estructura mostrada, todos los componentes excepto el espejo reflector de la derecha, están incluidos en el sensor. Esto permite la medición de objetos, siempre y cuando estos sean reflectores de luz infrarroja, para poder volver así al espejo refractor central, que combinará las ondas reflejadas por el espejo reflector de referencia para la medición de las ondas recibidas (situado arriba en la estructura de la Figura 7). [12]

Este tipo de sensores poseen resoluciones de unos 3nm y son extremadamente precisos. Sin embargo, el tamaño y peso de los sensores es demasiado grande y tienen un alto precio en el mercado. Debido a esto, suelen ser usados en la industria para comprobar las tolerancias en la fabricación de componentes metálicos, o en la astronomía (radioastronomía). [12] [13] Su alto peso, volumen y precio, a parte de su alta precisión, innecesaria para las aplicaciones robóticas de posicionamiento y localización, hacen que no sea el método adecuado para la utilización en las aplicaciones robóticas comentadas, que son las que pretende enfocar este proyecto.

El proyecto se centra en los sensores basados en el primer método de medición descrito en este apartado, mediante el uso del principio de "tiempo de vuelo". En robótica, existen dos tipos de sensores que utilizan dicho principio: [14]

- Sensores infrarrojos por proximidad: utilizan el nivel de luz infrarroja reflejada en un objeto para determinar si dicho objeto se encuentra dentro de un rango de valores predeterminado. Devuelven una respuesta booleana (sí o no) respecto a la medición efectuada.

- Telémetros láser: utilizan un pulso láser reflejado en un objeto para determinar la distancia a dicho objeto. La respuesta devuelta por el sensor, en este caso, es la distancia entre el objeto o superficie medida y el sensor.

Debido a las aplicaciones del proyecto, y ya que se pretende realizar el mapeo de un laboratorio y utilizar dichos datos para el posicionamiento y localización del robot al desplazarse dentro del mismo; se hará uso de un tipo de telémetro láser de rotación llamado LIDAR.

2.1.4.1 LIDAR

Aunque también existen LIDAR (*Light Detection and Ranging ó Laser Imaging Detection and Ranging*) de tipo fijo, que cumplen la misma función que los telémetros láser descritos en el apartado anterior; este proyecto se va a enfocar en los LIDAR rotativos. Esto es debido a que los fijos son unidimensionales, y por tanto, únicamente capaces de realizar mediciones de distancia a un objeto cada vez. Para poder realizar mediciones a lo largo de todo el campo de visión, dentro del alcance determinado del sensor, surgieron los LIDAR rotativos.

Esta tecnología es, básicamente, un telémetro láser montado sobre un motor capaz de girar a gran velocidad, captando información del entorno en 360º. En concreto, se trata de un foco emisor de haces de rayos infrarrojos, y de una lente receptora infrarroja con un escáner capaz de recibir y analizar los haces láser rebotados en los objetos. El resultado de la medición es una nube de puntos que proporciona la distancia a los diferentes objetos del entorno. Los puntos generados representan la posición relativa de los diferentes objetos respecto al sensor del LIDAR. [7] [15] La generación de los puntos de información puede verse en las representaciones de la Figura 8 y la Figura 9.

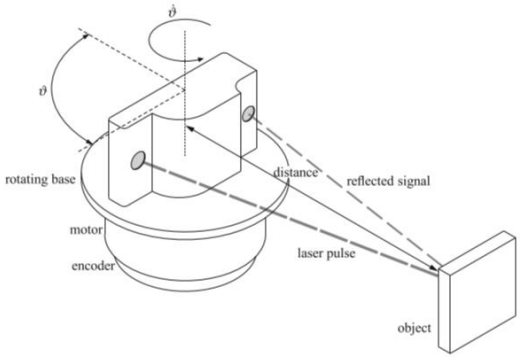


Figura 8: LIDAR.

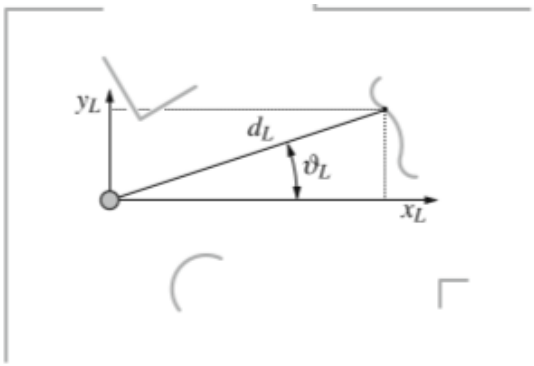


Figura 9: Cálculo de puntos LIDAR.

La distancia d_L es medida usando el foco emisor del láser y el escáner del LIDAR; mientras que el ángulo de rotación v_L es medido típicamente gracias a un encoder asociado al motor que hace efectivo el giro del LIDAR. Los puntos, por lo tanto, son obtenidos en coordenadas

polares; que pueden ser fácilmente convertidas a cartesianas relativas al sensor mediante la siguiente ecuación: [7]

$$x_L = d_L\, cos\varphi_L \quad \text{y} \quad y_L = d_L\, sin\varphi_L$$

Ecuación 2: Transformación de coordenadas polares a cartesianas.

La nube de puntos obtenida puede ser utilizada para la generación de un mapa del entorno (*mapping*), *path planning* y evasión de obstáculos. También, son bastante habituales en trabajos de Topografía, Geología, Arquitectura e Ingeniería civil. [7] Aunque, en referencia a los tres primeros usos descritos, cabe destacar que la tecnología LIDAR está en auge en el sector de la robótica y el desarrollo de vehículos autónomos. El ejemplo más avanzado y mejor implementado en términos de diseño de este tipo de sensores en el campo de la automoción es el que monta el prototipo Volkswagen I.D. Concept. Un diseño en el que en cada una de las esquinas del techo del coche se esconden cuatro LIDAR cilíndricos compactos que son capaces de escamotearse en el techado del vehículo. De esta forma, aparecen ocultos cuando se conduce el vehículo en modo manual, y se elevan cuando se activa el modo de conducción autónoma. [15] Ver Figura 10.

Figura 10: Volkswagen I.D. Concept.

2.1.5 CÁMARAS

Los diferentes tipos de cámaras, de los cuales se hablará durante esta sección, se están convirtiendo en un sensor cada vez más utilizado en la industria de la robótica, la creación de entornos digitales y aplicaciones de realidad aumentada. La gran ventaja de este tipo de sensores es la inmensa cantidad de información que son capaces de proporcionar y la excelente relación calidad-precio que ofrecen respecto a otro tipo de sensores, como el LIDAR. Es necesario aclarar que la forma de enfocar este tipo de sensor se va a realizar analizando las ventajas e inconvenientes que ofrece en el ámbito de la robótica y la navegación autónoma, que es el principal objetivo que se desarrolla a lo largo de este proyecto.

Se van a utilizar como ejemplo los productos de Intel para la explicación de los distintos tipos de cámaras; debido a que son actualmente la empresa puntera en la fabricación de este tipo de sensores y de su software asociado. Actualmente existen en el mercado dos tipos principales de cámaras con aplicaciones robóticas y de conducción autónoma. Intel ofrece una separación que ha nombrado de la siguiente forma:

- Intel RealSense Depth Camera series.
- Intel RealSense Tracking Camera series.

2.1.5.1 RGBD o Stereo Depth Cameras

El primer tipo ofrecido por Intel, denominado Depth, es un tipo de sensor que posee dos cámaras situadas en el mismo eje horizontal, tratando de imitar la vista humana. Este tipo de sensores, se conocen en el mercado como cámaras RGBD (*Red, Green, Blue, Depth*) o *Stereo Depth Cameras*. Incorporan un par de cámaras de alta resolución, que proporcionan la imagen RGB e imágenes de comparación; y un emisor y receptor activo de infrarrojos, que proporciona la medición de distancia a los diferentes objetos del entorno. El modo de funcionamiento es el siguiente:

Gracias a las dos cámaras RGB, situadas a una distancia conocida (ver Figura 11), un mismo píxel puede ser analizado desde dos diferentes ángulos, pudiendo conocer así la profundidad del mismo. Los algoritmos de reconocimiento y procesado de los diferentes pixeles utilizados para la medición de la profundidad están completamente integrados y desarrollados por Intel; y se procesan en una unidad incorporada en sus cámaras para ahorrar tiempo de procesamiento a los procesadores externos que se deseen conectar. Además de esto, las *Stereo Depth Cameras* incluyen también un foco emisor de patrones de rayos infrarrojos, a través de los cuales, y debido a la distorsión que sufre el patrón emitido (que es conocido), son capaces de obtener la medición de profundidad de los objetos de manera independiente a las cámaras RGB. Ver Figura 12. Esto hace que mediante la fusión de datos en un filtro se obtengan medidas de profundidad más robustas y precisas; pudiendo obtenerse tanto en exteriores como en interiores, con luz o sin luz. Esto permite un mapeado de puntos 3D prácticamente perfecto del entorno. [16] [17]

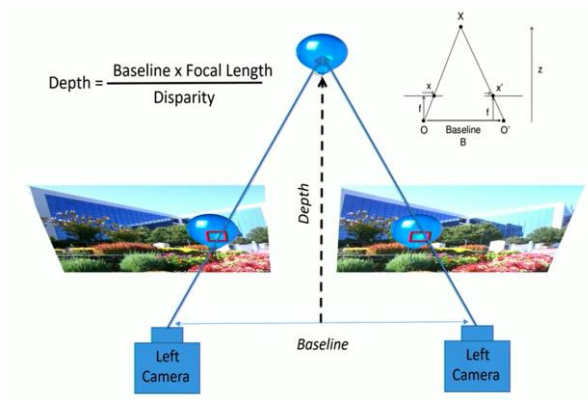


Figura 11: Cálculo de profundidad de las cámaras RGB.

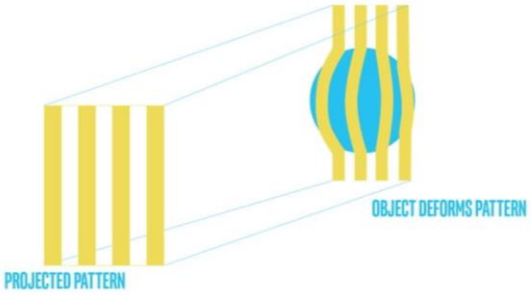


Figura 12: Patrón infrarrojo.

Sin embargo, pese a ser un tipo de sensor muy potente y con un amplio abanico de posibilidades, carece de una función que es esencial cuando se quiere diseñar un prototipo de conducción autónoma: el cálculo y seguimiento de la posición del vehículo. Para subsanar esta carencia, existen varias opciones: la utilización de una IMU en conjunto con los

encoders de los motores de las ruedas del vehículo (radio conocido), el uso de un LIDAR o el uso del segundo tipo de cámaras ofrecido por Intel comentado anteriormente.

2.1.5.2 Tracking Cameras

El segundo tipo de cámaras ofrecido por Intel mencionado con anterioridad son las denominadas *Tracking Cameras*, que incluyen una IMU en su interior, en conjunto con dos cámaras de ojo de pez (para poder visualizar un mayor campo de visión) y un chip *Movidius Myriad 2 VPU* encargado de procesar la información de la IMU y las cámaras. Esta solución, aunque Intel asegura que proporciona un seguimiento de la posición en una trayectoria cerrada (*closed loop*), nunca será tan robusta ni precisa como la utilización de un LIDAR. [17] Esto es debido a que el *tracking* de la posición es llevado por cámaras, que aunque ofrecen un campo de visión mucho más amplio que las integradas en sus cámaras RGBD pueden verse afectadas ante un entorno muy similar y de color parecido. Para entender mejor este problema, imaginemos que se desea realizar el mapeado de una habitación que posee paredes de formas circulares continuas y del mismo color. En este caso, las *Tracking Cameras* no serán capaces de diferenciar las imágenes de una pared a otra; y mientras la cámara RGBD realiza el mapeado 3D del entorno, el vehículo quedaría incapaz de *trackear* su posición relativa en la habitación. Con el LIDAR, este problema no ocurriría, aunque cabe resaltar, que el entorno propuesto es un entorno muy específico que muy raramente se va a dar.

2.2 SOFTWARE

En este apartado, se procederá a presentar las diferentes técnicas computacionales integradas en software más utilizadas en el diseño de un vehículo de conducción autónoma en el campo de la robótica.

2.2.1 SLAM

SLAM es un acrónimo de *Simultaneous Localization and Mapping*, en inglés; y corresponde a un sistema capaz de determinar la orientación y posición de un robot mediante la creación de un mapa de sus alrededores, mientras se realiza un *tracking* de la posición de dicho robot dentro del entorno en el que se encuentra. En este sistema, se procede a la realización de dos funciones principales, como son el mapeo del entorno y el sistema de navegación del vehículo autónomo. Para ello, lo más frecuente en el ámbito de la robótica es la utilización de una IMU (ver Apartados 2.1.1 y 2.1.1.1) en conjunción con otros sensores ópticos. Los dos sensores ópticos más utilizados, dan lugar a los dos tipos predominantes de tecnologías SLAM: [18]

- VSLAM (*Visual SLAM*): basado en el uso de cámaras.
- LIDAR SLAM: basado en el uso de un LIDAR 2D o 3D.

2.2.1.1 VSLAM

VSLAM o Visual SLAM es un sistema que hace uso de cámaras para mapear y generar puntos de ruta de navegación. Como se ha mencionado con anterioridad, es muy frecuente la inclusión de una IMU en este tipo de procedimiento; acuñando el nombre de VIO (*Visual-Inertial Odometry*).

Típicamente, en un sistema VSLAM, una serie sucesiva de puntos de interés calculados y generados por un algoritmo son identificados y seguidos haciendo uso de sucesivos *frames* aportados por las cámaras. Estos puntos de interés son utilizados para triangular la posición 3D del robot respecto a su entorno en un proceso denominado *feature-point triangulation*. El uso de la IMU es recomendable en este tipo de prácticas para hacer más robusto el proceso

señalado anteriormente; sobre todo en vehículos aéreos como drones, que no pueden incorporar los datos de odometría[2] provenientes de los encoders de sus motores tractores. [18]

Uno de los mayores inconvenientes asociados a esta tecnología es la reproducción de errores causados por la diferencia entre la localización percibida de los puntos de referencia y la localización real de los mismos. Para ello, es de vital importancia la realización de una calibración óptica de la cámara que minimice las posibles distorsiones geométricas de los datos obtenidos, que serán utilizados por el algoritmo SLAM elegido para realizar la navegación y localización del vehículo.

2.2.1.2 LIDAR SLAM

LIDAR SLAM es un sistema que hace uso de un LIDAR en conjunto con una IMU para realizar las tareas de mapeado y navegación de un vehículo. Funciona de una manera similar al VSLAM, solo que de una forma mucho más precisa.

Como ya se comentó en el Apartado 2.1.4.1, un LIDAR es un sensor que se compone de un foco emisor de haces de rayos infrarrojos y de una lente receptora infrarroja con un escáner capaz de recibir y analizar los haces láser rebotados en los objetos. Debido a la alta velocidad de la luz y la cantidad de información recibida por segundo, hacen de este sistema el más fiable y robusto de los dos comentados. Sin embargo, a no ser que se haga uso de un LIDAR 3D, este sistema posee un problema bastante grave. Con la utilización de un LIDAR 2D, la información del entorno queda limitada únicamente a los objetos que se encuentran en el mismo plano horizontal que el sensor; perdiendo gran parte de información del entorno y pudiendo causar choques o accidentes, al no ser posible la identificación de obstáculos que no se encuentren en dicho plano, como escaleras u objetos pequeños. [4] [17] [18]

[2] La odometría es el estudio de la estimación de la posición de un vehículo con ruedas a lo largo del tiempo mediante el uso de información proveniente de la rotación de sus ruedas.

Debido a esto, y según las necesidades del robot a diseñar, es frecuente la fusión de ambas tecnologías para garantizar un desempeño óptimo del mapeado y la navegación del robot. Precisamente, existen en el mercado cámaras diseñadas específicamente para realizar el mapeado 3D y detección de obstáculos del entorno (las cámaras RGBD); que en conjunto con un LIDAR o una Tracking Camera, proveen la información necesaria para el correcto desempeño de la tarea SLAM.

2.2.2 FILTROS DE KALMAN

El uso de filtros en aplicaciones robóticas, y especialmente, en el diseño de sistemas de navegación es una práctica altamente extendida debido a la importante función que desempeñan a la hora de realizar una medición precisa de diferentes variables, o de fusionar de forma eficiente la lectura de medidas redundantes en el sistema provenientes de diferentes sensores. En este proyecto, se describirán brevemente las principales características y funciones que desempeñan algunos filtros de Kalman, ya que suelen ser los más utilizados para la navegación de vehículos autónomos. Este tipo de filtros permiten la realización de dos tareas fundamentales: [4] [19]

- Estimar de forma óptima variables de interés de un sistema que no pueden ser medidas de forma directa, pero que se encuentran relacionadas matemáticamente (a través del modelo matemático del sistema) con otras variables que sí pueden ser medidas.

- Proporcionar la mejor estimación de una variable de estado a través de la fusión de diferentes medidas, relacionadas matemáticamente con dicha variable de estado, provenientes de diferentes sensores; y en presencia de ruido. Esto se traduce en una estimación del estado mucho más precisa y robusta que la que puede obtenerse con cada una de las medidas redundantes por separado.

Todos los filtros de Kalman son observadores de estado que basan sus algoritmos en métodos estocásticos, siguiendo una distribución gaussiana o normal, para representar la

incertidumbre producida por el ruido en el proceso de estado y en las medidas. Existen varios tipos de filtros de Kalman según la naturaleza del sistema. [19]

- Filtro Discreto de Kalman: estima de forma óptima variables de estado de un **sistema lineal** en presencia de ruido, pudiendo realizar dicha estimación incorporando una fusión de medidas procedentes de diferentes sensores.

- Filtro Extendido de Kalman: estima de forma óptima variables de estado de un **sistema no lineal** en presencia de ruido, pudiendo realizar dicha estimación incorporando una fusión de medidas procedentes de diferentes sensores. Para que la estimación sea precisa y fiable, el sistema no lineal debe poseer una buena linealización del mismo, siendo diferenciable y localmente lineal.

2.2.3 ROS

ROS (*Robot Operating System*) es un entorno diseñado en 2003 en la Universidad de Stanford en colaboración con Willow Garage que fue creado con la idea de proveer librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones robóticas. Con el fin de materializar las ideas por las cuales fue creado, ROS ofrece una gran abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes; además de un entorno software de código abierto, bajo la licencia BSD. [20]

2.2.3.1 Ventajas

Tras una breve introducción de ROS, es necesario ahondar en las diferentes ventajas que ofrece como entorno de desarrollo y en cómo ha revolucionado la industria y el sector de la robótica.

Antes de la llegada de ROS, la maquinaria robótica presente en las fábricas era diseñada por grandes compañías especializadas que utilizaban software privados y altamente especializados para cumplir con las funciones específicas que el robot debía cumplir. Cada vez que era necesaria la implementación de una nueva maquinaria, ya sea en la misma fábrica

o en otra distinta con otra compañía de ingeniería encargada de su desarrollo, los ingenieros encargados del diseño de la maquinaria tenían que desarrollar todo el proyecto desde la base. Es decir, un equipo de ingeniería tenía que desarrollar su propio software y diseñar las funciones y movimientos que fuesen a incorporar sus máquinas robóticas de forma individual y privada. La integración de otros proyectos al suyo propio se hacía una tarea realmente complicada, ya que cada empresa poseía sus propios métodos de abstracción y sus propios protocolos de comunicación. Esta falta de estandarización en la industria hacía que los proyectos robóticos fuesen muy costosos y sufriesen de tiempos de desarrollo muy elevados. En la búsqueda de un estándar en la industria de la robótica, surgió ROS, cuyas principales virtudes son las siguientes:

- Software modular: Facilita enormemente el desarrollo de proyectos complejos de robótica gracias a la existencia de extensas librerías con paquetes de código reutilizables. Gracias a estas librerías es posible incorporar de forma sencilla drivers para diferentes componentes, algoritmos de seguimiento, comandos de movimiento, etc.; aprovechando el conocimiento y años de experiencia de usuarios alrededor del mundo.

- Comunidad: Desde su creación en 2003, se ha convertido en un estándar en la industria y ha conseguido generar una comunidad de usuarios de gran tamaño que contribuyen con su trabajo y crean paquetes y librerías de uso público; algunas de ellas creadas y contrastadas por grandes profesionales de la robótica.

- Comunicación estandarizada: Se implementaron unos protocolos de comunicación estándar entre los diferentes elementos del sistema y para realización de transferencia de datos: TCPROS (el más usado) y UDPROS.

- Herramientas de desarrollo: Posee diferentes herramientas de desarrollo implementadas con el fin de permitir la monitorización, mejorar la solución de problemas y la posibilidad de visualizar los robots en entornos de simulación.

- Tecnologías de código abierto: Permite la incorporación de tecnologías externas de código abierto como:
    - MoveIt!: Software que permite la capacidad de realizar acciones de *motion planning*, incluyendo la manipulación de objetos y la navegación en entornos 3D.
    - Gazebo: Un motor físico de alta calidad que permite la realización de simulaciones visuales de uno o varios robots en sistemas físicos con características reales.
    - OpenCV: Una librería destinada a dar soporte en visión artificial (*computer visión*) incluyendo métodos para la adquisición, procesado y análisis de datos provenientes de imágenes o videos en tiempo real.

2.2.3.2 Estructura

ROS está compuesto, de forma simplificada, de una serie de procesos computacionales llamados **nodos** que llevan a cabo diferentes funciones independientes entre sí. Por ejemplo, particularizándolo para este proyecto: un nodo se encarga de controlar y publicar la información recogida por el LIDAR, otro nodo se encarga de publicar la información obtenida por la IMU, otro nodo se encarga del control de los motores, otro nodo se encarga de publicar la información obtenida por los encoders… La idea es la separación de tareas en nodos modulares que pueden ser incorporados a otros proyectos.

Para organizar la estructura de nodos en funcionamiento, existe un nodo especial llamado **Master** encargado del registro de todos los nodos. En él se guarda la información de los registros de los **topics** y los **services** de cada uno de los nodos de la red. Cada vez que se desea iniciar un nodo (*Publisher*, *Subscriber*, *Server* o *Client*), este se comunica con el *Master* para reportar su información de registro y recibir información sobre los demás nodos ya existentes, y los *topics* y *services* disponibles en la red. Ver Figura 13.

Existen 3 tipos de comunicación básica en ROS, los cuales se explicarán brevemente en los siguientes subapartados.

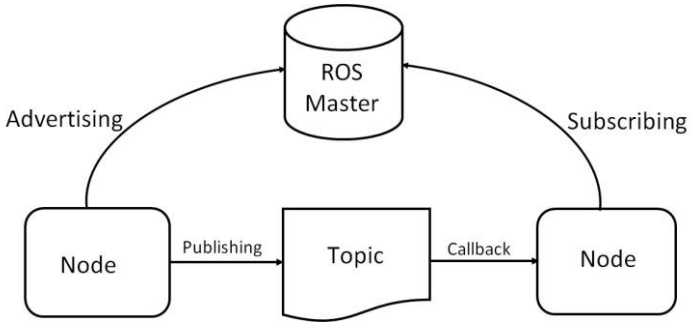


Figura 13: Función del ROS Master.

2.2.3.2.1 ROS Publisher/Subscriber

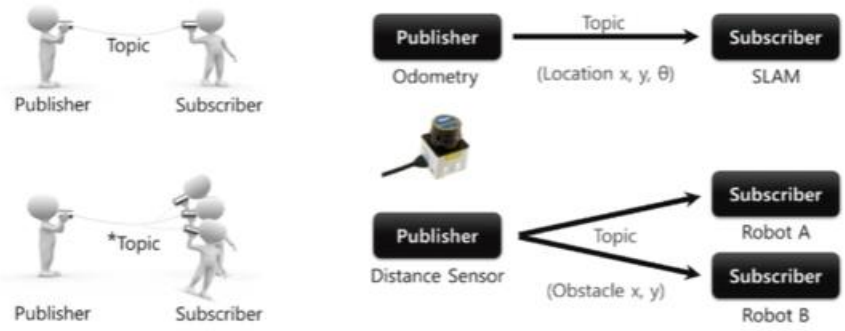


Figura 14: Estructura Publisher/Subscriber en ROS.

Como se puede apreciar en la Figura 14, este método de comunicación requiere como mínimo de la participación de dos nodos: uno que actúe como *Publisher* (publica información a un determinado *topic*), y otro que actúe como Subscriber (se suscribe y escucha información de un determinado *topic*). Un *topic* puede entenderse como un nombre que es utilizado para identificar el contenido de cierto mensaje; por ejemplo: velocidad de avance, localización, distancia recorrida… Cada *topic* lleva asociado un tipo característico de mensaje, que es una estructura de datos formada por tipos primitivos estándar (*integer, float, boolean* ...). En la Figura 14, se puede observar un ejemplo en el que un nodo tipo *Publisher* (llamado Odometry) publica información en un *topic* (llamado Location) mediante mensajes con una estructura de datos *x, y, θ*. Suscrito a dicho *topic*, recibiendo toda la información publicada por Odometry en forma de mensajes, se encuentra el nodo *Subscriber*

llamado SLAM. En ROS puede haber múltiples *Publishers* y *Subscribers* para un mismo *topic*; y un mismo nodo puede publicar y suscribirse a múltiples *topics*.

2.2.3.2.2 ROS Services

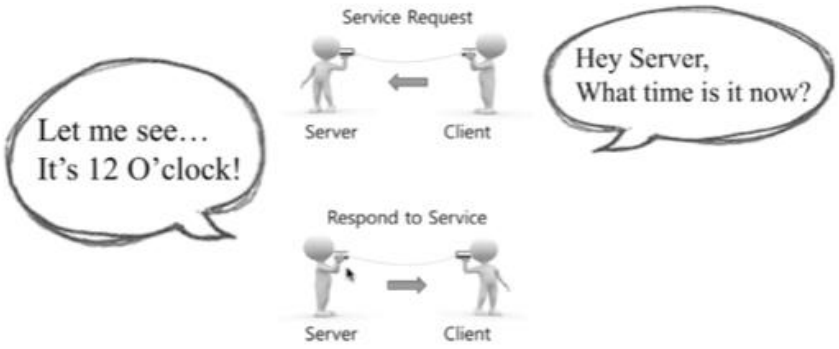


Figura 15: Estructura Services en ROS.

La comunicación *Publisher/Subscriber* es un tipo de comunicación unilateral (de *Publishers* a *Subscribers*) y que puede implicar a una gran cantidad de nodos a la vez. Sin embargo, en ciertas ocasiones es necesario una interacción tipo *petición – respuesta*, como la que puede apreciarse en la Figura 15. Este tipo de interacciones son llevadas a cabo en ROS a través de *Services*, estructuras definidas mediante dos mensajes: uno para el *Request* (petición), y otro para el *Reply* (respuesta). De esta forma, un nodo tipo *Server* ofrece un determinado servicio (*Service*) bajo un nombre concreto, y otro nodo tipo *Client* usa ese servicio enviando la petición (*Request*) propia de dicho servicio, y espera hasta que el *Client* le envíe su respuesta.

Para aclarar con un ejemplo, una estructura tipo *Service* podría estar compuesta de la siguiente petición y respuesta:

```
float32 width
float32 height
---
float32 area
```

En ella, el *Client* le envía al *Server* una *Request* compuesta de dos datos tipo *float32* que representan el ancho y largo de un rectángulo; y espera hasta que el *Server* le envía de vuelta la respuesta (*Reply*), compuesta por un dato tipo *float32* que representa el área. Su

funcionamiento es parecido a la llamada de funciones en C++, en las que se envían ciertos parámetros y se espera a que dicha función devuelva los parámetros de retorno.

2.2.3.2.3 ROS ActionLib

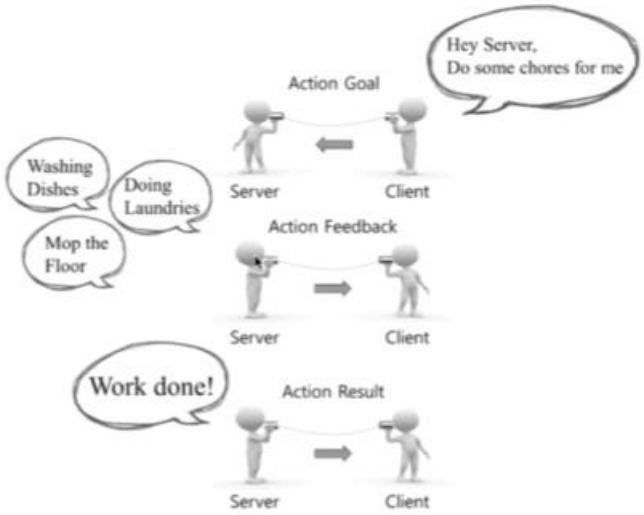


Figura 16: Estructura ActionLib en ROS.

El último modo de comunicación surge gracias a la comunidad de usuarios de ROS, que para resolver el tiempo de espera que sufre el *Client* en *Servicies* que requieren de cierto tiempo de procesado, desarrollaron lo que se conoce como *ActionLib*. La idea es similar a la de un *Service*, como el explicado en el apartado anterior, pero con la diferencia de que este proceso se realiza de forma asíncrona. En este caso, el *Client* manda una petición de servicio al *Server*, pero no se queda parado esperando una respuesta; sino que puede seguir realizando otros procesos de forma paralela, y es el *Server* el que notificará al *Client* cuando haya terminado de realizar el servicio.

2.3 VEHÍCULOS DE REFERENCIA

En esta sección se expondrán diversos dispositivos o robots utilizados ampliamente en la industria para el diseño y simulación de vehículos autónomos. Siendo el primero, el vehículo que se ha tomado como referencia a seguir para el desarrollo de este proyecto.

2.3.1 QCAR BY QUANSER

Este vehículo autónomo es el que se ha utilizado como referencia para desarrollar la instrumentación del vehículo del presente proyecto. Aunque existen ciertas diferencias entre el vehículo desarrollado por Quanser y el vehículo del proyecto; como el ordenador de a bordo utilizado, o la falta de cámara RGBD, la base es la misma y el software utilizado ha sido optimizado para las tareas que se buscaban desarrollar. [21]

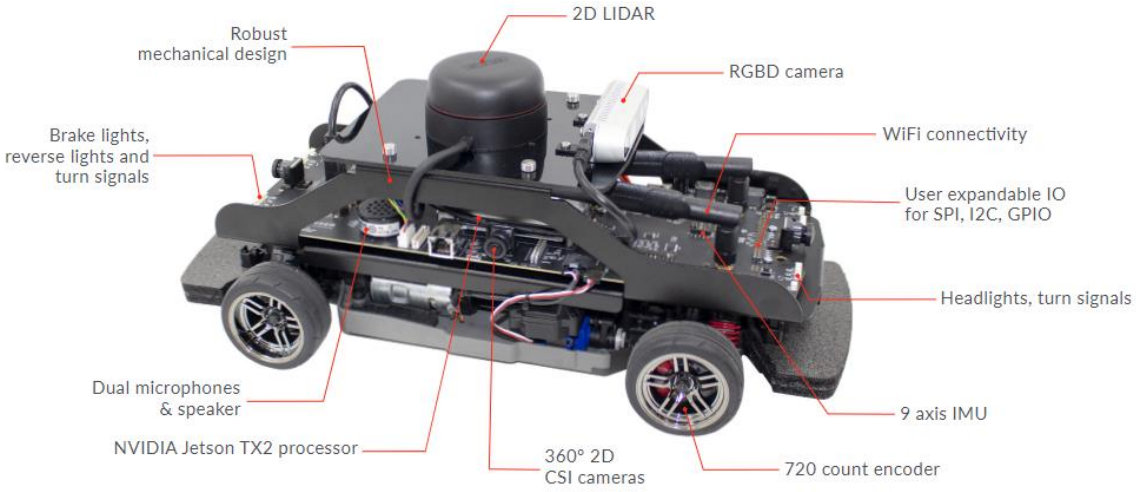


Figura 17: QCar desarrollado por Quanser.

Los componentes hardware más importantes del QCar mostrado en la Figura 17 son los siguientes:

- NVIDIA Jetson TX2 processor – Ordenador de a bordo.
- 2D LIDAR – Sensor LIDAR 2D.
- IMU inercial de 9 ejes.

- Intel D435 RGBD Camera – Stereo Depth Camera (Ver punto 2.1.5.1).
- Encoder de 720 cuentas en las ruedas motrices.

2.3.2 TURTLEBOT3 BY OPEN ROBOTICS

El TurtleBot3 es distribuido por Open Robotics, una organización independiente sin ánimo de lucro fundada por miembros de la Global Robotics Community. El primer Turtlebot, que se encuentra ya descatalogado, fue creado en Willow Garage por Melonee Wise y Tully Footy en noviembre de 2010. Willow Garage es un laboratorio de investigación robótica asentado en California, USA, que desarrolló el software de código abierto ROS (Robot Operating System) en el que se basa gran parte de este proyecto. Este particular robot, el TutleBot, es famoso por ser el primer robot incluido dentro del sistema ROS con el que era posible realizar simulaciones dentro del motor físico Gazebo, normalmente incorporado en ROS. [22] [23]

Debido a la importancia que ha tenido en la investigación y desarrollo de numerosos algoritmos de control, técnicas SLAM y de navegación autónoma, era imprescindible incluirlo en este apartado. A pesar de tener dos ruedas, es un robot cuyo hardware y funcionalidades son muy similares a las que se han buscado en este proyecto; y permite muchas posibilidades para el desarrollo de aplicaciones IA para vehículos autónomos.

Figura 18: TurtleBot3 modelo Burger.

Sus componentes hardware más importantes son los siguientes: [23]

- Raspberry Pi 3 Model B and B+ – Ordeandor de a bordo.
- IMU inercial de 9 ejes.
- LDS-01 – Sensor LIDAR 2D.
- Encoder en las ruedas motrices.

Capítulo 3. HARDWARE

3.1 DISPOSITIVOS HARDWARE DEL VEHÍCULO

El vehículo autónomo diseñado para el proyecto presenta la siguiente configuración de dispositivos hardware:

- Una *Raspberry Pi 3 Model B+*, que cumple la función de ordenador de a bordo y se encarga de procesar y enviar los datos procedentes de los sensores.
- Una placa de conexiones *Pi-EzConnect* fabricada por Alchemy Power Inc., que facilita la implementación de las conexiones de los diferentes pulsadores a los pines de la *Raspberry Pi 3 Model B+*.
- Una *Pi 3 Click Shield* fabricada por MikroElektronika, que facilita la conexión entre la IMU y la *Raspberry Pi 3 Model B+*.
- Una IMU de 6 ejes modelo *MPU-6000* incluida en un chip pequeño llamado *MPU IMU click* fabricado por MikroElektronika, que incluye un giroscopio de 3 ejes y un acelerómetro de 3 ejes.
- Un LIDAR 2D modelo *RPLIDAR A2M8*, que se encarga de la obtención de los datos de medida del entorno del vehículo.
- Dos drivers modelo *MD25* fabricados por Robot-Electronics, que controlan los motores y proporcionan información sobre los mismos.
- Cuatro motores *EMG30* fabricados por Robot-Electronics, que son los encargados de mover las ruedas motrices del vehículo autónomo.
- Una *batería LiPo* de tres celdas y 1300mAh y 11,1V de salida, que se encarga de alimentar todos los componentes activos presentes en el vehículo.

- Un *convertidor de tensión de 5V a 3,3V*, que permite las conexiones I2C entre los drivers de los motores y la *Raspberry Pi 3 Model B+*.

- Un *regulador de tensión de 12V a 5V*, que permite la alimentación, desde la batería de 11,1V, de la *Raspberry Pi 3 Model B+* y el *RPLIDAR A2M8*.

- Tres *pulsadores* fabricados por *RS PRO* con referencia: *734-6788*, que han sido conectados a pines lógicos de la *Raspberry Pi 3 Model B+* para la selección de diferentes estados de actuación del vehículo.

- Un *interruptor* de palanca SPST fabricado por *RS PRO* con referencia: *734-7154*, que permite dar y quitar tensión a los motores de forma independiente al resto del circuito por si fuese necesario en un caso de emergencia.

A continuación, en la Figura 19, se muestra un esquema sobre la estructura de conexiones utilizada para la instrumentación de los componentes hardware del vehículo. En dicho esquema, es preciso destacar que las conexiones en rojo corresponden a la transmisión de tensión, y que las conexiones en azul identifican la transmisión de datos entre los distintos componentes.

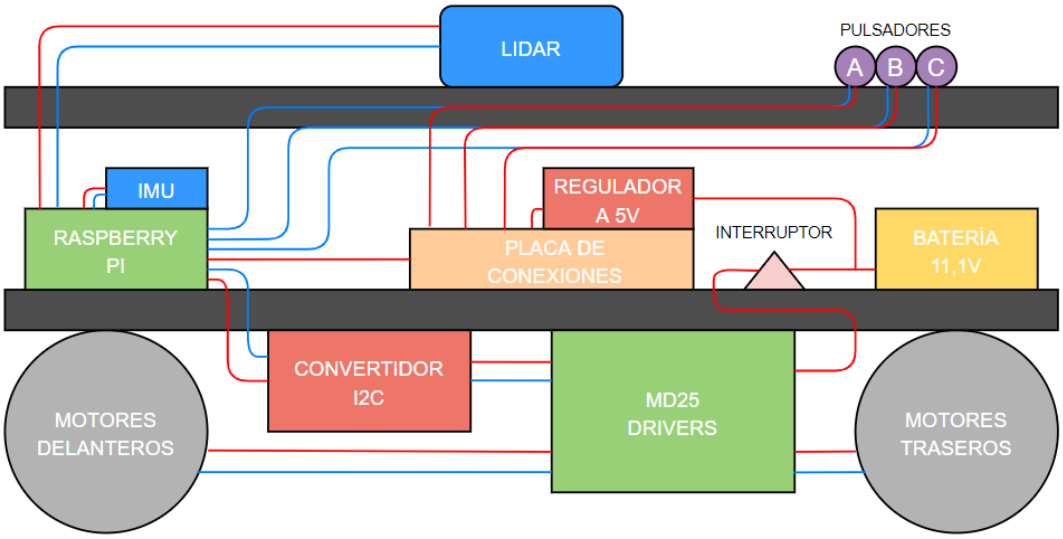


Figura 19: Esquema del conexionado del vehículo.

3.2 *RASPBERRY PI 3 MODEL B+*

La *Raspberry Pi 3 Model B+* es un modelo de Raspberry, que son unos dispositivos de bajo precio que reúnen las capacidades básicas de un ordenador en un tamaño muy reducido, comúnmente conocidos como SBC (*Single Board Computer*). En concreto, la Raspberry que se ha utilizado ha sido la 3ª generación en su modelo B+, lanzada al mercado por la *Raspberry Pi Foundation* en marzo de 2018. Este nuevo modelo fue la actualización natural de la anterior *Raspberry Pi 3 Model B*, incluyendo mejoras en el procesador y la conectividad. Entre sus características principales destacan las siguientes: [24]

Figura 20: Raspberry Pi 3 Model B+.

- Procesador *Broadcom BCM2837B0*, Cortex-A53 de 64-bit SoC @ 1,4GHz.
- GPU (unidad de procesamiento gráfico) modelo *Broadcom VideoCore IV*.
- Memoria RAM de 1GB tipo LPDDR2 SDRAM.
- 1 espacio para tarjeta Micro SD para la carga del sistema operativo y almacenamiento.

- Conectividad Wifi de doble banda 802.11.b/g/n/ac a 2,4GHz y 5GHz.
- Conectividad Ethernet con una velocidad máxima de 300Mbps.
- Conectividad Bluetooth 4.2 de bajo consumo energético.
- 40 pines de conexión GPIO (*General Purpose Input/Output*), que facilitan la conexión de diversos dispositivos hardware.
- 4 puertos USB 2.0, 1 puerto HDMI, 1 puerto MIPI DSI display port, 1 puerto MIPI CSI camera port para la instalación de un módulo de cámara desarrollado por la Raspberry Pi Foundation, y 1 puerto compartido para conectividad de audio por medio de 1 Jack de 3.5mm o 1 conector RCA.

3.2.1 ALTERNATIVAS EN EL MERCADO

En el mercado existen variedad de SBCs destinados a la investigación y desarrollo de proyectos tecnológicos como el vehículo que se va a instrumentar en este proyecto. A continuación, se presentarán las posibles alternativas que se podrían haber utilizado para la función de ordenador de a bordo del vehículo autónomo; y se expondrán las razones por las cuáles se tomó la decisión final de utilizar la *Raspberry Pi 3 Model B+*.

3.2.1.1 Raspberry Pi

La propia Raspberry Pi Foundation posee diferentes modelos, antiguos y recientes, que podrían haberse seleccionado para este proyecto. Entre los modelos más similares al escogido, y que reúnen la mayoría del hardware y conexiones descritas en el apartado 3.2, es posible encontrar principalmente dos modelos.

La *Raspberry Pi 3 Model B*, modelo inmediatamente anterior al escogido, que fue lanzado en 2016, y que posee características prácticamente idénticas al modelo seleccionado para este proyecto. Las principales diferencias que separan ambas versiones de la Raspberry Pi 3 es el salto de un procesador de 1,2GHz a una versión más potente de 1,4GHz. Esta diferencia de procesamiento ha sido uno de los factores decisivos a la hora de la selección del nuevo

modelo respecto al antiguo; aunque no ha sido el único ni el más primordial. La mejora más determinante por la cual se ha decidido utilizar la *Raspberry Pi 3 Model B+* para instrumentar el vehículo autónomo ha sido la renovada conectividad de red; pasando de una conexión única de 2,4GHz, a una conectividad Wifi de doble banda con posibilidad de acceder a las redes de 5GHz. Estas últimas, ofrecen conexiones mucho más rápidas, pasando de velocidades máximas de transmisión de datos de 50 o 60Mbps; a 867Mbps, en el caso de las frecuencias de 5GHz. [25] Este hecho ha resultado ser diferencial debido a la imposibilidad de conexión y transmisión de información entre los entornos software de Matlab – Simulink y ROS (*Robot Operating System*) utilizados durante el proyecto; cuestión que se tratará con más detalle en el apartado 5.2.2.

La *Raspberry Pi 4 Model B*, modelo inmediatamente superior al escogido, que fue lanzado en junio de 2019, y posee diferentes versiones con características notablemente superiores al modelo de Raspberry Pi escogido para este proyecto. Todas las versiones de la nueva *Raspberry Pi 4 Model B*, poseen un nuevo procesador que sube hasta los 1,5GHz y poseen una renovada GPU modelo *Broadcom VideoCore VI*. Donde existen diferentes opciones disponibles a la hora de elegir es en la capacidad de la memoria RAM incluida en la Raspberry, pudiendo escoger entre 2GB, 4GB u 8GB tipo LPDDR4-3200 SDRAM; que permiten una transferencia de datos significativamente más rápida (3200 MT/s)[3] que la antigua tecnología LPDDR2 (800 MT/s) incluida en las *Raspberry Pi 3*. [26] Esta mejora, justificaría la decisión de escoger cualquiera de las versiones de la nueva *Raspberry Pi 4 Model B*; ya que como se explica en el Capítulo 4. , surgieron problemas a la hora de compatibilizar e instalar diagramas de Simulink en ROS e incluso de instalar el propio ROS, que con la versión de 2GB (la más baja) no hubiesen ocurrido. El resto de las mejoras presentes en las nuevas *Raspberry Pi 4 Model B*, como son la capacidad de conexión a dos monitores 4K a 60fps a través de 2 puertos micro-HDMI, la inclusión de 2 puertos USB 3.0,

[3] MT/s: Millones de transferencias por segundo.

o el nuevo puerto de carga (pasa de micro-USB a USB tipo C), no son realmente importantes para el desarrollo de este proyecto. [27]

3.2.1.2 NVIDIA Jetson Nano

Una de las alternativas más interesantes a la Raspberry Pi es la *NVIDIA Jetson Nano*, lanzada por Nvidia en marzo de 2019. Aunque realmente es una competidora directa de la *Raspberry Pi 4 Model B 4GB* debido a la configuración hardware que ofrece. La *NVIDIA Jetson Nano* ofrece un diseño muy similar a la encontrada en cualquier modelo B de Raspberry Pi, aunque con algunas diferencias que se comentarán a continuación.

Figura 21: NVIDIA Jetson Nano.

Como se puede apreciar en la Figura 21, la diferencia más notable con los modelos B de Raspberry Pi es la inclusión de un disipador de calor pasivo en forma de aletas negras situado sobre la CPU y la GPU de la *NVIDIA Jetson Nano*. Esta implementación, aunque incrementa significativamente el rendimiento de la tarjeta, no es una diferencia insalvable; ya que existen módulos de ventiladores y disipadores por aletas disponibles para Raspberry Pi. La *NVIDIA Jetson Nano* posee un procesador *Cortex-A57 de 64-bit SoC @ 1,43GHz*,

ligeramente superior al de la *Raspberry Pi 3 Model B+* usada en este proyecto, y ligeramente inferior a cualquiera de las versiones de la *Raspberry Pi 4 Model B*. También ofrece una memoria RAM tipo LPDDR4 de 4GB, ligeramente más rápida que la encontrada en la versión 4 equivalente de Raspberry Pi.

Sin embargo, la diferencia más notable (a parte del precio, mostrado en la Tabla 1) se encuentra en la capacidad gráfica de su GPU *Maxwell de 128 núcleos* diseñada por NVIDIA. Esta GPU, ofrece rendimientos muy superiores, respecto a cualquiera de los modelos de Raspberry Pi comentados hasta el momento, a la hora de procesar *AI Frameworks* y modelos para aplicaciones como clasificación de imágenes, detección de objetos, segmentación, y procesamiento digital de voz. [28] Esto es debido a que la GPU que incorpora es compatible con CUDA, lo que hace posible, por ejemplo, el análisis y procesamiento de imágenes prácticamente a tiempo real; cosa que con cualquiera de las Raspberry Pi sería imposible. [29]

Es notable mencionar también que la *NVIDIA Jetson Nano* no posee conectividad Wifi ni bluetooth, como si ofrecen todos los modelos de Raspberry Pi; aunque esto es fácilmente solucionable a través de una tarjeta de red Intel o del uso de un USB Wifi. [30] Si el precio no es un problema, la elección de la *NVIDIA Jetson Nano* es muy recomendable, sobre todo si se pretende trabajar integrando cámaras en el proyecto; ya que además de lo mencionado anteriormente, existe una grande y creciente comunidad de ROS para la SBC de NVIDIA, siendo actualmente la SBC de moda para numerosos proyectos. [29]

3.2.1.3 Intel NUC Boards

Esta alternativa es especialmente interesante si se desea diseñar un vehículo autónomo integrando las cámaras específicas de Intel comentadas en el Apartado 2.1.5. Existen muchos modelos con una gran capacidad de procesamiento y que prácticamente son ordenadores al uso en tamaño reducido; pero debido a su elevado coste económico, en este apartado se va a tratar únicamente el modelo más modesto que ofrece la marca. Dicho modelo es el *Intel NUC Board NUC8CCHB*, que fue lanzado por Intel a finales del año 2019.

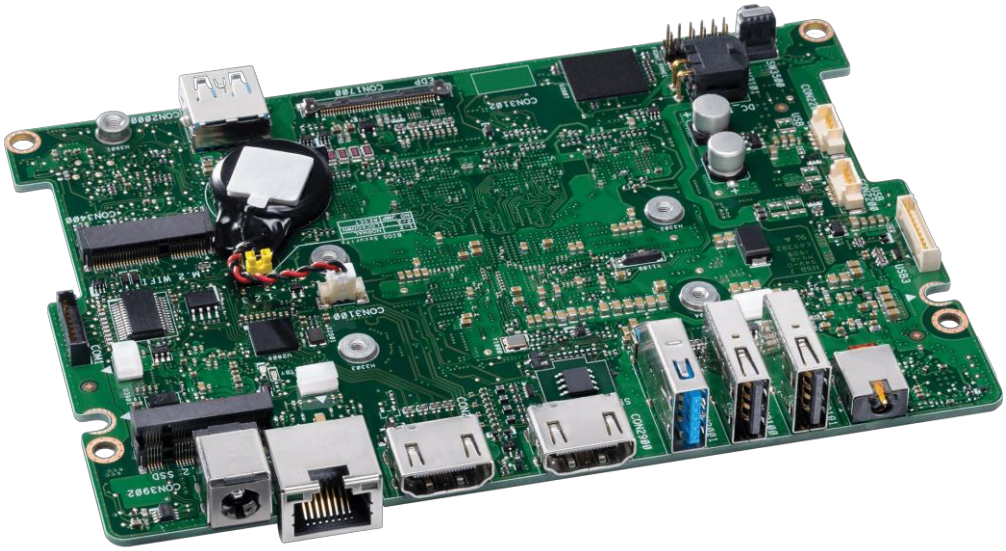


Figura 22: Intel NUC Board NUC8CCHB.

Como se puede apreciar en la Figura 22, la disposición es muy similar a las otras opciones que se han comentado hasta el momento; pero esta tarjeta esconde una característica que las demás no poseen, el software. Debido a su procesador *Intel Celeron Processor N3350* de doble núcleo de 64-bit @ 2,4GHz y su GPU integrada *Intel HD Graphics 500*, esta SBC es capaz de usar Windows 10 64-bit como cualquier ordenador doméstico. Esto, le confiere unas posibilidades que las demás tarjetas SBC comparadas hasta el momento no poseían; como la posibilidad de instalar el propio software de Intel dedicado a robótica, Matlab, Simulink, e incluso una máquina virtual o un USB booteable con Ubuntu y ROS. Posee una memoria RAM de 4GB tipo LPDDR3 y una memoria interna SSD tipo M.2 soldada a la placa de 64GB. [31]

Sin embargo, y a pesar de todo lo comentado anteriormente, su mayor ventaja respecto a las demás tarjetas SBC reside en el ecosistema que Intel ofrece en sus diferentes cámaras, módulos y LIDAR; especialmente diseñados para trabajar con sus propios micro-ordenadores, y que reciben actualizaciones automáticas de software y se encuentran perfectamente integrados dentro del ecosistema software. Una desventaja que tiene respecto a las tarjetas SBC comentadas anteriormente es que no posee pines de conexión GPIO, ni

interfaces de conexión UART, SPI o I2C; por lo que sería necesaria la conexión por medio de USB de alguna placa de conexión tipo Arduino o Raspberry. [32]

3.2.2 COMPARATIVA DE PRECIOS Y JUSTIFICACIÓN DE USO

En este apartado se presentará una tabla comparativa de los precios más bajos encontrados para cada una de las tarjetas SBC comentadas anteriormente[4]. A continuación se justificará la elección de la *Raspberry Pi 3 Model B+* para el proyecto tratado a lo largo de este documento.

Los precios para las diferentes opciones de SBC, así como los distribuidores donde se ha encontrado la oferta correspondiente, son los siguientes:

Precio de venta en España		
SBC	**Precios**	**Distribuidor**
Intel NUC Board NUC8CCHB	213.89 €	Mouser Electronics
NVIDIA Jetson Nano	116.04 €	RS Online
Raspberry Pi 4 Model B 4GB	64.99 €	PcComponentes
Raspberry Pi 3 Model B+	44.89 €	PcComponentes
Raspberry Pi 3 Model B	37.44 €	PcComponentes

Tabla 1: Comparativa de precios tarjetas SBC.

Observando los precios acuñados en la tabla superior, es claro advertir que las Raspberry Pi conllevan unos gastos menores respecto a las demás SBC consideradas. Dado que el proyecto iba a estar basado desde un principio en sensores que no necesitaban de un alto grado de procesamiento gráfico, debido a que las medidas y las técnicas utilizadas eran más exigentes con rendimientos de la CPU, se optó por probar en un primer momento con la *Raspberry Pi 3 Model B*.

[4] Precios correspondientes a España con fecha 25/07/2020.

En una primera parte del proyecto, consistente en la utilización de Matlab y Simulink que se verá en el apartado 4.2, la Raspberry Pi escogida cumplía con sus funciones asignadas sin ningún tipo de problema. Fue, sin embargo, en una segunda parte, en la que se decidió investigar la implementación de ROS con Matlab y Simulink, explicado en mayor detalle en el apartado 4.3, cuando se tuvo que dar un salto en la gama de Raspberry Pi y optar por el modelo inmediatamente superior, la *Raspberry Pi 3 Model B+*. Esto fue debido a la falta de conectividad Wifi a la banda de 5GHz, como ya se ha comentado anteriormente, hecho que resultó ser esencial para la correcta comunicación entre ROS y Simulink. Además, su mejora en procesador mejoró notablemente la experiencia de usuario a la hora de implementar Ubuntu 18.04, sistema operativo del que se hablará con más detenimiento más adelante.

3.2.3 CONEXIONADO

La *Raspberry Pi 3 Model B+* posee diferentes interfaces de conexión, siendo las siguientes las más destacables: UART, SPI, I2C, Wifi de doble banda, USB, puertos GPIO, Bluetooth 4.2 y Ethernet. Durante diferentes estados del proyecto, se ha hecho uso de la mayoría de estas comunicaciones, con excepción de las comunicaciones UART y Bluetooth. [24]

La alimentación de la Raspberry Pi ha de ser a 5V, y es posible de diversas formas: a través de un puerto micro-USB específico para carga, o a través de los pines GPIO. Durante el desarrollo del proyecto, solamente se ha utilizado el puerto de carga micro-USB. En la disposición final del proyecto, como puede observarse en la Figura 19, la Raspberry es alimentada a través de la batería LiPo, que ofrece una tensión de salida de 11,1V. Para poder realizar de forma segura la carga, se ha procedido a instalar un Regulador de Tensión de 12V a 5V, al que se ha soldado un conector micro-USB a su salida de baja tensión.

El protocolo de comunicación empleado entre la Raspberry Pi y el ordenador ha variado en cada una de las dos fases en las que se ha trabajado en este proyecto, explicado a lo largo del Capítulo 4. En concreto:

- En una 1ª fase en la que se trabaja exclusivamente con Matlab y Simulink, la comunicación entre ambos dispositivos se ha llevado a cabo a través de Wifi

mediante el protocolo de comunicación TCP/IP y el protocolo de mensajes Mavlink; que juntos, permiten el envío y recibo de información desde una estación base (El ordenador con Matlab y Simulink, en este caso). [33]

- En una 2ª fase en la que se implementa el uso de Simulink y ROS, la comunicación entre ambos dispositivos se ha llevado a cabo a través de Wifi mediante el protocolo de comunicación TCP.

Las conexiones con el resto de los componentes hardware serán descritas en los apartados desarrollados específicamente para cada uno de ellos en las secciones siguientes. Sin embargo, sí que se realizará una aclaración sobre el orden de conexión de la *Pi 3 Click Shield* y de la *Pi-EzConnect*, conectadas a través de los puertos GPIO en el orden mostrado en la siguiente figura:

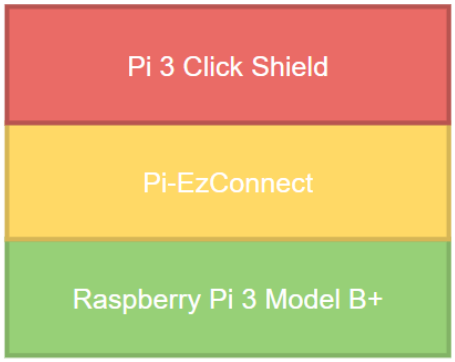


Figura 23: Orden de conexión de placas.

3.3 PI-EZCONNECT

La *Pi-EzConnect* es una placa de conexión fabricada por Alchemy Power Inc. Que facilita las conexiones de periféricos a los puertos GPIO de la Raspberry Pi. Es montada inmediatamente encima de la Raspberry Pi y ofrece una breadboard de pequeño tamaño donde es posible la soldadura de las diferentes conexiones a los puertos GPIO, facilitando así la conexión de diferentes sensores, resistencias y capacitores. Además de esto, ofrece varios buses de tensión a 5V y 3,3V, así como buses para la conexión a tierra. Si durante la

instrumentación del proyecto se prefiere no utilizar la breadboard disponible para la soldadura, esta placa ofrece también conexión a todos los GPIO y buses descritos con anterioridad a través de bloques de conexión fijados por tornillos. [34]

Todas las posibilidades anteriores pueden comprobarse en la siguiente Figura:

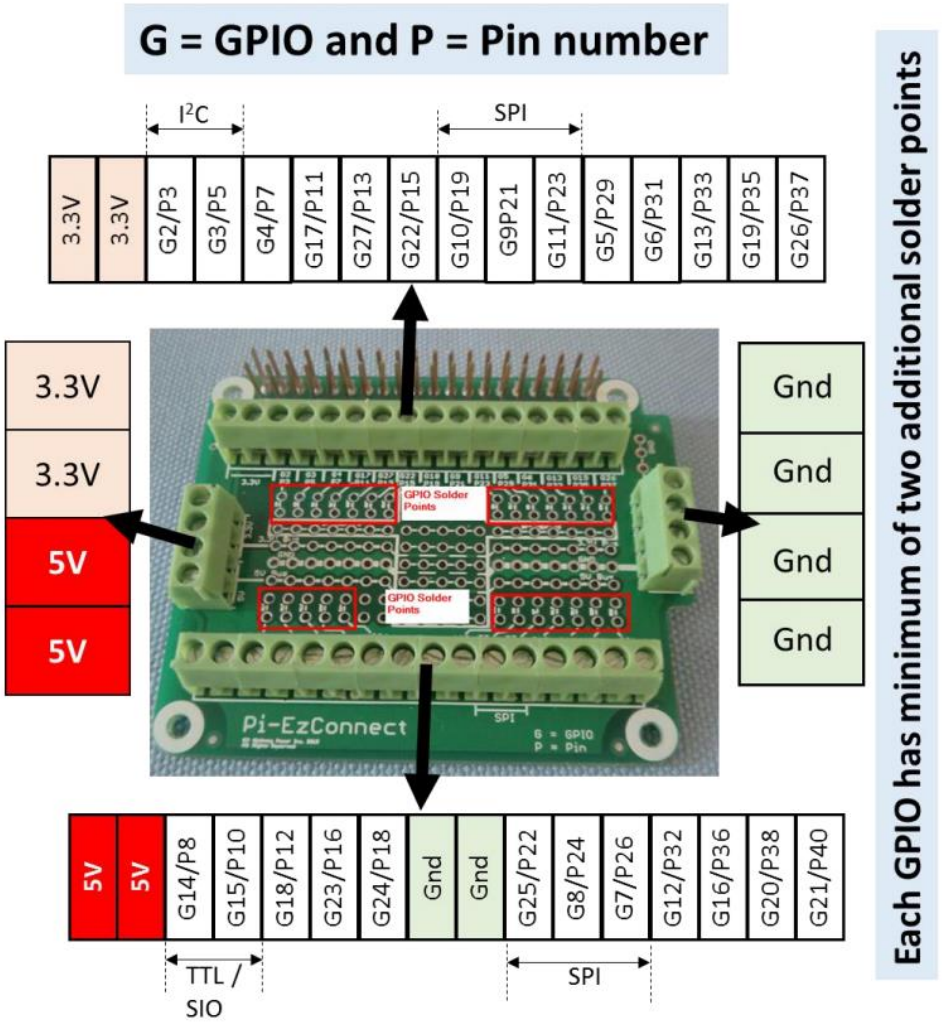


Figura 24: Detalles de conexión para la Pi-EzConnect.

En este proyecto, para asegurar el correcto funcionamiento de la instrumentación realizada, se ha optado por soldar todas y cada una de las conexiones realizadas a la *Pi-EzConnect*; las cuales se irán describiendo a medida que se desarrollen los diferentes componentes hardware.

La utilización de la placa *Pi-EzConnect* es muy importante a la hora de realizar proyectos debido a la facilidad con la que permite conectar componentes a la Raspberry Pi. Además, ofrece la posibilidad de trabajar en varios proyectos a la vez con la misma Raspberry Pi, sin necesidad de desmontar todas las conexiones establecidas en ella; simplemente se desconecta de la Raspberry Pi y se puede trabajar en otro proyecto con otra placa de conexiones guardando la instrumentación elegida para el proyecto anterior.

3.4 PI 3 CLICK SHIELD

La *Pi 3 Click Shield* es una placa fabricada por MikroElektronika que permite la conexión de cientos de *click boards* gracias a la incorporación de dos conexiones mikroBUS. Esto facilita enormemente la conexión de diferentes sensores, incorporados en *click boards*, sin necesidad de atornillar ni soldar ninguna de sus conexiones. [35] Además, garantiza la correcta conexión de todos los componentes añadidos debido precisamente a que el usuario solo tiene que preocuparse de introducir los sensores en el mikroBUS correspondiente, como puede observarse en la Figura 25:

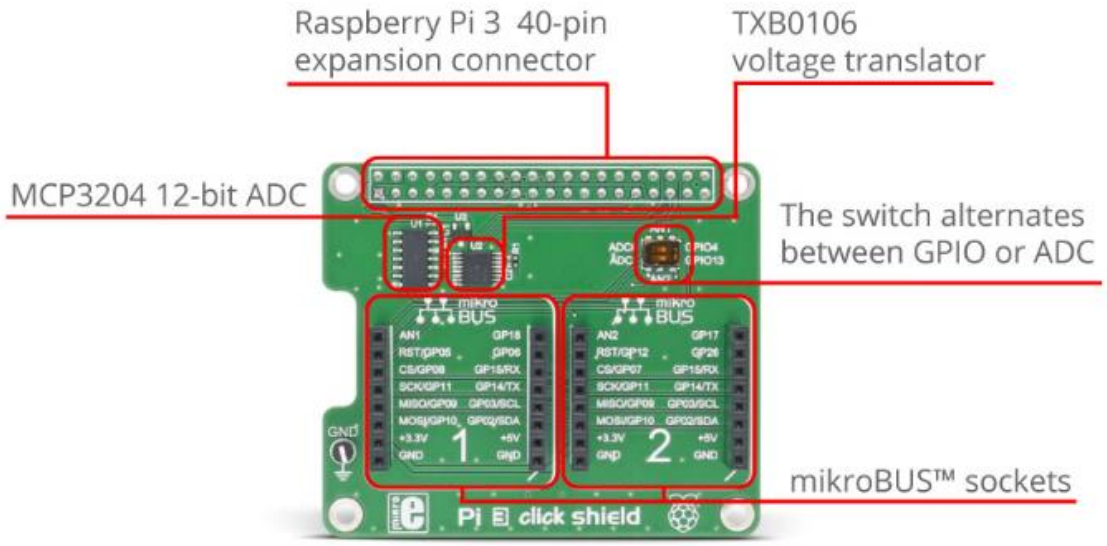


Figura 25: Esquema de conexiones de la Pi 3 Click Shield.

La *Pi 3 Click Shield* es compatible con las *Raspberry Pi 3 Model B/B+*, *Raspberry Pi 2 Model B*, y *Raspberry Pi 1 Model A+/B+*. Una de las características más importantes de esta placa de conexiones es la incorporación de un convertidor ADC, que permite la medida de datos procedentes de sensores analógicos, función que la Raspberry Pi no presenta al no poseer puertos GPIO analógicos en su configuración de pines. [35]

3.5 MPU-6000

En este proyecto, se ha decidido incorporar el modelo MPU-6000 de IMU integrada en una de las *click boards* comentadas en el apartado anterior fabricadas por MikroElektronika. Como ya se explicó en el Estado del Arte en los apartados 2.1.1 y 2.1.1.1, la IMU es un sensor fundamental para medir la orientación y la posición del vehículo del proyecto.

El modelo elegido de MikroElektronika es el llamado *MPU IMU click*, que integra la IMU MPU-6000 de 6 ejes. Ver Figura 26. Estos, se encuentran en una configuración basada en un acelerómetro de 3 ejes ortogonales en el espacio, y un giróscopo de 3 ejes homólogamente ortogonales. Además, también incorpora un DMP (*Digital Motion Processor*) capaz de procesar los diferentes datos provenientes del acelerómetro y el giroscopio. El *MPU IMU click* incorpora capacidades de comunicación con la Raspberry Pi a través de SPI, I2C, RST y líneas INT; y está diseñada para ser alimentada a los 3,3V que proporcionan los pines de la Raspberry Pi.

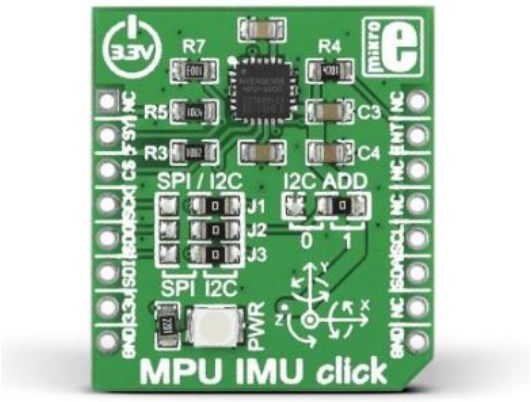


Figura 26: MPU IMU click.

3.5.1 CONEXIONADO

El conexionado de este sensor es bastante directo gracias a la configuración mikroBUS ideada por MikroElektronika. Sin embargo, los protocolos de comunicación han de ser ajustados a través de resistencias, tal y como se puede apreciar en la Figura 26. En el desarrollo del proyecto, se han utilizado dos protocolos de comunicación diferentes:

- SPI: Este protocolo ha sido utilizado en la primera parte del proyecto al trabajar exclusivamente con Matlab y Simulink; debiendo soldar las tres resistencias contenidas en el segmento SPI/I2C localizables en la Figura 26 a las conexiones específicas para SPI.

- I2C: Este protocolo permite la conexión de diferentes dispositivos a través de únicamente 2 cables comunes a todos los componentes entre los que se desea establecer comunicaciones. El I2C ha sido utilizado en la segunda parte del proyecto al integrar Simulink y ROS; y análogamente al SPI, es necesario soldar las resistencias a las conexiones específicas para I2C.

3.5.2 FUNCIONAMIENTO

Es necesario aclarar, que aunque se disponga de un acelerómetro y un giroscopio en cada uno de los tres ejes cartesianos; debido a que el vehículo instrumentado en este proyecto posee únicamente un movimiento en el plano XY, tan solo se han utilizado las medidas obtenidas por el acelerómetro en los ejes X e Y, y la medida obtenida por el giroscopio en el eje Z. Estas medidas son suficientes para la representación del movimiento efectuado por el vehículo.

3.6 LIDAR

Durante el desarrollo de este proyecto, se ha decidido implementar dos versiones del modelo RPLIDAR fabricado por Slamtech, el modelo RPLIDAR A1M8 y el modelo RPLIDAR A2M8. Ambos modelos han funcionado correctamente y, como era de esperar, el modelo A2M8 proporcionó mejores datos de lectura que el modelo A1; a pesar de su menor rango de alcance. El modelo A1M8 presenta un alcance de 0,15-12m, mientras que el modelo A2M8 ofrece alcances de 0,15-6m. Sin embargo, el hecho diferencial por las que las medidas tomadas resultan ser mejores en el modelo de menor alcance es debido a su mayor frecuencia de escaneo, con valores típicos de 10Hz para el A2M8 y de 5,5Hz para el A1M8; lo que se traduce en un mayor número de muestras por revolución y una resolución angular ligeramente inferior. Debido a que el vehículo de este proyecto se encuentra enfocado a la navegación en interiores, es preferible el uso del modelo A2M8 sobre el A1M8 debido a la mayor precisión y puntos de información que ofrece; a costa de sacrificar un alcance que, en espacios interiores, no supone ningún beneficio. [36] [37]

Figura 27: RPLIDAR A2M8.

Como ya se ha comentado con anterioridad, la instrumentación del LIDAR es una de las partes más importantes de este proyecto; ya que permite obtener la medida de la distancia

relativa al vehículo de los diferentes objetos de su entorno, información esencial para la integración de sistemas SLAM.

Debido a la dificultad presente en la integración de un driver funcional del LIDAR en Matlab y Simulink, se optó por la utilización del sistema ROS para obtener los datos del sensor; decisión que marcaría la separación del proyecto en las dos fases comentadas anteriormente.

3.6.1 CONEXIONADO

El RPLIDAR A2M8 usa únicamente un conector macho tipo XH2.54-5P, el cual puede observarse en la Figura 28.

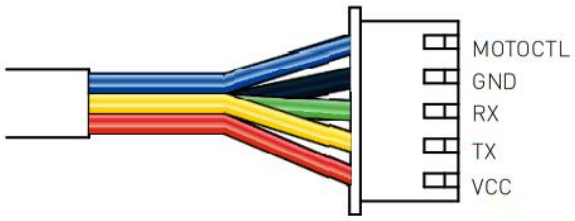


Figura 28: Esquema del conector XH2.54-5P.

Las características de los diferentes cables pertenecientes al conector son las siguientes:

Color	***Señal***	***Descripción***	***Valor Típico***
Rojo	VCC	Entrada de alimentación	5V
Amarillo	TX	Salida del puerto serie del procesador del LIDAR	0V / 3,3V
Verde	RX	Entrada del puerto serie al procesador del LIDAR	0V / 3,3V
Negro	GND	Tierra	0V
Azul	MOTOCTL	Señal PWM de control de la velocidad de rotación del LIDAR	0V - 5V

Tabla 2: Definición de las señales de interfaz externa del LIDAR.

Como se puede apreciar en la Tabla superior, la interfaz de comunicación utilizada por el LIDAR es a través de puerto serie (UART); y posee una señal específica para el control de la velocidad de rotación de su motor a través de un PWM en un rango de voltaje de 0V a 5V. Además, para facilitar la experiencia de usuario, el RPLIDAR A2M8 incorpora un detector de velocidad de rotación y un control adaptativo en su propio procesador, que ajusta automáticamente la resolución angular de acuerdo con la velocidad empleada. Dicha velocidad podrá ser ajustada hasta alcanzar un máximo de 4000 muestras por segundo. [37]

El RPLIDAR A2M8 utilizado, facilita una tarjeta conectora para transformar el puerto XH2.54-5P descrito con anterioridad a una entrada micro-USB; facilitando así el uso y la conexión del LIDAR con el ordenador o la Raspberry Pi.

3.7 DRIVERS MD25

El MD25 es un driver robusto fabricado por Robot-Electronics pensado para controlar dos motores mediante conexión en serie o I2C. Se encuentra especialmente diseñado para trabajar con los motores EMG30, pertenecientes al mismo fabricante, y los que han sido utilizados en la instrumentación del vehículo de este proyecto.

El MD25 es capaz de controlar dos motores de manera conjunta o de forma totalmente independiente. Debido a la configuración de 4 ruedas que posee el vehículo diseñado en este proyecto, ha sido necesario el uso de dos drivers MD25 para realizar el control de la totalidad de los motores. Estos drivers, permiten también la lectura de la corriente en el motor; así como de los encoders localizados en los motores. La resolución de los encoders integrados en los motores es de 360 pulsos por revolución, y se accederá al registro asociado en el driver cada 1 ms. [38]

3.7.1 CONEXIONADO

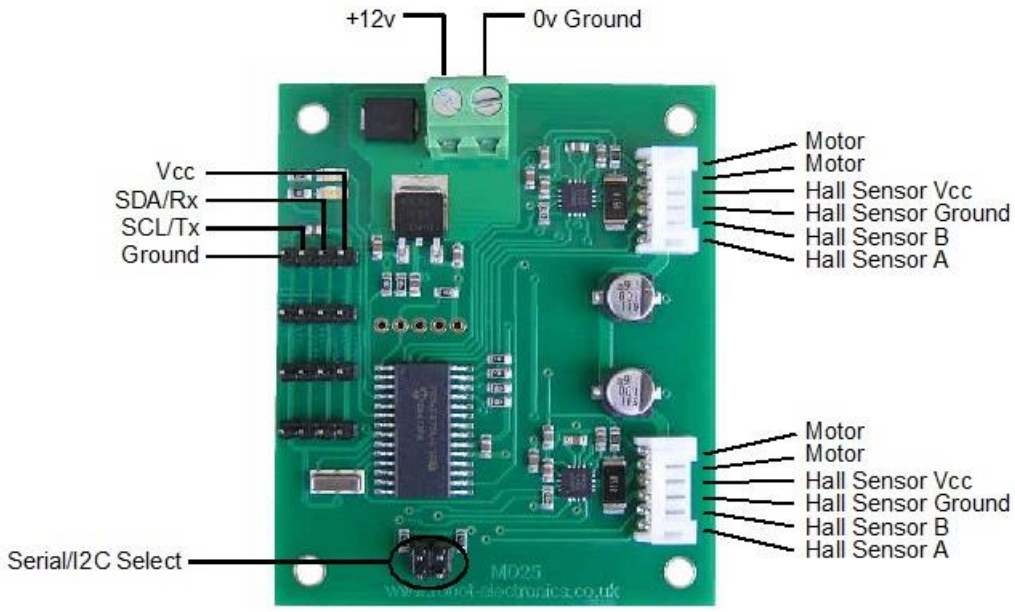


Figura 29: Estructura de las conexiones del MD25.

Como se puede apreciar en la Figura 29, el conexionado de los drivers MD25 consta de cuatro partes: [38]

- La alimentación: Tiene lugar gracias a los conectores situados en la parte superior de la Figura 29. Ambos puertos son conectados a través de sus respectivos cables, a los polos correspondientes de la batería de 11,1V utilizada para el proyecto; pasando antes por un interruptor, que se encargará de permitir la desconexión independiente de los motores en caso de emergencia.

- Selección de protocolo de comunicación: En la parte inferior de la Figura 29 se localizan dos pares de pines que podrán ser ajustados mediante el uso de jumpers para establecer el protocolo de comunicación con el ordenador de a bordo deseado (Raspberry Pi 3 Model B+ en este caso). El usuario podrá elegir entre comunicación I2C a 100kHz de reloj interno, y comunicación en serie a 9600 bps, 19200 bps o 38400 bps. Para este proyecto, se ha decidido utilizar el protocolo I2C de

comunicación, para el cual no es necesaria la colocación de ningún jumper en la zona señalada anteriormente.

- Puertos de conexión con los motores: Cada uno de los dos motores controlados por cada driver, son conectados a través de los puertos blancos de 6 pines localizados en la parte derecha de la Figura 29. Dichos conectores vienen incluidos en los motores EMG30 utilizados para este proyecto y poseen las conexiones de alimentación, control y lectura de los diferentes sensores del motor.

- Conexión de comunicación: La comunicación con la Raspberry Pi es llevada a cabo a través de las series en paralelo de 4 pines localizadas en la parte izquierda de la Figura 29. Ambos drivers MD25 se han conectado en paralelo a la Raspberry Pi conectando los 4 pines de cada driver a través de un mismo cable. Las conexiones salientes de las tarjetas MD25 pertenecientes a la comunicación I2C (SDA y SCL) presentan una lógica de tensión a 5V, incompatible con los 3,3V que es capaz de leer la Raspberry Pi. Por ello, ha sido necesaria la implementación de un convertidor de tensión de 5V a 3,3V para poder realizar de forma efectiva dicha conexión. Los cables de Vcc y Ground han sido conectados a los buses habilitados para dichas funciones en la Pi-EzConnect. Por otro lado, las conexiones SDA y SCL han sido conectadas, respectivamente, a los pines G2 y G3 de la Pi Ez-Connect (correspondientes a los GPIO 2 y GPIO 3 de la Raspberry Pi). Ver Figura 24.

3.8 MOTORES EMG30

Los motores EMG30 fabricados por Robot-Electronics han sido diseñados específicamente para ser utilizados en conjunción con los drivers MD25 usados también en este proyecto. Estos motores son alimentados a 12V a través de los drivers y se encuentran equipados con encoders magnéticos de efecto Hall; que resultan mucho menos sensibles a la contaminación ambiental y al exceso de calor que los otros tipos de encoders existentes en el mercado (ópticos y mecánicos), ofreciendo así, medidas más fiables. [7]

Figura 30: Motor EMG30.

Los motores EMG30 utilizados para la instrumentación del vehículo de este proyecto, los cuales pueden ser vistos en la Figura 30, presentan una caja de reducción de 30:1; haciéndolos ideales para proyectos robóticos de pequeño y mediano tamaño. Además, vienen equipados con un condensador estándar de supresión de ruido localizado a lo largo de los devanados del motor. Los EMG30 poseen un rango de velocidades comprendido entre 1,5 – 200 rpm; y sus encoders son capaces de realizar 360 cuentas por revolución. [39]

3.8.1 CONEXIONADO

Cada motor EMG30 presenta un conector macho de 6 pines que es conectado directamente a los puertos hembra correspondientes localizados en os drivers MD25. Como se puede apreciar en la Figura 29, los dos primeros pines corresponden a la señal PWM dedicada al control de los motores, los dos pines siguientes corresponden a la alimentación y la conexión a tierra de los encoders de efecto Hall, y los dos últimos corresponden a la medida obtenida de dichos encoders. El orden seguido para la descripción de cada uno de los pines presentes en la conexión es la facilitada en la Figura 29 en sentido de lectura de arriba abajo. [39]

3.9 BATERÍA LIPO DE 11,1V

La batería escogida para la instrumentación de este proyecto ha sido una batería LiPo (Litio-Potasio) con una tensión de salida de 11,1V. La batería presenta una configuración de 3 celdas, que posibilitan una tensión de salida (11,1V) adecuada a los 12V que exigen los motores EMG30 seleccionados en este proyecto. Presenta una capacidad de carga total de 1300mAh y una descarga continuada máxima de 45C (58,5A). [40]

Figura 31: Batería LiPo de 11,1V y 1300mAh.

La función principal de la batería mostrada en la Figura 31 es la de alimentar todos los componentes presentes en el vehículo autónomo instrumentado para este proyecto. Dicha alimentación se discutirá con mayor detalle en el apartado siguiente.

3.9.1 CONEXIONADO

Como se puede apreciar en el esquema mostrado en la Figura 19, los dos polos de la batería siguen dos caminos principales.

En un primer camino, ambos polos llegan hasta un interruptor que permite dar y cortar tensión a los drivers de los motores de forma independiente al resto del circuito, con la intención de poder actuar sobre los mismos en caso de emergencia.

En un segundo camino, ambos polos llegan hasta un regulador de tensión de 12V a 5V con el fin de poder alimentar a la Raspberry Pi, que a su vez se encarga de alimentar el resto de los componentes del vehículo. Para ello, a la salida del regulador de tensión mencionado previamente, se ha adaptado un conector micro-USB, coincidiendo así el tipo de conector con la fuente de entrada de alimentación predeterminada de la Raspberry Pi.

3.10 PULSADORES

Figura 32: Pulsador RS Pro Ref: 734-6788.

En este proyecto, se han decidido implementar tres pulsadores fabricados por *RS PRO* con referencia: *734-6788*. Estos pulsadores de botón, los cuales pueden ser apreciados en la Figura 32, se han integrado en el vehículo para permitir al usuario seleccionar el paso de un estado a otro dentro de una máquina de estados, la cual se explicará en mayor profundidad en el apartado 4.2.2.1.

Los pulsadores elegidos son de acción momentánea y han sido diseñados específicamente para ser integrados a través de un encaje a presión en placas PCB o por soldadura. El interruptor integrado posee una vida mecánica de 50.000 ciclos, y tiene una tensión de

contacto nominal en DC de 28V; los cuales son más que suficientes para cumplir con las características del circuito del vehículo de este proyecto. [41]

3.10.1 CONEXIONADO

Como puede apreciarse en la Figura 33, el pulsador escogido posee tres pines que actúan de la siguiente manera:

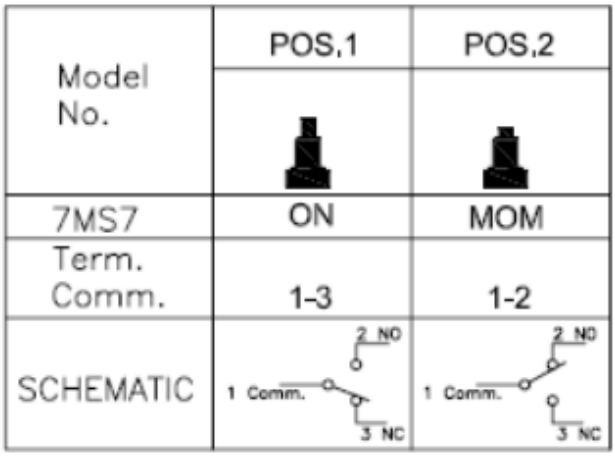


Figura 33: Esquema de funcionamiento de los pulsadores.

El pin con nombre *Comm.* será el pin de salida de los pulsadores, que llegará hasta los puertos GPIO5, GPIO6 y GPIO26 de la Raspberry Pi transmitiendo el valor de salida correspondiente a los pulsadores A, B y C, respectivamente. Esta conexión se realiza directamente sobre la placa de conexiones que la *Pi-EzConnect* ofrece para tal uso, como ya se comentó en el apartado 3.3.

Se ha decidido utilizar el pin *NC* como conector a tierra y el pin *NO* como conector a 5V; de esta forma, el puerto lógico de la Raspberry Pi recibirá siempre un valor de 0 proveniente de cada uno de los pulsadores hasta que se decida pulsar el botón de alguno de ellos, momento en el que el puerto asociado a dicho pulsador recibirá un valor igual a 1.

3.11 INTERRUPTOR

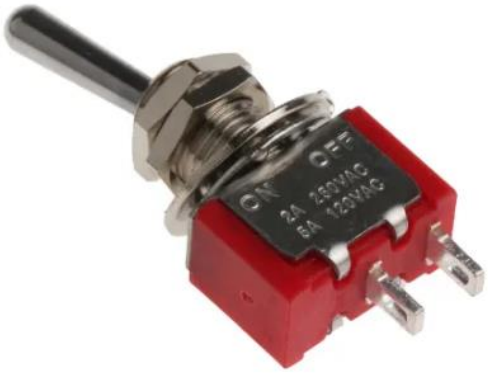


Figura 34: Interruptor de palanca RS PRO Ref: 734-7154.

Para la instrumentación del vehículo llevada a cabo en este proyecto, se ha decidido incluir un interruptor de palanca fabricado por RS PRO, el cual puede apreciarse en la Figura 34. Como ya se ha comentado en los apartados anteriores, la decisión de implementar un interruptor entre la batería y la alimentación de los drivers MD25 que alimentan a los motores es por seguridad. Gracias a esta inclusión, la alimentación de los motores puede habilitarse o deshabilitarse de forma manual por el usuario en caso de que así lo crea necesario. [42]

3.12 REGULADOR Y CONVERTIDOR DE TENSIÓN

Como ya se ha mencionado en los apartados anteriores, para poder llevar a cabo una alimentación adecuada a los valores que demanda cada uno de los componentes, ha sido necesaria la inclusión de un regulador de tensión de 12V a 5V y un convertidor de tensión de 5V a 3,3V. La función que cumplen cada uno de ellos ha sido ya discutida en los apartados anteriores, y su conexión y localización en el circuito eléctrico del coche puede ser comprobada en la Figura 19.

Capítulo 4. SOFTWARE

En este capítulo se explican las diferentes opciones consideradas para el desarrollo de los drivers de los sensores y actuadores que han sido instrumentados en el vehículo del proyecto. Se introducen las diferentes ventajas e inconvenientes de cada una de las opciones; y para facilitar la lectura y comprensión de los drivers utilizados a lo largo de este proyecto, se ha procedido a la separación del presente capítulo en dos partes bien diferenciadas, correspondientes con las dos formas de implementación y diseño de los drivers.

4.1 DRIVERS: MATLAB-SIMULINK VS ROS

Las dos opciones para la implementación y desarrollo de drivers que se han llevado a cabo a lo largo del proyecto han sido las siguientes:

- Matlab-Simulink con el paquete de soporte para Raspberry Pi: este es el procedimiento habitual que se lleva realizando en el laboratorio de la universidad los últimos años. Estos drivers, ofrecen mucho control sobre las medidas y su tratamiento debido al elevado conocimiento que se tiene de ambos entornos, además de la cantidad de funcionalidades y elementos de programación visuales (bloques de Simulink) que en ellos se ofrecen, facilitando en gran medida la implementación de dichos drivers. Sin embargo, el problema reside en los drivers de elementos hardware más complejos y sofisticados, que requieren de un tiempo de desarrollo elevado y unos niveles de dificultad altos. Para su funcionamiento es necesario la instalación de una imagen de Raspberry Pi OS en la Raspberry Pi.

- ROS: es el procedimiento que se está explorando en el momento de la elaboración de este proyecto. Como ya se ha comentado en el apartado 2.2.3.1, este entorno lleva casi dos décadas de desarrollo y tiene detrás una extensa comunidad que comparte de forma totalmente abierta muchos de sus trabajos. Al tener una gran cantidad de software modular disponible en las redes, es relativamente sencillo encontrar drivers

completamente funcionales y contrastados en varios proyectos de los elementos hardware que se deseen integrar. De esta forma, no habría que desarrollar el driver desde cero como ocurre en Matlab-Simulink, sino que habría que adaptarlo a las necesidades propias del proyecto que se esté realizando. Dentro del uso de ROS se han barajado diferentes formas de implementación de los drivers:

- ROS en Raspberry Pi OS: Esta opción supone la instalación de ROS *Melodic Morenia* en la imagen propia de Raspberry Pi OS. Lo que se busca con esta solución es comprobar si es posible generar un paquete ROS desde Simulink haciendo uso de los paquetes de soporte de Raspberry Pi y ROS Toolbox presentes en Simulink. Gracias a esto, la implementación de diferentes sensores disponibles en el laboratorio de la universidad, o cualquier driver desarrollado en Simulink debería poder ser volcado en la Raspberry Pi haciendo uso del paquete de soporte para acceder a los diferentes pines GPIO o registros de conexión establecidos en Simulink a través de un nodo ROS. La instalación de ROS en Raspberry Pi OS se encuentra convenientemente explicada en el ANEXO II.

- ROS en Ubuntu: Esta opción contempla el aprendizaje profundo del entorno de programación para la creación de paquetes en ROS, o la utilización de algunos de los paquetes disponibles en internet. En esta opción, sería necesario programar absolutamente todas las conexiones (I2C, SPI, GPIO…) en C++ o Python, que son los lenguajes de programación disponibles en ROS; a parte de los propios drivers. Es el método que se antoja más complejo y con un tiempo de aprendizaje más elevado. En esta opción la Raspberry Pi se ha probado con Ubuntu Mate, tal y como se explica en el apartado 4.3.1. Su instalación se encuentra también convenientemente explicada en el ANEXO I.

- ROS con un Arduino como concentrador hardware: En esta opción se utiliza un Arduino para conectar los sensores y leer los datos proporcionados por los mismos. Dichos datos son enviados a la Raspberry Pi a través de una

conexión en serie y publicados mediante un nodo ROS en un determinado *topic*. Una vez conseguido esto, se puede realizar el tratamiento de datos desde Simulink o volcar un paquete a la Raspberry Pi desde Simulink para realizar dicho tratamiento de datos.

4.2 *MATLAB Y SIMULINK*

El trabajo realizado durante una primera fase del proyecto se encuentra comprendido casi en su totalidad en Simulink, pero es necesario mencionar que todos los ficheros y diagramas contenidos en él pueden funcionar debido a los numerosos scripts de Matlab que definen gran parte de lo que sucede en ellos. Debido a esto, se ha decidido que ambos programas compartan el nombre de este apartado; aunque realmente se va a hablar principalmente sobre Simulink, y tan solo se hará mención a Matlab cuando sea necesario realizar ciertas aclaraciones.

4.2.1 ESTRUCTURA JERÁRQUICA DE SIMULINK

Con el fin de entender de forma correcta el trabajo de instrumentación que se ha realizado a través de Simulink, es necesario adquirir una visión global de la estructura dentro de la cual se va a realizar la incorporación de dichos elementos.

Existen dos ficheros principales de Simulink: un primer fichero, destinado a ser ejecutado por el PC del usuario, donde se realizan intercambios de información con la Raspberry Pi, ordenador de a bordo del vehículo; y otro fichero, destinado a ser ejecutado por la propia Raspberry Pi, que será el encargado de monitorizar y controlar los elementos activos del vehículo según los controles y parámetros escogidos. Este proyecto se centra en el segundo fichero, que es el que incorpora los drivers pertenecientes a la instrumentación de los sensores y los actuadores escogidos para este proyecto. La estructura de este último fichero es la siguiente:

En un primer nivel de jerarquía, del cual dependen todos los demás, se encuentran los diferentes subsistemas y buses de control que aparecen reflejados en la Figura 35.

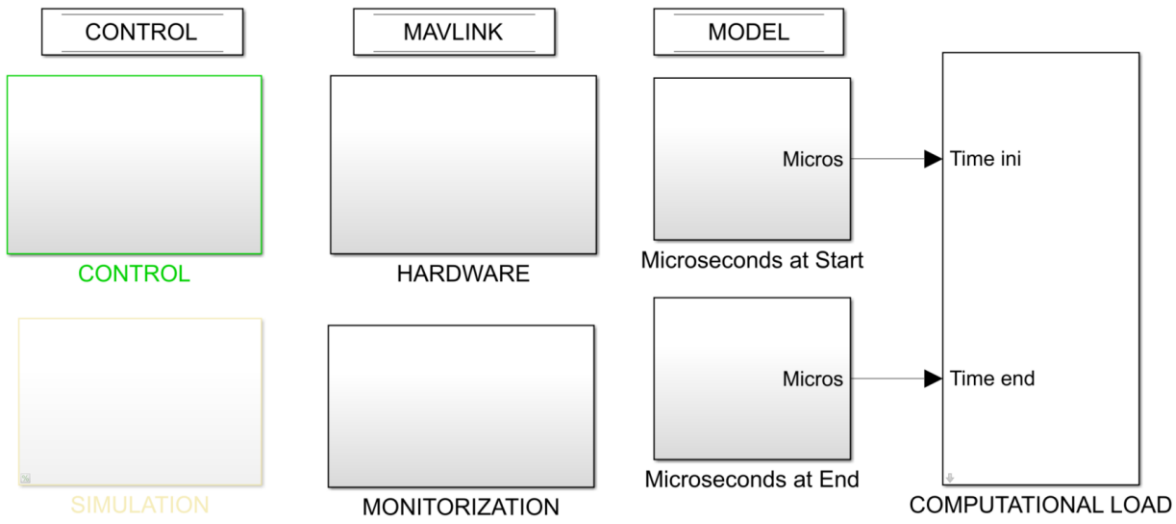


Figura 35: Fichero de Simulink para Raspberry Pi.

Los buses de *CONTROL*, *MAVLINK* y *MODEL* son estructuras de Matlab que contienen múltiples variables definidas previamente en ciertos *scripts* pertenecientes al proyecto; y son transferidos al fichero de Simulink para que los diferentes bloques que se han desarrollado en él puedan utilizar y modificar la información que contienen. Como puede apreciarse en la Figura 35, el fichero se encuentra organizado en cuatro grandes bloques; ya que los diagramas asociados al *COMPUTATIONAL LOAD* cumplen únicamente con la función de comprobar si la carga computacional exigida en la Raspberry Pi sobrepasa sus capacidades de procesamiento. Los cuatro grandes bloques del fichero son los siguientes:

- CONTROL
- SIMULATION
- MONITORIZATION
- HARDWARE

4.2.2 CONTROL

Dentro del bloque de *CONTROL* que puede apreciarse en la Figura 35, se encuentran los siguientes subsistemas:

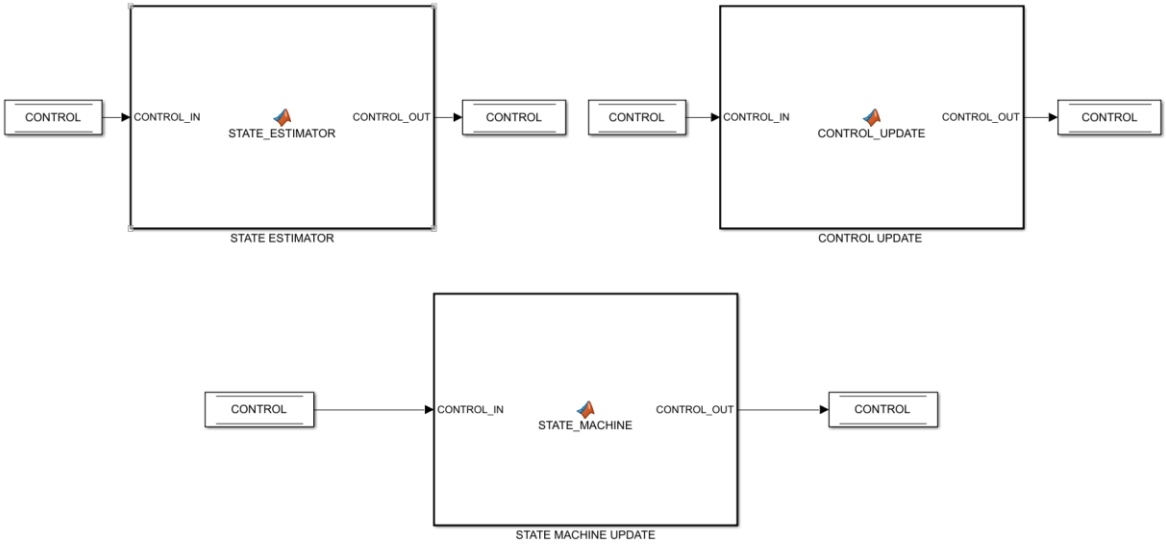


Figura 36: Subsistemas contenidos dentro del bloque CONTROL.

El primero de los subsistemas, denominado ***STATE ESTIMATOR***, actualiza el bus de *CONTROL* estimando en tiempo real las diferentes variables de estado del vehículo; que son la velocidad de avance, la velocidad de giro, el ángulo de giro, la orientación, la posición, las velocidades nombradas anteriormente respecto de un sistema fijo… Dependiendo de la configuración elegida por el usuario, dichas variables de estado a controlar pueden ser estimadas a través de las medidas directas facilitadas por los sensores del vehículo o a través de la fusión de dichas medidas. La fusión de las medidas puede llevarse a cabo a través de un sistema SLAM, un filtro de partículas, un EKF (Extended Kalman Filter), o cualquier algoritmo diseñado para tales propósitos. De momento, en este proyecto se encuentran disponibles las opciones de estimación de las variables de estado a través de las medidas directas de los sensores, o a través de la fusión de las mismas mediante un EKF; que como ya se explicó en el apartado 2.2.2, permite una estimación óptima de las variables de estado a través de la fusión de todos los sensores relacionados matemáticamente con las mismas, haciendo uso de métodos estocásticos y reduciendo las perturbaciones provenientes del ruido en las medidas.

En el segundo de los subsistemas, denominado ***CONTROL UPDATE***, también se modifica el bus de *CONTROL*, usando la estimación previa que se hizo en el subsistema anterior de

las variables de estado para actualizar las variables de los lazos básicos del control del vehículo; como la velocidad de avance, la velocidad de giro, el ángulo de giro, y demás variables de estado. En este subsistema, también se actualizan los lazos de control superiores que definen la navegación de un *waypoint* (punto de ruta fijo) a otro, o incluso la planificación y el seguimiento de diversas trayectorias.

El último subsistema, denominado ***STATE MACHINE UPDATE***, actualiza la parte del bus de *CONTROL* referente al comportamiento del vehículo en relación con la máquina de estados representada en la Figura 37.

4.2.2.1 State Machine Update

La máquina de estados contenida en el subsistema *STATE MACHINE UPDATE* comienza con un primer estado denominado ***BOOTING***, en el que se realiza la puesta en marcha del vehículo. Durante dicha puesta en marcha, el vehículo, a través de la Raspberry Pi, espera la recepción de una serie de mensajes enviados desde el fichero de Simulink ejecutado por un PC externo, que son necesarios para el correcto funcionamiento del vehículo. Conforme la Raspberry Pi va recibiendo los mensajes, esta los decodifica y extrae los valores de diferentes

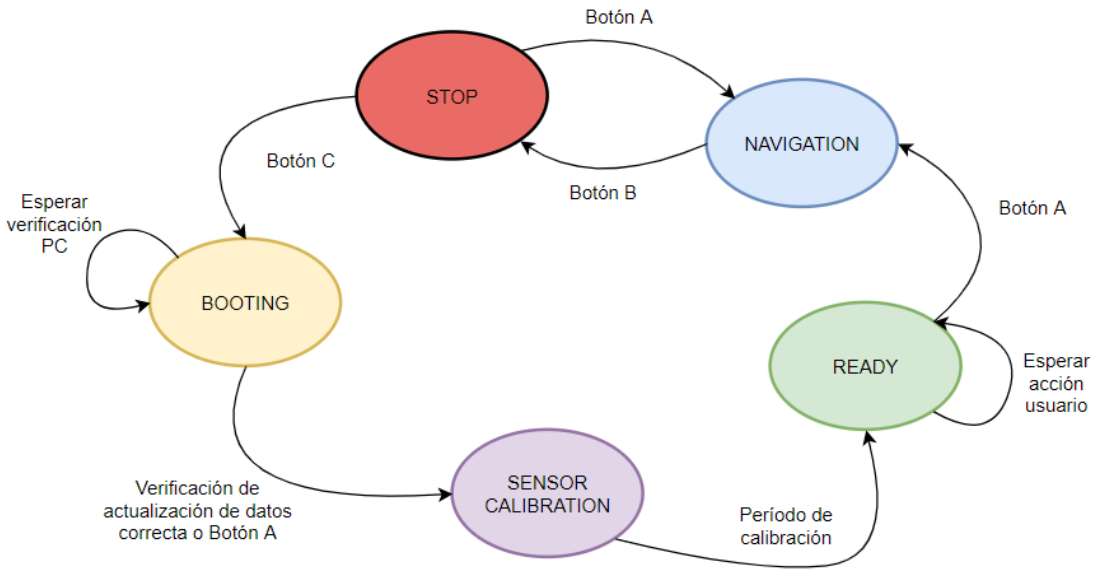


Figura 37: Máquina de estados del bloque de CONTROL.

parámetros contenidos en ellos. Tras esto, la Raspberry Pi actualiza los parámetros recibidos y vuelve a generar mensajes de vuelta al PC para que este pueda verificar que la actualización de dichos parámetros se ha llevado a cabo de forma satisfactoria. Tras dicha verificación, la Raspberry Pi pasa de forma automática al siguiente estado.

Si la verificación por parte del PC no es recibida por la Raspberry Pi, ya sea por un fallo de conexión o porque el usuario no pretende hacer uso del PC; pasado un determinado tiempo, la Raspberry Pi ofrece la opción al usuario de cargar la configuración de parámetros inmediatamente anterior, que se considerará como configuración por defecto. Si el usuario accede al uso de dicha configuración, deberá pulsar el botón A instalado en el vehículo, para hacer efectiva dicha elección y continuar hacia el estado siguiente.

El segundo estado, denominado ***SENSOR CALIBRATION***, procede a realizar una calibración de los sensores instalados en el vehículo que necesitan ciertos ajustes iniciales. En este caso, se realiza únicamente la calibración de los sensores pertenecientes a la IMU. Para ello, es necesario que el vehículo se encuentre apoyado sobre una superficie plana y horizontal; ya que el proceso de calibrado de la IMU es el siguiente:

Primero, se procede a quitar el offset existente en los giróscopos, si lo hubiese. Y segundo, se procede a forzar a que las componentes en los ejes X e Y del acelerómetro tengan valor nulo (igual a cero). Como consecuencia de esto último, el eje Z del acelerómetro será el único de sus ejes que refleje una medida distinta de cero, que además deberá coincidir con el valor de la gravedad del lugar donde se haga la calibración. Cumplido cierto tiempo establecido para que se complete sin problemas el proceso de calibración, la Raspberry Pi pasa de forma automática al siguiente estado disponible.

En el estado ***READY***, el vehículo se encuentra listo para comenzar a funcionar y realizar las diferentes funciones que el usuario haya configurado al inicio. El vehículo permanecerá en este estado hasta que el usuario intervenga a través de la pulsación del botón A; que indicará a la Raspberry Pi que el vehículo puede proceder a realizar la navegación correspondiente.

En el estado de ***NAVIGATION***, el vehículo procederá a navegar por el entorno en cualquiera de los tres modos de control elegido, previamente al estado de *BOOTING*, por el usuario. Estos estados de control son los siguientes: *Open Loop*, en el que se aplican determinadas tensiones a los motores mediante un control de lazo abierto; *Free Navigation*, en el que se ajustan la velocidad de avance y la velocidad de giro mediante sendos controles en lazo cerrado (típicamente un PID); y *Wall Follower*, en el que se ajusta la velocidad de avance mediante un control en lazo cerrado y se realiza un seguimiento de pared mediante un PID de un solo lazo, un PID en cascada, o mediante un regulador por realimentación de estado, con el fin de mantener una cierta distancia a la pared mediante el ajuste del ángulo de giro del vehículo.

Como se puede apreciar en el esquema de la Figura 37, para salir del estado de *NAVIGATION*, es necesario que el usuario intervenga mediante la pulsación del botón B instalado en el vehículo; tras el cual, este entraría de forma automática en el estado ***STOP***. En dicho estado, como su propio nombre indica, el vehículo se encuentra parado a la espera de recibir la siguiente consigna por parte del usuario. Si dicho usuario desea finalizar con la navegación, tan solo deberá esperar un determinado tiempo a que el vehículo se apague de forma automática. Si por el contrario, el usuario desea cargar una nueva configuración de parámetros desde el PC para ser probados en un nuevo ensayo, deberá primero ajustar dicha configuración en el fichero de Simulink específico para el PC; y posteriormente, pulsar el botón C, que devolverá al vehículo al estado inicial de *BOOTING*.

4.2.3 SIMULATION

Dentro del bloque de *SIMULATION* se encuentran una serie de diagramas, definidos a partir de parámetros obtenidos en ensayos, que permiten realizar simulaciones fidedignas del comportamiento del vehículo sin la necesidad de la utilización del mismo. En este bloque, se utiliza el bus de *CONTROL* de forma similar a como se utiliza en el bloque de control; con la diferencia de que los diagramas definidos dentro de este bloque hacen uso también del bus *MODEL*, el cual no se utiliza en el anterior bloque. Este último bus, posee todos los parámetros obtenidos a través de ensayos específicamente diseñados para la obtención de un

modelo matemático capaz de emular todas las funciones del vehículo de la forma más próxima a la realidad posible.

Este bloque de *SIMULATION* resulta especialmente importante a la hora de diseñar nuevos controles aplicables al vehículo; ya que permite una rápida monitorización de las respuestas que se generarían en el vehículo físico.

4.2.4 MONITORIZATION

Dentro del bloque de *MONITORIZATION* se encuentran otros dos bloques referentes a la visualización y monitorización de variables. Uno de los bloques está destinado a la monitorización de variables cuando se hace uso de la simulación; y el otro, permite la visualización de variables cuando el vehículo real se encuentra en funcionamiento y se produce una comunicación con el ordenador de a bordo, que las procesa y envía al PC para su posible visualización.

4.2.5 HARDWARE

En el bloque de *HARDWARE*, se encuentran tres bloques principales: *SENSORS, ACTUATORS* y *COMMUNICATIONS*. En el bloque de *COMMUNICATIONS*, se encuentran definidos varios protocolos de comunicación, los cuales se utilizarán en conjunción con el protocolo de mensajería Mavlink, para la transmisión de mensajes entre el PC y la Raspberry Pi. Dichos protocolos de comunicación son: UART, TCP/IP y UDP; de los cuales, ha sido el protocolo TCP/IP el escogido para la realización de las tareas de comunicación entre el PC y la Raspberry Pi.

Dentro de los otros dos bloques, pertenecientes a los sensores y actuadores, es donde se ha desarrollado gran parte del trabajo realizado en este proyecto; ya que es donde se encuentran los diferentes drivers que se han diseñado para la instrumentación del vehículo.

4.2.5.1 Sensors

Dentro de este bloque, se encuentran el driver referente a la IMU *MPU-6000* y un driver muy sencillo para la integración de los botones.

4.2.5.1.1 Driver IMU

Como ya se comentó en el apartado 3.5.1, el modelo de IMU utilizada en este proyecto tiene la posibilidad de conexión y transferencia de datos a través de SPI o I2C. En el driver desarrollado en Matlab-Simulink se ha optado por el uso de SPI; ya que para la realización de la lectura de datos, el tiempo de CPU es notablemente inferior que el necesario en I2C. La frecuencia de transmisión de datos en SPI es de 8MHz, mientras que en I2C es de 400kHz, ofreciendo unas prestaciones claramente superiores.

Como en la mayoría de los elementos hardware que poseen una cierta complejidad estructural en cuanto a lectura de datos, modos de funcionamiento, y modos de actuación, es necesaria la existencia de diferentes registros (de lectura y/o de escritura) que permitan controlar dichas funciones. La estructura de un driver es similar en la mayoría de los componentes hardware, ya se trate de sensores o de actuadores:

1. Se definen las escalas, filtros adicionales, modos de funcionamiento y lectura de datos mediante la escritura de diferentes valores, facilitados por el fabricante, en los registros correspondientes.

2. Se procede a la lectura de los datos deseados y se realiza un tratamiento sobre las mismas para poder ser incorporadas dentro de un determinado control. Este segundo proceso es repetido secuencialmente en cada periodo de muestreo.

En el caso concreto de la IMU MPU-6000 usada en el vehículo de este proyecto, la primera parte del driver, descrita anteriormente, es llevada a cabo en un fichero de Matlab; encargado de facilitar el acceso a todos los registros y las configuraciones disponibles a la segunda parte del driver desarrollada en Simulink.

La parte del driver desarrollada en Simulink, que es realmente la parte funcional del driver, puede observarse en la Figura 38. Siguiendo el flujo de datos observado en el diagrama, de izquierda a derecha, se aprecian los siguientes procesos:

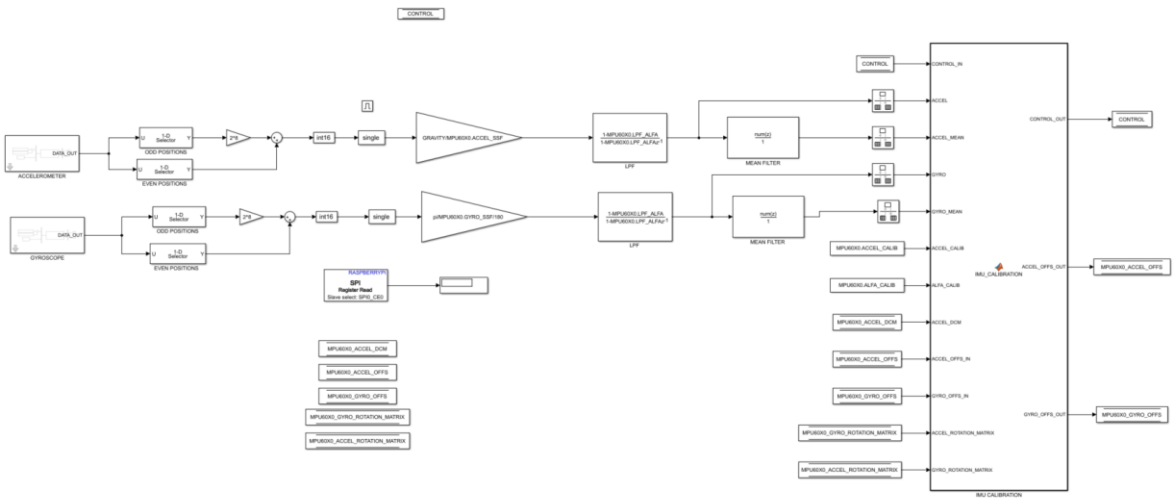


Figura 38: Driver MPU-6000 en Simulink.

1. En la parte izquierda del diagrama se encuentran los bloques de acceso a los registros de datos de los acelerómetros (bloque superior) y de los giroscopios (bloque inferior). Cada uno de ellos proporciona una cadena de datos de 6 bytes, en los cuales se encuentran agrupados los datos de cada uno de los 3 ejes (2 bytes por eje); ya que los datos pertenecientes a cada uno de ellos son de tipo *int16*.

2. Para poder obtener el valor numérico correspondiente a cada uno de los ejes, se separan los datos cada dos bytes; y tras esto, se procede a multiplicar el valor del byte superior por 2^8 y agregar a esa medida el valor del byte inferior. Este proceso se realiza de forma paralela para los datos provenientes del acelerómetro y del giróscopo.

3. Tras la obtención de las tres medidas en numeración decimal correspondientes a cada uno de los ejes, se procede a aplicar una escala (facilitada por el fabricante) para obtener los datos en las medidas deseadas. En el caso de este proyecto, se opta por

las medidas definidas por el SI, que corresponden a m/s^2 para los acelerómetros y a rad/s para los giróscopos.

4. Tras la conversión de unidades anterior, se procede a aplicar un filtro externo a dichas medidas; el tipo de filtro usado en este caso es un filtro paso bajo de primer orden diseñado en tiempo discreto que actúa cada 1ms. Normalmente, y como ocurre con la MPU-6000, el fabricante ofrece una serie de filtros internos, que se han decidido obviar para poder tener mayor control sobre el filtrado de las señales recibidas a través del filtro externo aplicado. Tras el paso por este filtro, se obtienen unas primeras medidas de la aceleración (acelerómetro) y de la velocidad angular (giróscopo) presentes en cada eje.

5. A la salida de las medidas filtradas en el proceso anterior se aplica un filtro de media móvil, en el cual se integran cada 10ms las medidas procedentes del acelerómetro y del giróscopo que se utilizarán, tras su calibrado, para realizar la estimación de estados en un filtro EKF.

6. Por último, haciendo uso de todas las medidas obtenidas, se procede a realizar la fase de calibración de la IMU MPU-6000 correspondiente al estado de *Calibration* presente en la máquina de estados descrita en el apartado 4.2.2.1. Durante este proceso, se eliminan los posibles offset presentes en los acelerómetros y en los giróscopos.

4.2.5.1.2 Driver pulsadores

El driver para la integración de los botones es un diagrama muy sencillo en el cual se hace uso del paquete de soporte de Simulink para Raspberry Pi con el fin de actualizar la variable del bus de *CONTROL* referente al estado de cada uno de los botones. Este diagrama puede apreciarse en la Figura 39.

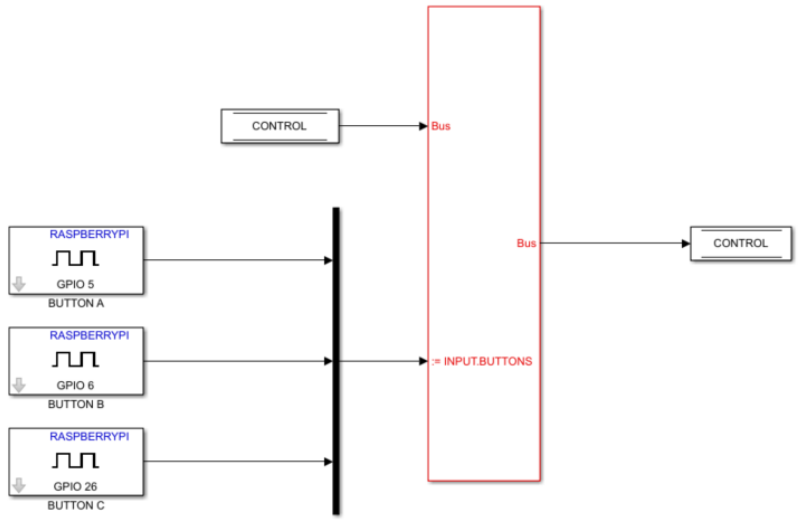


Figura 39: Driver lectura de botones.

4.2.5.2 Actuators

En el bloque de Simulink correspondiente a los diferentes actuadores del proyecto, se encuentra únicamente el driver de ambos MD25.

4.2.5.2.1 Driver MD25

En este bloque, se encuentra el driver de los dos *MD25* incorporados para la lectura de los encoders y para el manejo de la actuación sobre los cuatro motores *EMG30* instrumentados en el vehículo. La conexión con los dos *MD25* se ha realizado a través de I2C, como se explicó anteriormente en el apartado 3.7.1.

Para llevar a cabo la configuración de los modos de funcionamiento y establecer las direcciones de los diferentes registros de escritura y/o lectura, se ha desarrollado un fichero en Matlab común para ambos MD25. En dicho fichero, tras identificar las direcciones I2C de cada MD25 (han de ser diferentes, se pueden cambiar enviando una serie de comandos), se actualizan todos los registros presentados por el fabricante en el datasheet. En este mismo fichero de Matlab, se actualizan también los valores de ciertos parámetros de configuración que se enviarán a los registros *Speed1* y *Speed2* para establecer una actuación independiente en cada uno de los motores. Se establece también el valor del registro *Acceleration rate* para

establecer un valor de aceleración en el que el cambio de velocidad máxima entre un sentido y otro sea realizado en un tiempo igual a 1,275s (valor por defecto del fabricante). Por último, se establece un valor igual a 1 en el registro llamado *Mode*, estableciendo valores de comando para los registros de velocidad comentados anteriormente entre -128 (máxima velocidad hacia atrás) y 127 (máxima velocidad hacia adelante); siendo 0 el equivalente a un valor de *stop*.

La otra parte del driver de los MD25 corresponde con la realizada en Simulink, cuyo nivel jerárquico más alto puede observarse en la Figura 40 expuesta a continuación.

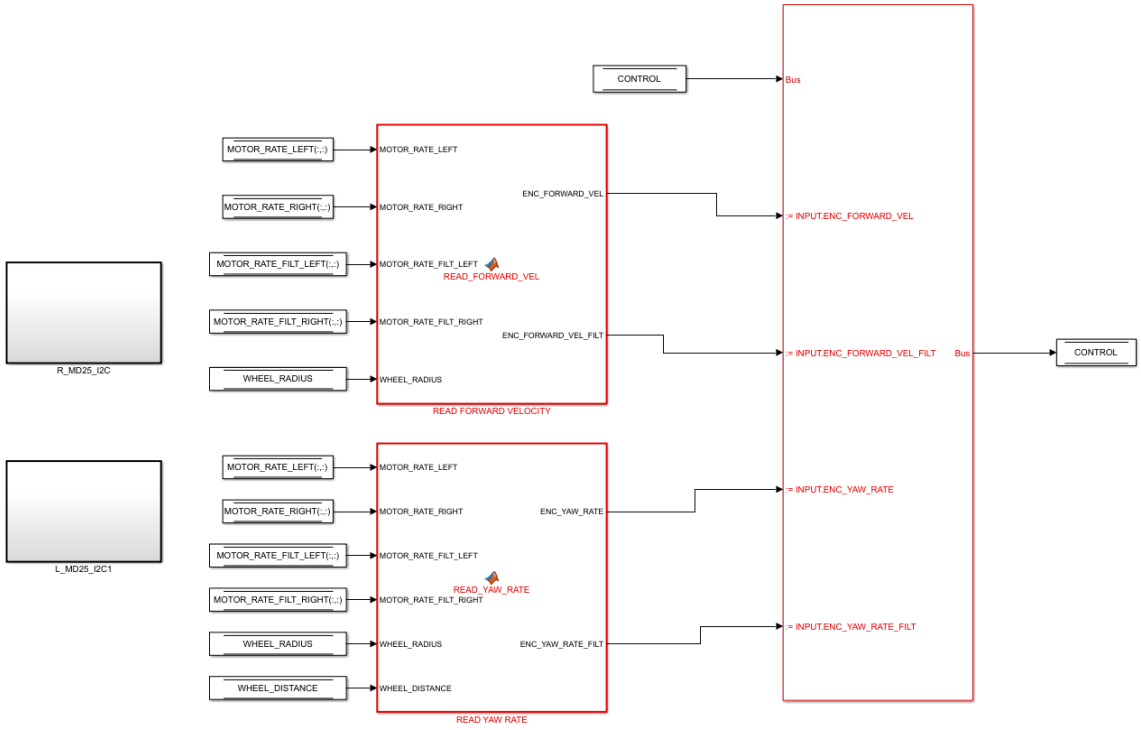


Figura 40: Primer nivel jerárquico del driver de los MD25.

En este primer nivel jerárquico se procede a la transformación de los datos provenientes de los encoders de cada uno de los motores a los valores de velocidad de avance y velocidad angular de giro (*yaw rate*). Para poder realizar dicha conversión, se utilizan dos bloques separados que incluyen diferentes parámetros del vehículo, como el radio de la rueda o la distancia entre ejes, que resultan necesarios para la obtención de los estados mencionados. Como se puede observar en la Figura 40, para poder llevar a cabo el cálculo anterior, es

necesario conocer los valores referentes a la velocidad angular de los motores de la izquierda y de la derecha, filtrados y sin filtrar. Estos valores, son calculados de forma paralela para los motores presentes en el hemisferio izquierdo del vehículo y para los presentes en el lado derecho a través de los dos bloques que aparecen a la izquierda del diagrama de la Figura 40.

Tomando como ejemplo el bloque referente a los motores de la derecha, dentro del mismo se encuentra la estructura presentada en la Figura 41, en la que se dividen las funciones realizadas en el MD25 de los motores derechos en cinco bloques principales.

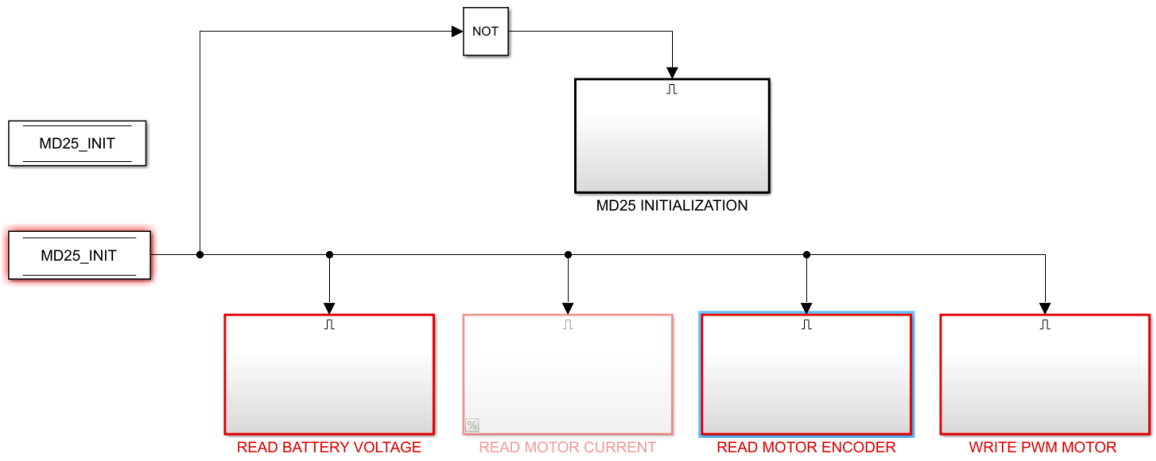


Figura 41: Bloques motores del lado derecho del driver de los MD25.

Un primer bloque denominado *MD25 INITIALIZATION* en el que se establece el modo de funcionamiento contenido en el registro *Mode*, tal y como se explicó anteriormente en este apartado.

Un segundo bloque denominado *READ BATTERY VOLTAGE*, en el que, como su propio nombre indica, se procede a leer, desde el registro correspondiente, la tensión que llega al MD25 desde la batería LiPo. Dicho valor de tensión es filtrado a través de un filtro lineal de primer orden en tiempo discreto, tras el cual se obtiene el valor final de tensión utilizado en el control del vehículo.

Un tercer bloque denominado *READ MOTOR CURRENT*, en el que se procede a la lectura y el filtrado de las corrientes presentes en los motores mediante una metodología muy similar a la descrita en el bloque anterior.

Un cuarto bloque denominado *READ MOTOR ENCODER*, en el que se procede a la lectura de los datos proporcionados por los encoders, los cuales se filtran mediante un filtro no lineal. A partir de los datos filtrados y no filtrados, se obtienen mediante operaciones aritméticas las velocidades angulares presentes en los motores, tanto filtradas como sin filtrar.

Un último bloque denominado *WRITE PWM MOTOR*, en el que se calcula la señal PWM que es necesaria enviar a los motores para obtener la velocidad requerida por el control del vehículo.

4.2.6 SOFTWARE UTILIZADO EN LA RASPBERRY PI

En esta primera parte del proyecto, en la que se trabaja exclusivamente con Matlab y Simulink, el sistema operativo que se ha decidido utilizar en la Raspberry Pi ha sido *Raspberry Pi OS*, anteriormente llamado *Raspbian*. La decisión de utilizar este sistema operativo es bastante lógica, ya que es el entorno desarrollado específicamente para las tarjetas Raspberry Pi. Con el fin de poder establecer comunicación con Matlab y Simulink, es necesaria la habilitación de ciertas interfaces a través del menú de *Configuración de Raspberry Pi*:

- SSH: para la comunicación con el PC.
- SPI: para habilitar la comunicación con la IMU.
- I2C: para habilitar la comunicación con los MD25 y los motores.
- Remote GPIO: para permitir a Simulink operar y obtener información de los puertos lógicos de la Raspberry Pi.

Tras la habilitación de las mencionadas interfaces, y la conexión de la Raspberry Pi a la misma señal Wifi que el PC, la conexión con Simulink debería poder hacerse efectiva; tan

solo es necesaria la indicación en Simulink de la dirección IP asociada a la Raspberry Pi que el usuario desea conectar.

4.3 ROS

Tras la implementación satisfactoria de los drivers referentes a la IMU y a los MD25 en el entorno de Matlab-Simulink, se prosiguió con la implementación del LIDAR. Este sensor, es el componente hardware más complicado de incorporar en un driver; y puesto que su desarrollo en Simulink requiere de un alto grado de conocimiento y de tiempo, se decide explorar la opción de ROS directamente para este sensor.

4.3.1 INSTALACIÓN

ROS es un entorno que ha sido creado desde un principio para su uso en las distribuciones de Ubuntu y Debian, basadas en Linux. Actualmente, existe una versión con pleno soporte para Windows, que aunque no se ha probado durante este proyecto, sí que se ha encontrado información que apunta a que no se encuentra del todo bien implementado. Para el desarrollo de este proyecto, esto último no va a ser un problema; ya que es posible implementar ROS en Matlab y Simulink a través de la instalación de *ROS Toolbox*. Incluso, dada la importancia de ROS y su estandarización, Matlab y Simulink, han integrado el uso de Gazebo y diversas utilidades robóticas en el paquete llamado *Robotics System Toolbox*. Todos los *add-ons* necesarios para el desarrollo de este proyecto se encuentran listados en el apartado 1.4.

La instalación de ROS en el PC es bastante sencilla, donde realmente es algo más complicada y se han encontrado bastantes impedimentos ha sido en su instalación en la Raspberry Pi. El principal problema, es que como se ha comentado al principio de este apartado, ROS no se encuentra optimizado para el sistema operativo por defecto de la Raspberry Pi, el Raspberry Pi OS (en el que se consiguió finalmente su instalación); por lo que se buscaron diversas alternativas:

- Instalación de la imagen Ubiquity Robotics: Esta imagen desarrollada por Ubiquity Robotics está basada en Ubuntu 16.04 e integra, sin necesidad de instalación, ROS

en su distribución *Kinetic*. Esta es la opción recomendada desde Matlab y Simulink, y según la información facilitada por Ubiquity Robotics, está diseñada para funcionar en las Raspberry Pi 3 Model B/B+ y en la Raspberry Pi 4 Model B. Sin embargo, esta solución está pensada para trabajar con los robots que la empresa desarrolladora de la imagen software tiene a la venta; y cada vez que se enciende la Raspberry Pi, ya existen por defecto un ROS Master y varios *topics* y nodos activos en la red. Se ha intentado desactivar sin éxito esta función predeterminada siguiendo los pasos explicados en la página web del fabricante. Además de este contratiempo, la imagen presenta ciertos problemas de compatibilidad con redes Wifi en España y ciertos fallos de software; por lo que no es una solución recomendable.

- Instalación de Ubuntu 18.04: Se ha intentado la instalación de Ubuntu 18.04 en la Raspberry Pi, versión recomendada para la distribución *Melodic Morenia* de ROS. Se ha probado la instalación de dos formas diferentes: mediante la instalación de Ubuntu 18.04 Desktop (no optimizada para hardware con arquitectura *arm64*, como es el caso de las Raspberry Pi), y con la instalación de Ubuntu Server. La primera opción no se ha conseguido hacer funcionar de forma correcta; de hecho no ha sido posible ni cargar el escritorio principal. La segunda opción, software especialmente desarrollado por Ubuntu para Raspberry Pi para aplicaciones de servidor o de IoT, ha podido instalarse de forma correcta. Sin embargo, esta opción carece de toda funcionalidad gráfica y tan solo posee una terminal. No se ha procedido a instalar ROS en esta terminal debido a la imposibilidad de ejecutar ciertas herramientas de ROS que precisan de una interfaz gráfica para su funcionamiento. Sin embargo, sí se ha tratado de instalar uno de los escritorios recomendados en la página web propia de Ubuntu, aunque sin éxito para ninguna de las opciones propuestas; por lo que esta opción tampoco es recomendable.

- La instalación de **Ubuntu Mate 18.04.2** en su versión de 64-bit ha resultado estar realmente bien optimizada para la Raspberry Pi. Tras solucionar ciertos problemas de conexión SSH entre la Raspberry Pi y el PC, ha sido posible la instalación de ROS en su distribución *Melodic Morenia* directamente desde código siguiendo las

instrucciones documentadas en la página de documentación oficial de ROS: **wiki.ros.org**. Esta imagen software ha sido la utilizada para llevar a cabo todos los apartados referentes a ROS en la Raspberry Pi, exceptuando el ensayo con el paquete de soporte para Raspberry Pi de Simulink. La instalación y los pasos que se han llevado a cabo para arreglar los diferentes problemas de conexión se encuentran recogidos al final de este documento en el ANEXO I.

- Instalación de **ROS (source) en Raspberry Pi OS**: Después de varios intentos y de una instalación muy lenta de varias horas; ha sido posible la instalación de ROS en su versión *Melodic Morenia* en una Raspberry Pi 3 Model B+ bajo Raspberry Pi OS. Esta solución, aunque dista de ser perfecta debido a errores recurrentes en el funcionamiento del software relacionados a funciones realizadas con el ratón; es sin duda la solución más interesante de todas las abordadas. Si se consigue un ratón que no de problemas, o se aprenden ciertos comandos de teclado (como se ha hecho para este proyecto), esta solución incluye todas las funcionalidades de ROS junto con la sencillez a la hora de configurar interfaces de comunicación en la Raspberry Pi como SSH, I2C, VNC… Pero sin duda, la mayor ventaja que presenta esta solución es la de poder hacer uso del paquete de soporte de Simulink para construir nodos funcionales en ROS; y poder hacer uso de los diferentes drivers y controles que hacen uso de dicho soporte, mientras se ejecutan otros procesos paralelamente desde ROS. La instalación de ROS *Melodic Morenia* no es sencilla, y puede prolongarse durante varias horas, pero acaba mereciendo la pena. La instalación de ROS *Melodic Morenia* en Raspberry Pi OS se encuentra convenientemente explicada en el ANEXO II.

4.3.2 DRIVER DEL LIDAR

Sin lugar a duda, el driver del LIDAR ha sido el más completo y sencillo de instalar en la Raspberry Pi haciendo uso de ROS. El paquete escogido para integrar el LIDAR en el vehículo ha sido el denominado *rplidar_ros*, ofrecido en GitHub por *robopeak*. *RoboPeak* es un equipo de investigación y desarrollo centrado en plataformas y aplicaciones robóticas que fue fundado en 2009. El mismo grupo, fundó la compañía *Slamtec* en 2013, la encargada

de desarrollar los dos LIDAR con los que se ha trabajado en este proyecto. Debido a esto, el paquete instalado ha resultado estar bastante completo y bien desarrollado. En el mismo paquete, *RoboPeak* incluye drivers y funciones para todos sus modelos actualmente disponibles (A1, A2 y A3); característica gracias a la cual ha sido posible la realización de ensayos con los modelos A1 y A2M8 de RPLIDAR. Dichas pruebas y los resultados obtenidos en ellas pueden observarse en el apartado 5.2.1.

La instalación del paquete ha resultado bastante sencilla debido al gran desarrollo comentado anteriormente, y a las claras indicaciones presentes en el archivo *README.md* del mismo. Simplemente es necesario copiar el repositorio disponible en GitHub en la carpeta *src* del *catkin workspace* deseado:

```
cd ~/catkin_ws/src
git clone https://github.com/robopeak/rplidar_ros.git
cd ..
catkin_make
```

Un punto a tener en cuenta es que cada vez que se desconecte el LIDAR o se apague la Raspberry Pi, es necesario dar al puerto USB la autorización de escribir. Para ello, simplemente hay que introducir el siguiente código en la terminal del SBC utilizado:

```
ls -l /dev | grep ttyUSB
sudo chmod 666 /dev/ttyUSB0
```

Una vez realizado este paso, es importante observar las diferentes opciones que ofrece este driver. El paquete permite la visualización de los datos procedentes del LIDAR a través de dos opciones diferentes: a través de *RViz*, o a través de un servicio cliente. El resultado de ambas opciones puede observarse en el apartado 5.2.1. En cualquier caso, la estructura del driver del LIDAR al conectarlo a Matlab es la misma.

4.3.2.1 Estructura del driver del LIDAR

Como puede apreciarse en la Figura 42 obtenida a través de la herramienta *rqt_graph*, el paquete instalado permite la creación de un nodo en ROS denominado */rplidarNode*, que una vez conectado a Matlab es capaz de transmitir un mensaje al topic */scan*. El mensaje transmitido al nodo receptor en Matlab corresponde a un tipo de mensaje predefinido en

ROS denominado *sensor_msgs/LaserScan*, cuya estructura de datos puede comprobarse en la Figura 43.

Figura 42: rqt_graph del driver del LIDAR en ROS.

```
pi@raspberrypi:~ $ rosmsg show sensor_msgs/LaserScan
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Figura 43: Estructura del mensaje tipo sensor_msgs/LaserScan.

4.3.3 DRIVER MD25

Para la integración de los drivers de los MD25 en ROS se han probado los diferentes paquetes disponibles en la red, pero ninguno ha cumplido con las expectativas requeridas en el proyecto. En concreto, uno de ellos se encontraba inacabado y sin ofrecer realmente ninguna funcionalidad; y el restante, más desarrollado, había sido diseñado para ser utilizado mediante un concentrador hardware Arduino, que permitiese la transmisión de los datos obtenidos de los MD25 hacia la Raspberry Pi a través de una comunicación en serie. Dado que ninguna de las dos soluciones disponibles sirve a los propósitos del presente proyecto, se han barajado diferentes soluciones:

- Aprender el entorno ROS en profundidad y desarrollar directamente los drivers en ROS para la Raspberry Pi. Esta opción es quizás la ideal si se busca un control total

sobre el driver del dispositivo; ya que todas las funcionalidades y características del driver han sido codificadas a mano. Sin embargo, este proceso requiere de una elevada cantidad de tiempo, de la cual no se dispone en este proyecto; por lo que ha sido desechada.

- Aprender el entorno de Arduino y utilizar alguno de sus modelos más sencillos para crear un concentrador hardware. Esta solución es la más parecida al último de los paquetes disponibles comentados anteriormente, y que tal y como se comprobaría más adelante, tampoco se hallaba bien desarrollado. Sin embargo, esta solución resulta interesante debido a la gran comunidad de usuarios de Arduino disponible en internet, la cual ofrece código abierto de una infinidad de proyectos; acercándose algunos a las necesidades del proyecto. Si se adoptase finalmente esta solución, tras la adquisición de datos por parte del Arduino, estos deberían ser transmitidos a la Raspberry Pi a través de comunicación en serie para que esta pueda compartirlos en la red ROS a través de un nodo *Publisher*. Una vez obtenidos los datos, resulta relativamente sencillo diseñar en Simulink un bloque encargado del tratamiento de la información recibida a través de dicho nodo, e incorporar el bloque diseñado como otro nodo ROS independiente a través de Simulink Coder.

4.3.3.1 Conversión de bloque Simulink a ROS

Finalmente, se ha decidido explorar la segunda opción propuesta en el apartado anterior. Debido a la escasez de tiempo disponible para la realización de este proyecto, los esfuerzos se han concentrado en la tarea más desconocida en el laboratorio de control de la universidad, la conversión de diagramas de Simulink a nodos ROS ejecutables directamente desde la Raspberry Pi.

Para poder realizar esta función, han sido necesarios la introducción de los bloques de Simulink pertenecientes al paquete *ROS Toolbox*; y para la conversión a código C++ compilable y ejecutable en ROS, han sido utilizados los paquetes *Simulink Coder*, *Matlab Coder* y *Embedded Coder*.

El ensayo requiere de dos ficheros Simulink diferentes para llevarse a cabo: uno que corre en el PC, y otro que es convertido a C++ para poder ser instalado en el entorno ROS de la Raspberry Pi. La estructura del fichero de **Simulink perteneciente al PC** desarrollado para el ensayo ha resultado sencilla, y puede ser apreciada en la Figura 44.

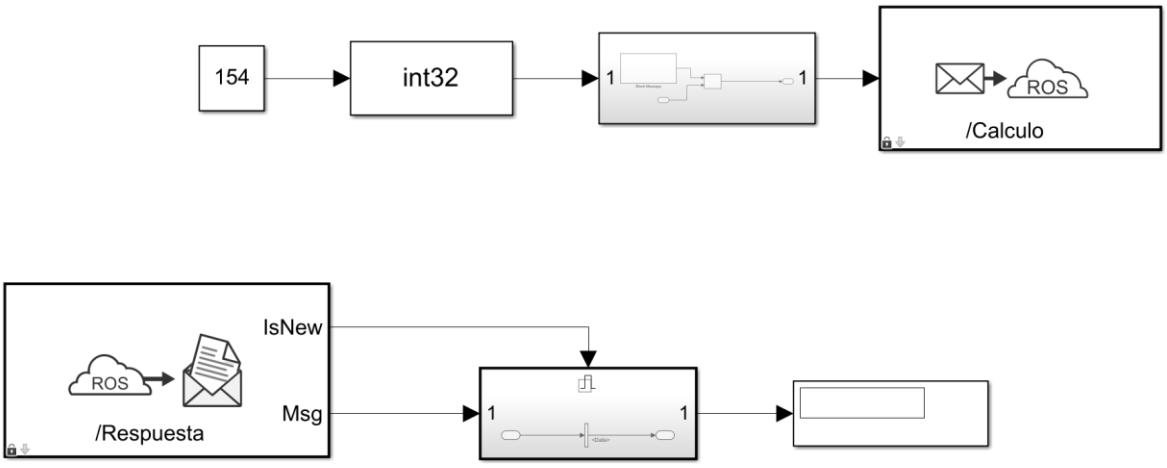


Figura 44: Estructura Simulink – ROS para PC.

La primera fila de bloques de Simulink, realiza publicación de un cierto valor en formato *int32*, que es publicado en el *topic* ***/Calculo*** a través de un nuevo nodo que es generado mediante Simulink. El subsistema que se aprecia en dicho proceso convierte el valor *int32* deseado en un mensaje *int32* específico de ROS; y puede apreciarse con mayor detalle en la Figura 45.

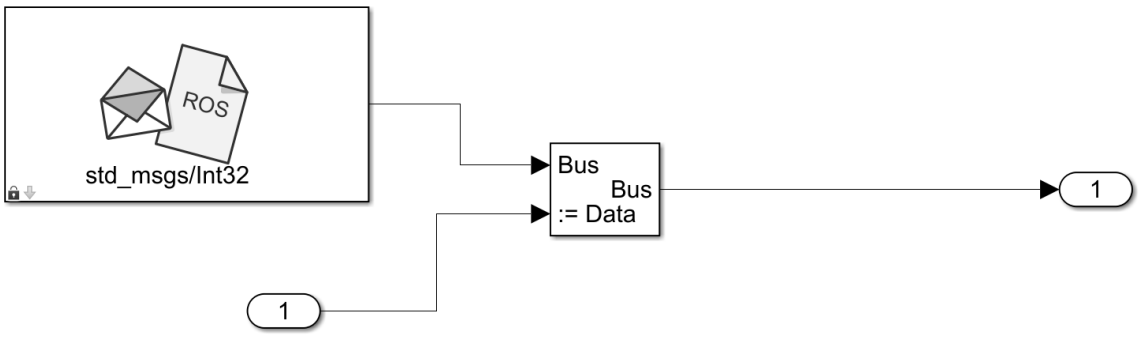


Figura 45: Creación de un mensaje ROS en Simulink.

Como se puede apreciar en la Figura 45, simplemente es necesaria la integración del bloque *Blank Message* perteneciente al *ROS Toolbox*; el cual convierte el *int32* generado en Simulink en un *std_msgs/Int32* compatible con ROS mediante un *Bus Assignment*.

La segunda fila de bloques presentes en la Figura 44 corresponde a la recepción de datos procedentes del *topic* ***/Respuesta*** gracias al bloque *Subscribe* perteneciente al *ROS Toolbox*. Para el tratamiento de los datos recibidos, se hace uso de un subsistema que, cada vez que llega un mensaje nuevo, hace uso de un *Bus Selector* para seleccionar la variable del mensaje deseada. Ver Figura 46.

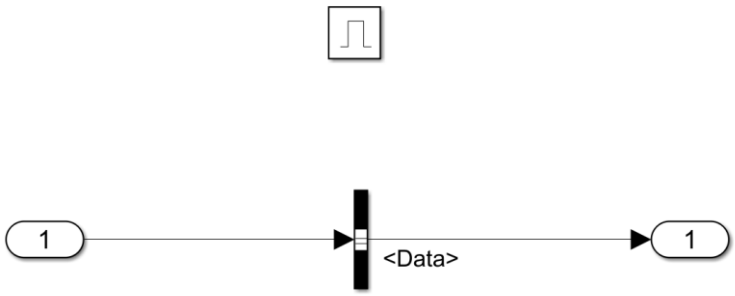


Figura 46: Subsistema de seleccion de datos procedentes de ROS.

La estructura del fichero de **Simulink que pretende ser volcada en la Raspberry Pi** como nodo ejecutable en ROS es la presentada en la Figura 47.

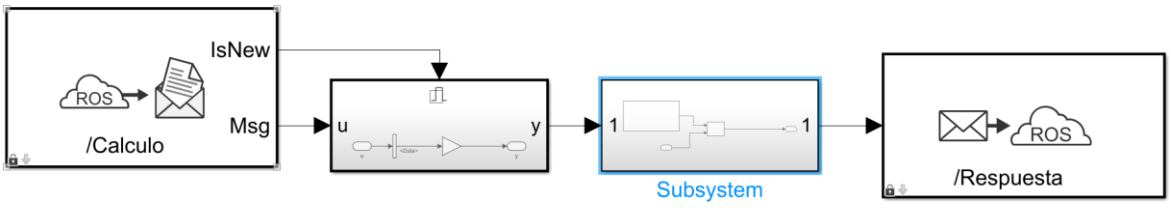


Figura 47: Estructura Simulink volcada en la Raspberry Pi.

Como puede apreciarse, la estructura hace uso de un bloque *Subscribe* para recibir los datos publicados por el PC en el *topic* ***/Calculo***. Tras esto, se integra en un subsistema tipo *Enabled Subsystem* en el cual se hace uso de un *Bus Selector* y un *Gain* para modificar el valor recibido; tal y como puede apreciarse en la Figura 48. Tras su paso por el subsistema, el

valor triplicado es introducido en otro subsistema exactamente igual al mostrado en la Figura 45; para crear un mensaje tipo *std_msgs/Int32* que pueda ser publicado en el *topic* ***/Respuesta*** que recibirá de vuelta el PC.

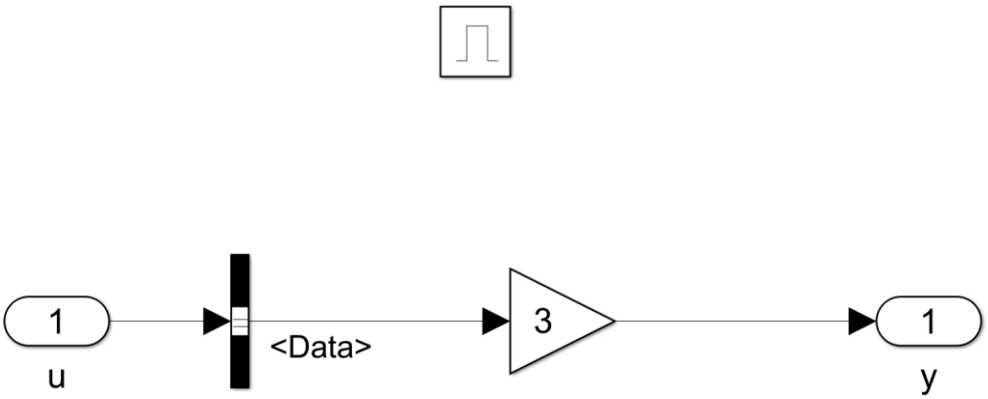


Figura 48: Enabled Subsystem volcado en la Raspberry Pi.

El proceso de conexión realizado para ambos ficheros se encuentra convenientemente descrito en el ANEXO II.

4.3.3.1.1 Problemas de procesado y conexión

Gracias a la realización del ensayo descrito en el apartado anterior, ha sido posible la identificación de una serie de problemas cuya resolución ha resultado ser de vital importancia en otros proyectos similares desarrollados en el laboratorio de la universidad; y lo será para futuros proyectos de igual forma.

El primero de los problemas surge al intentar compilar el fichero de Simulink destinado a la Raspberry Pi. Una vez establecida y comprobada la conexión a través de ROS entre la Raspberry Pi y el PC, y tras presionar la función de *Build & Load* presente en la pestaña *Robot* de Simulink; el diagrama se descarga en la Raspberry Pi y ROS procede a su instalación. Debido a la insuficiente memoria RAM presente en la Raspberry Pi 3 Model B+ (1GB), esta se ve incapaz de completar la compilación e instalación del diagrama y se congela en cierto punto del proceso. Para ello, es necesario habilitar una parte

correspondiente de la memoria en la microSD como memoria RAM auxiliar de la Raspberry Pi.

El proceso para su habilitación se encuentra recogido en el ANEXO III. Seguir todo el proceso hasta fijar un valor de *swappiness* igual a 60. [43]

Tras esto, al repetir el proceso de *Build & Load*, debería haberse instalado en el *catkin workspace* un nuevo paquete con el nombre del fichero de Simulink. Para ejecutarlo desde la Raspberry Pi, abrir un nuevo terminal y escribir (siendo *NOMBRE* el nombre del fichero de Simulink) lo siguiente:

```
rosrun NOMBRE NOMBRE_node
```

Por último, tan solo queda ejecutar el fichero perteneciente al PC encargado de enviar y recibir los valores deseados. Para ello, es preciso dirigirse a la pestaña *SIMULATION* y presionar la opción *Run*.

Los resultados obtenidos y los problemas de conexión surgidos asociados a la conexión de 5GHz en Wifi se encuentran explicados en el apartado 5.2.2.

4.3.4 DRIVER DE LA IMU

Como ya se ha comentado anteriormente, una de las grandes ventajas presentes en ROS es la de poder integrar, de forma modular, el trabajo y los conocimientos que miles de personas han compartido en internet. La plataforma predilecta para la publicación y descarga gratuita de este tipo de trabajos es el portal *GitHub,* donde la comunidad de ROS comparte y expone a revisión la mayor parte de sus proyectos. A la hora de buscar los diferentes paquetes que el usuario desee integrar en su propio proyecto, es importante leer con atención la descripción que dejan los autores de los mismos en el archivo *README.md*; para comprobar que efectivamente, dicho paquete cumple con las características de implementación y hardware que requeridas.

Para la integración del driver de la IMU, se ha escogido el paquete desarrollado por el usuario de GitHub *chrisspen* denominado *ros_mpu6050_node*. Como se puede comprobar por el nombre del paquete, se trata de un modelo de IMU diferente al que se incorpora en este

proyecto, que es la MPU-6000; sin embargo, ambas IMU son muy parecidas, e incluso comparten el mismo datasheet elaborado por su fabricante. Debido a esto, y a la escasez de paquetes específicos para la MPU-6000, se ha optado por la elección de este paquete.

Tras cumplir todo el proceso de instalación indicado en el ANEXO I, se procede a la clonación del paquete de la IMU y su posterior instalación:

1. Instalar I2Cdevlib:

```
sudo mkdir -p /usr/share/arduino/libraries
cd /usr/share/arduino/libraries
sudo git clone https://github.com/chrisspen/i2cdevlib.git
```

2. Instalar Bcm2835:

```
cd /tmp
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.50.tar.gz
tar zxvf bcm2835-1.50.tar.gz
cd bcm2835-1.50
./configure
make
make check
sudo make install
```

3. Clonar el Proyecto en el workspace y proceder a su instalación:

```
cd ~/catkin_ws/src
git clone https://github.com/chrisspen/ros_mpu6050_node.git
cd ..
catkin_make --pkg ros_mpu6050_node
```

Tras realizar este proceso, y como ocurre con la mayoría de los paquetes clonados desde internet, es necesario realizar ciertos cambios para que funcionen de acuerdo con el hardware específico del proyecto. Para ello, es necesaria la instalación de la herramienta *i2c-tools*, que permite la visualización de los puertos I2C conectados a la Raspberry Pi (puede ser cualquier dispositivo similar). Para ello:

```
sudo apt install i2c-tools
```

Esta, tras su instalación, permite mostrar la dirección utilizada por la IMU en la Raspberry Pi.

Tras comprobar la dirección del puerto I2C asignado a la IMU en la Raspberry Pi, se procede a editar el archivo *mpu6050_node.cpp* con el fin de establecer los parámetros del puerto I2C

correspondientes al hardware del vehículo; y acto seguido, paso necesario cada vez que se modifique algún archivo en C++ referente a cualquier paquete, se procede a la recompilación del *catkin workspace* mediante el uso del comando *catkin_make*.

4.3.4.1 Estructura del driver de la IMU

Como puede apreciarse en la Figura 49 obtenida a través de la herramienta *rqt_graph*, el paquete instalado permite la creación de un nodo en ROS denominado */mpu6050_node*, que una vez conectado a Matlab es capaz de transmitir un mensaje al topic */imu/data*. El mensaje transmitido al nodo receptor en Matlab corresponde a un tipo de mensaje predefinido en ROS denominado *sensor_msgs/Imu*, cuya estructura de datos puede comprobarse en la Figura 50.

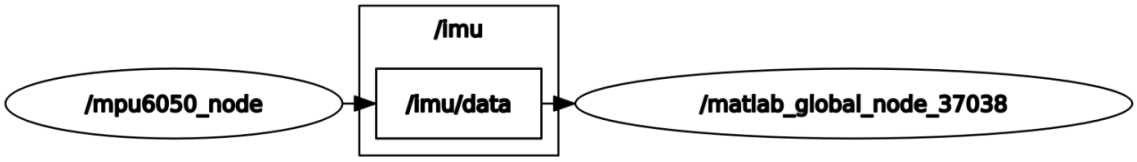


Figura 49: rqt_graph del driver de la IMU en ROS.

```
pi@raspberrypi:~ $ rosmsg show sensor_msgs/Imu
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
  float64 x
  float64 y
  float64 z
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
  float64 x
  float64 y
  float64 z
float64[9] linear_acceleration_covariance
```

Figura 50: Estructura del mensaje tipo sensor_msgs/Imu.

Finalmente, tras realizar diversos ensayos, los resultados obtenidos a través del paquete descargado distan mucho de ser correctos y, debido a ello; se decide explorar la última de las opciones posibles: el uso de ROS en Raspberry Pi OS.

4.3.4.2 Conversión de bloque Simulink con paquete de soporte para Raspberry Pi a ROS

A pesar de que ya se había probado anteriormente con éxito la solución de utilizar un Arduino como concentrador hardware para obtener los datos de diferentes sensores, se ha querido ir un paso más allá; y tras conseguir una instalación satisfactoria de ROS *Melodic Morenia* en Raspberry Pi OS, se ha procedido a comprobar si es posible hacer uso de las funcionalidades ofrecidas por el paquete de soporte para Raspberry Pi en nodos ROS. Para ello, y debido a la falta de tiempo, se ha decidido la realización un ensayo sencillo en el que se procede a volcar un nodo ROS desde Simulink que incluye la lectura de un registro I2C de la IMU a través de un bloque perteneciente al paquete de soporte para Raspberry Pi. Para llevar a cabo dicha tarea, tal y como ocurre en el ensayo descrito en el apartado 4.3.3.1, es necesaria la utilización de dos ficheros de Simulink, uno convertido a nodo ROS que es ejecutado en la Raspberry Pi; y otro encargado de recibir información en el PC.

La estructura del fichero de **Simulink perteneciente al PC** presente en este ensayo puede apreciarse en la Figura 51. El fichero se compone de una serie de bloques orientados a la recepción de datos procedentes del *topic* ***/WhoAmI***. Esto es logrado gracias al bloque *Subscribe* perteneciente al *ROS Toolbox* y a un subsistema tipo *Enabled Subsytem* que, cada vez que llega un mensaje nuevo, hace uso de un *Bus Selector* para seleccionar la variable deseada del mensaje. La estructura presente en el subsistema es la misma empleada en la Figura 46. Tras la salida del subsistema, el dato obtenido puede ser visualizado a través de un bloque *display* de Simulink.

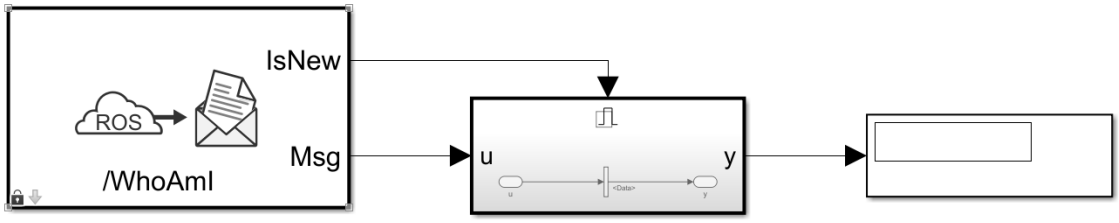


Figura 51: Estructura para PC de nodo ROS con paquete de soporte para Raspberry Pi.

La estructura del fichero de **Simulink que pretende ser volcado en la Raspberry Pi** como nodo ejecutable en ROS es la presentada en la Figura 52.

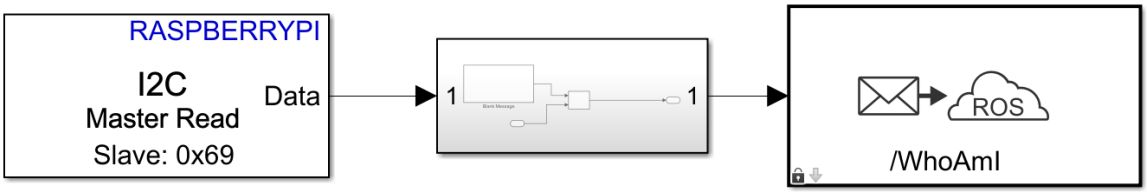


Figura 52: Nodo ROS con paquete de soporte para Raspberry Pi.

Como se puede apreciar, se pretende comprobar si las funciones ofrecidas por el paquete de soporte para Raspberry Pi en Simulink pueden ser convertidas y utilizadas como un nodo en ROS. Para ello, se utiliza el bloque de lectura *I2C Master Read* para obtener el valor presente en la dirección 0x75 de la IMU, correspondiente al registro *WHO_AM_I*, que devuelve los 6 bits más significativos de la dirección I2C de la MPU-6000 (en este caso: 0x68 en numeración hexadecimal, y 104 en decimal). En el fichero también se hace uso de un subsistema encargado de generar el mensaje que será publicado en el *topic* ***/WhoAmI*** gracias al bloque *Publish* perteneciente al paquete *ROS Toolbox*. La estructura presente dentro del subsistema es la misma que puede apreciarse en la Figura 45.

Los resultados obtenidos durante la realización de este ensayo pueden ser apreciados en el apartado 5.2.4.

Tras una obtención de unos resultados positivos durante el ensayo, se ha querido ir un paso más allá y comprobar si la compatibilidad del paquete de soporte para Raspberry Pi de Simulink es compatible también con ROS en Ubuntu Mate, y no solo con Raspberry Pi OS.

El mismo fichero de Simulink mostrado en la Figura 52 fue convertido a un nodo ROS, y ejecutado de forma satisfactoria también en Ubuntu Mate. Tras este resultado, ha quedado comprobado que cualquiera de los dos sistemas operativos (Raspberry Pi OS y Ubuntu Mate) son compatibles con volcados de nodos ROS haciendo un uso íntegro de Simulink y de los paquetes de soporte asociados a la Raspberry Pi; dejando al usuario la elección del sistema operativo según sus preferencias personales.

Capítulo 5. ANÁLISIS DE RESULTADOS

En este capítulo, se va a realizar un repaso completo de los resultados obtenidos a partir de la integración de todos los sensores y actuadores comentados en el Capítulo 4. Siguiendo la estructura llevada a cabo en el resto de esta memoria, este capítulo se dividirá en los resultados obtenidos a través de:

- Matlab y Simulink
- ROS
- Matlab y Simulink con ROS

5.1 MATLAB Y SIMULINK

5.1.1 MEDICIÓN DE LA IMU

En la Figura 53 se muestran los resultados de los trabajos explicados en el apartado 4.2.5.1.1 sobre las medidas obtenidas de la IMU desde Matlab y Simulink exclusivamente. En el gráfico se muestran las respuestas de cada uno de los tres acelerómetros independientes de la MPU-6000.

Ciertas aclaraciones acerca de los datos obtenidos de los acelerómetros de la IMU deben ser ofrecidas para que un análisis de los mismos pueda ser realizado de forma correcta. Como puede observarse en la Figura 53, el offset presente en el eje Z no se corresponde con el valor de la aceleración de la gravedad presente en Madrid (9,81 m/s^2), lugar donde se realizó el ensayo. Esto es debido a que la prueba se realizó con diferentes cables que se encontraban en un estado de tensión elevado; compensando, mediante dicha tensión, la acción de la gravedad en cierta medida. En particular, el offset obtenido durante el ensayo fue de alrededor de 8,6 m/s^2.

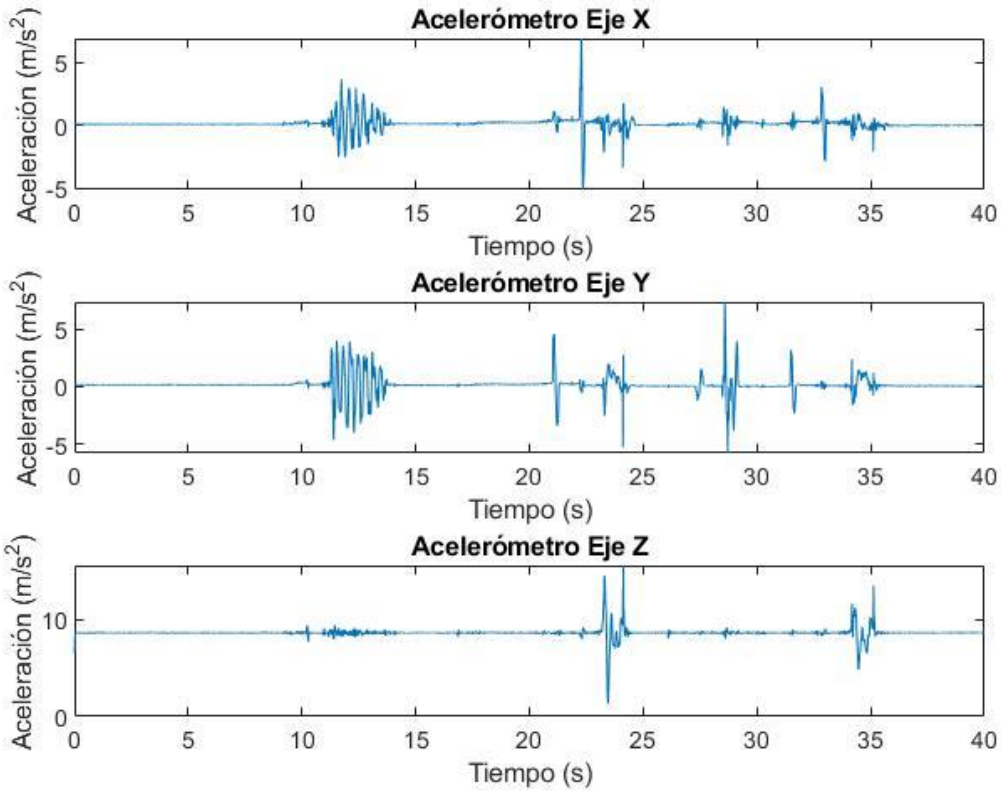


Figura 53: Medición de los acelerómetros de la IMU.

A pesar de esto, se comprobó que el funcionamiento del sensor y el resultado de las medidas se producían de forma correcta. Las oscilaciones presentes en el gráfico son obtenidas como el resultado de diversos movimientos del sensor; gracias a los cuales, se pudo determinar que las mediciones en el conjunto de los ejes se producían de forma satisfactoria.

En términos generales, los resultados obtenidos a través de sucesivos ensayos resultaron ser bastantes satisfactorios. Como se puede apreciar en la Figura 53, la medición se lleva a cabo de forma muy precisa y sin apenas ruido; a pesar de la tensión sufrida por los cables y del hecho de que los movimientos realizados en el sensor se realizaron con la mano, y no sobre el vehículo.

5.1.2 MEDICIÓN DE LOS MD25

En este apartado, se procede a presentar los resultados obtenidos a partir de la medición de los encoders presentes en los motores; tal y como se explica en el apartado 4.2.5.2.1. En este análisis de resultados, se ha hecho hincapié en presentar datos de medición finales como la velocidad de avance o la velocidad angular presente en el eje Z; con el fin de facilitar la comprensión y el análisis de las mediciones obtenidas.

Antes de comenzar con el análisis de los datos obtenidos, es necesario explicar cómo se llevó a cabo el presente ensayo. Antes de comenzar con la prueba, el vehículo fue alzado del suelo permitiendo que la totalidad de las ruedas pudiesen rodar libremente; ya que se pretendía comprobar el correcto funcionamiento del driver diseñado para el control de los drivers MD25. En futuras versiones de este proyecto, será necesario realizar el mismo ensayo con el vehículo situado en el suelo, obteniendo así el valor de parámetros como el peso, los coeficientes de rozamiento y las dinámicas del vehículo; que permitirán la obtención de unas medidas de velocidad más ajustadas al movimiento real del coche.

En el ensayo presente en este apartado, se ha logrado obtener las medidas de la velocidad de avance y la velocidad angular de giro en el eje Z a través del envío de sendos escalones de tensión a los motores. Para ello, se han asignado de forma conjunta los mismos valores a los motores de las ruedas presentes en el hemisferio derecho del vehículo; procediendo de forma análoga con la parte izquierda del mismo. Además, se ha añadido una señal de control que actúa como un *enable* para poder proceder a realizar los cambios de tensión en ambas partes del coche de forma síncrona.

La secuencia de las señales de tensión y del *enable* enviadas a los motores es la mostrada en la Figura 54; sin embargo, con el fin de facilitar la lectura de datos, en la Figura 55 se muestran los escalones de tensión que reciben los motores tras el efecto causado por el *enable*.

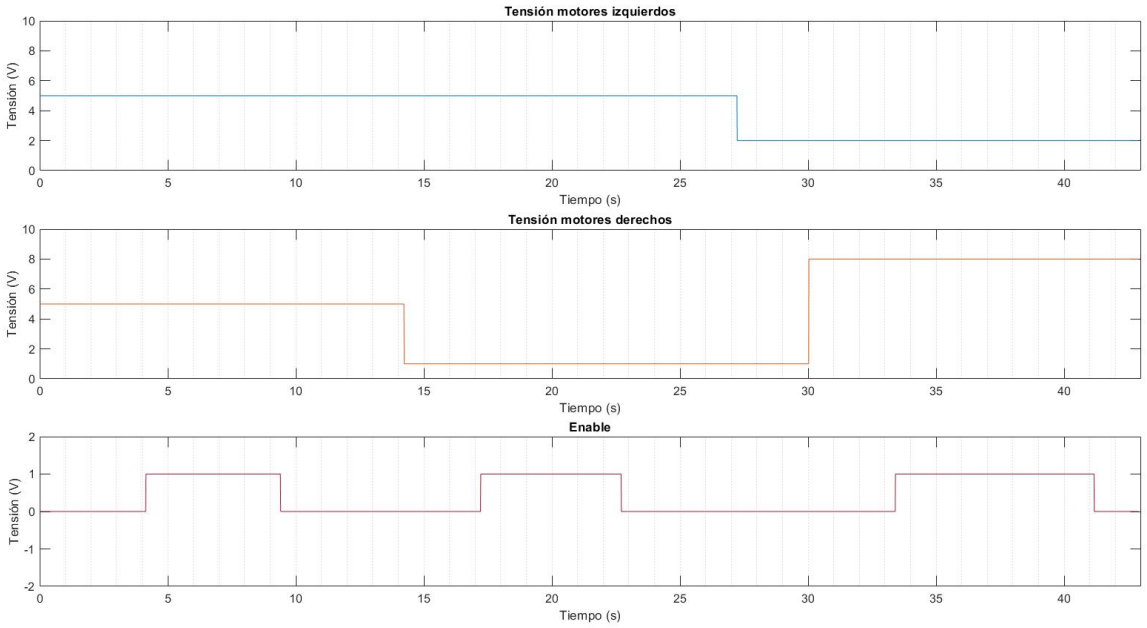


Figura 54: Tensión de los motores con enable.

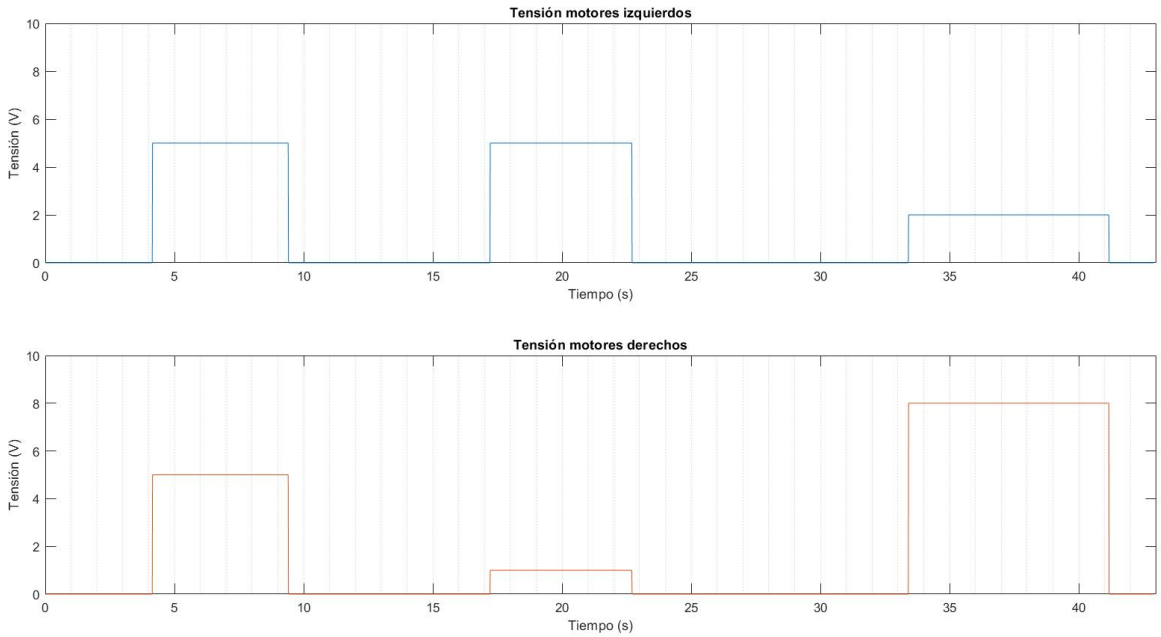


Figura 55: Escalones de tensión aplicados a los motores.

Como se puede apreciar en la Figura 55, los motores izquierdos sufren dos primeros escalones a 5V, y un tercero a 2V; mientras que los motores derechos sufren un primer escalón a 5V, un segundo a 1V, y un tercero a 8V.

El efecto de los escalones de tensión en la velocidad de avance del vehículo puede apreciarse en la Figura 56, donde se muestra la velocidad antes y después de aplicarse el filtro no lineal mencionado en el apartado 4.2.5.2.1. Es posible comprobar también como el filtro no lineal es capaz de eliminar la mayor parte del ruido y los saltos provenientes de la señal original; sin embargo, causa un retardo considerable en la atenuación del pico sufrido al inicio al dar tensión a los motores por primera vez, hecho que no representa problema alguno para el control del vehículo.

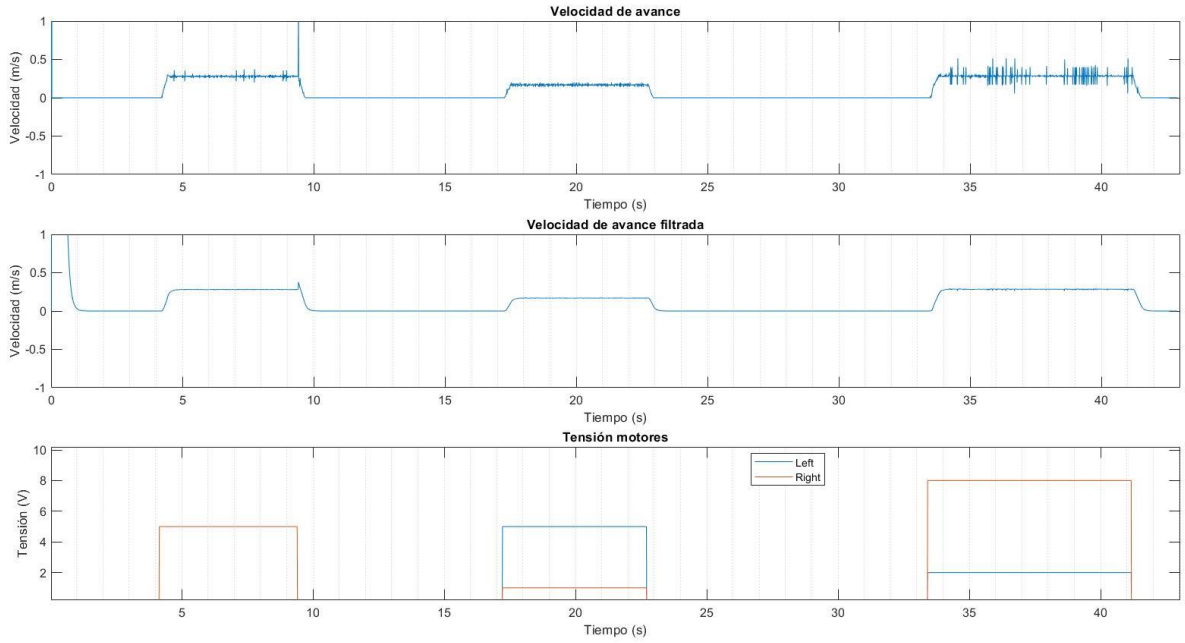


Figura 56: Velocidad de avance del vehículo.

La segunda de las variables de estado mencionadas en el apartado 4.2.5.2.1 correspondiente a la velocidad angular de giro o *yaw rate* también se ve afectada, lógicamente, por los escalones de tensión aplicados a los motores. El efecto causado sobre la velocidad angular de giro del vehículo se encuentra representada en la Figura 57. La corrección aplicada por el filtro no lineal produce resultados muy similares a los apreciados en la Figura 56, con un

retardo inicial en la atenuación del momento correspondiente a la puesta en marcha de los motores; y con un buen desempeño en la reducción del ruido y los saltos pertenecientes a la señal original. Cabe destacar sin embargo la presencia de cierto ruido en el tercer y último escalón de tensión; en el que el giro ensayado se corresponde con un giro realmente brusco, que en operación real del vehículo jamás tendría lugar, dado que se aplican una diferencia de tensiones a ambos hemisferios del vehículo tremendamente exagerada.

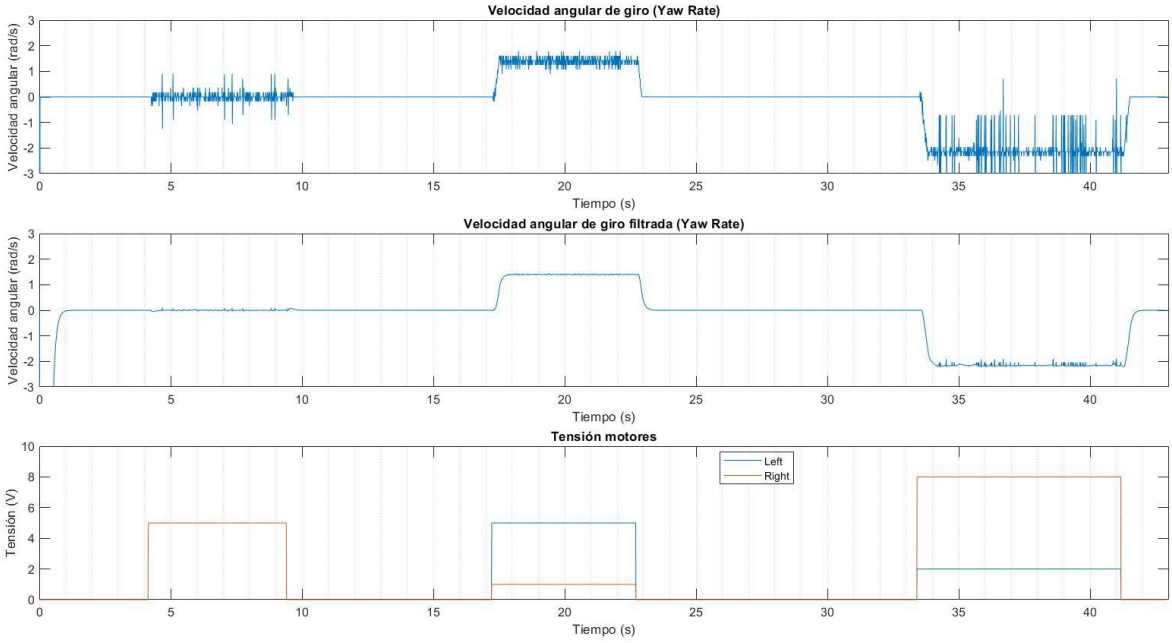


Figura 57: Velocidad de giro (yaw rate) del vehículo.

5.2 ROS

En este apartado se procede a presentar las metodologías llevadas a cabo para la integración en ROS de los drivers de los componentes hardware que instrumentan el vehículo. Es necesario aclarar que todos los resultados, incluidos en este apartado y sus correspondientes

subapartados, se han obtenido exclusivamente a través de la Raspberry Pi, a no ser que se indique lo contrario.

5.2.1 MEDICIÓN DEL LIDAR

En este apartado se van a presentar algunos de los resultados obtenidos a través del sensor del LIDAR. Es de obligada mención, que todas las Figuras aportadas a lo largo de este apartado han sido obtenidas a través de ensayos realizados con el modelo RPLIDAR A1; pero aunque no se muestren en este documento, todas las pruebas realizadas han sido igualmente satisfactorias con el modelo A2M8.

El primer ensayo que se realizó fue a través del software específico para PC facilitado por el fabricante, llamado *FrameGrabber*. Esta primera toma de contacto se ha realizado para asegurar el correcto funcionamiento del LIDAR. Para ello, se estableció una configuración de 4000 muestras por revolución en una habitación cerrada, en la que se pudo obtener la información presente en la Figura 58.

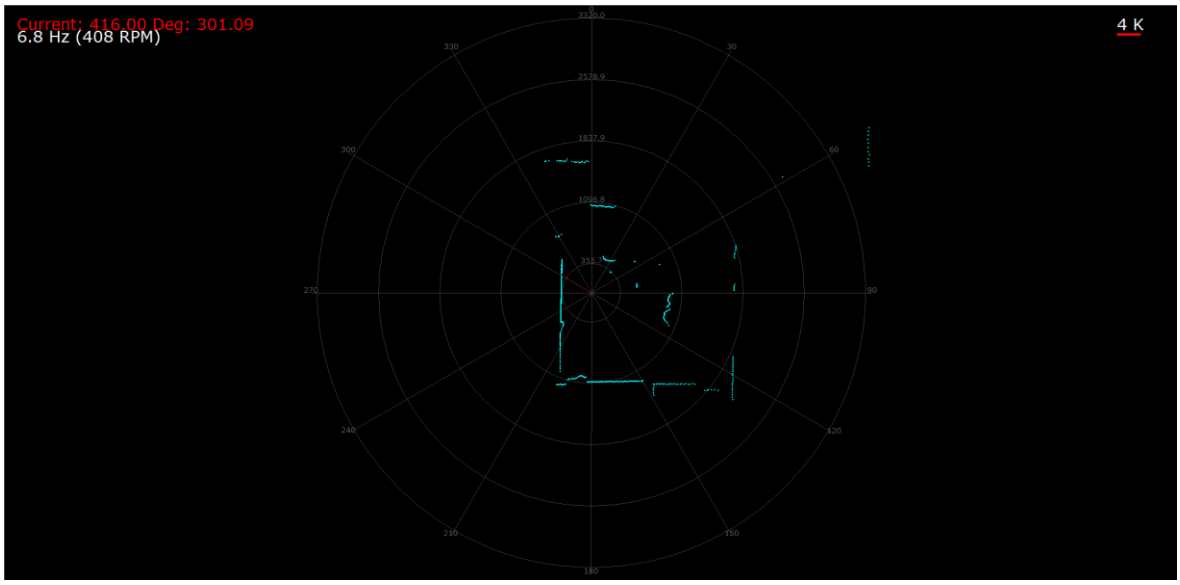


Figura 58: RPLIDAR en FrameGrabber.

Como se puede apreciar en la Figura 58, el software proporcionado por el fabricante es capaz de mostrar la nube de puntos obtenidos por el LIDAR en tiempo real; pudiendo además

seleccionar los diferentes puntos de información para obtener el módulo (mm) y el ángulo (DEG) correspondiente a la medición que representan.

Tras comprobar el correcto funcionamiento del LIDAR con el software oficial, se ha procedido a probar las diferentes funcionalidades ofrecidas por el driver (también oficial) desarrollado para ROS. Como ya se ha explicado en el apartado 4.3.2, la publicación de los datos procedentes del driver se realiza mediante el *topic /scan*. Con el fin de presentar los datos obtenidos de una forma más visual, se va a proceder a mostrar una de las funciones incorporadas en el driver en la que es posible implementar dichas medidas en el entorno gráfico *RViz*; mostrando una nube de puntos de una forma muy similar a la presentada por el *FrameGrabber*. A continuación, en la Figura 59, se muestra la aplicación de dicha función al circuito de pruebas del laboratorio de control de la universidad.

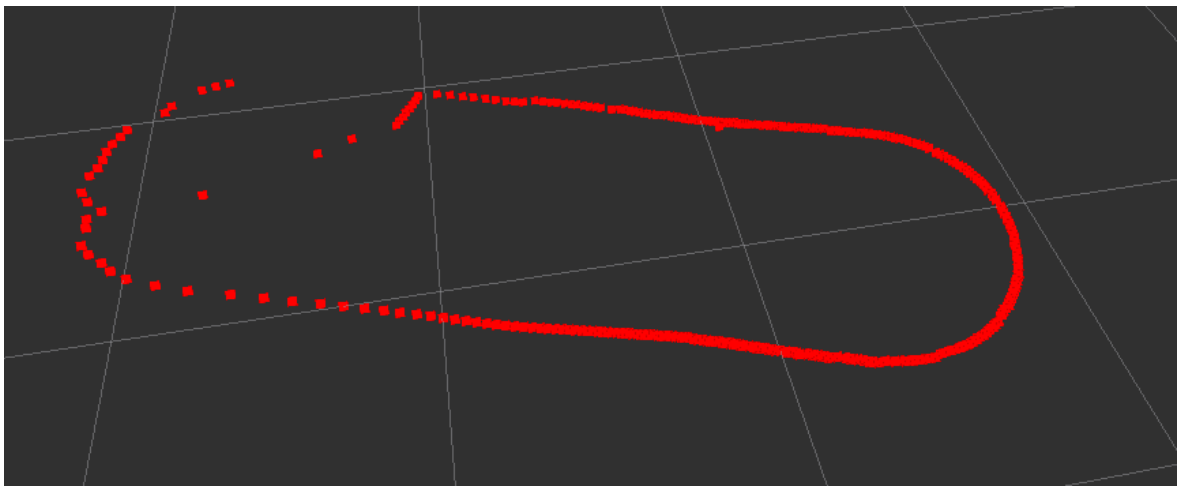


Figura 59: LIDAR ROS en RViz - Circuito.

Por último, se ha probado un algoritmo SLAM que se halla bien documentado en internet, el denominado *HECTOR SLAM*. Dicho algoritmo se encuentra contenido en un paquete descargable para ROS y, haciendo uso de las medidas publicadas por el driver del LIDAR comentado anteriormente, realiza un LIDAR SLAM que puede ser visualizado desde *RViz*. La gran ventaja que aporta este algoritmo es que se encuentra totalmente integrado en ROS y es posible ejecutarlo directamente en la tarjeta SBC; aunque es necesario aclarar que la *Raspberry Pi 3 Model B+* utilizada en este proyecto no ha sido capaz de correrlo. A

continuación, en la Figura 60 y en la Figura 61, se muestran diferentes fases de un mismo ensayo en el que se hace uso del *HECTOR SLAM* en un dispositivo ROS más potente.

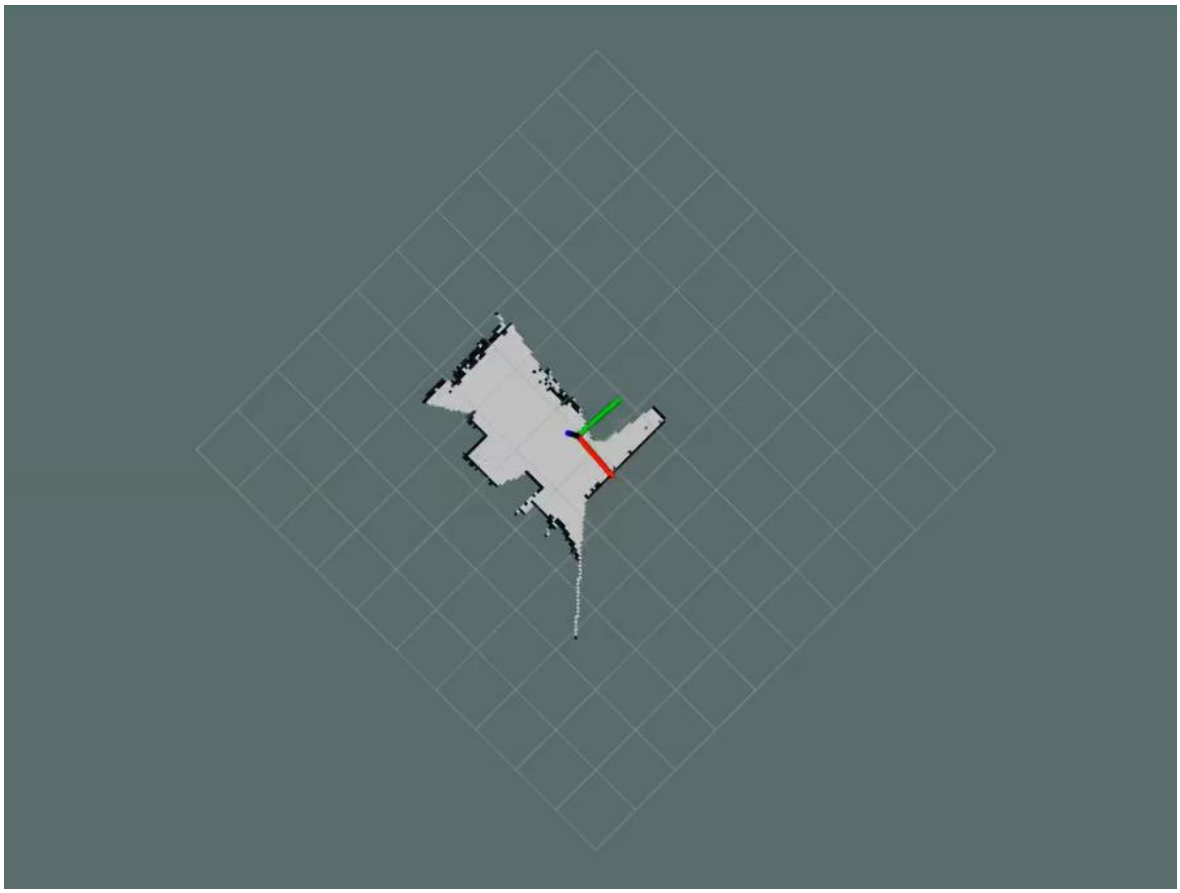


Figura 60: HECTOR SLAM en RViz – Pto. inicial.

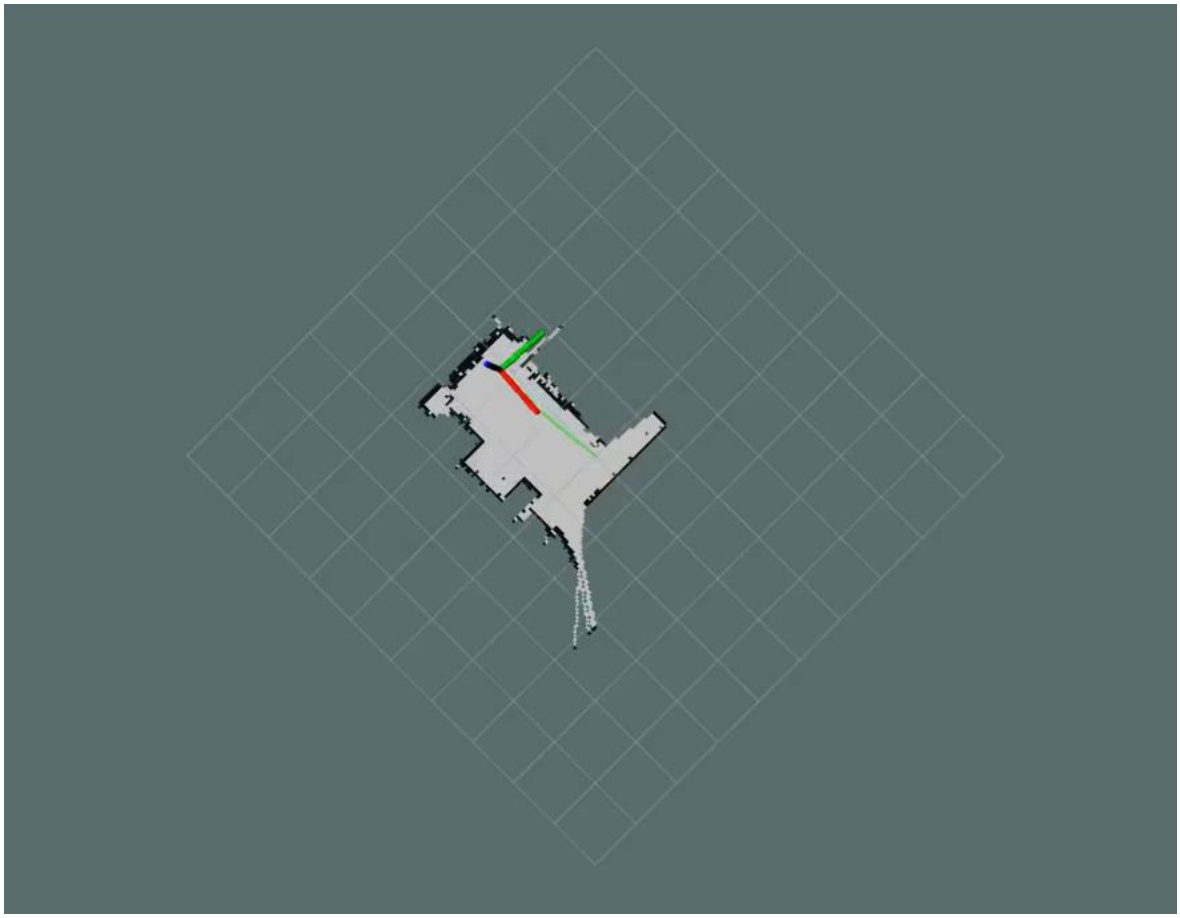


Figura 61: HECTOR SLAM en RViz – Pto. final.

Como se puede comprobar a partir de las figuras mostradas anteriormente, el *HECTOR SLAM* hace uso tanto de la información del sensor laser del LIDAR como de la IMU interna que incorpora, para poder realizar el mapeado de la habitación a la vez que describe la trayectoria y la orientación del vehículo.

5.2.2 MEDICIÓN DEL ENSAYO SIMULINK – ROS

En este apartado se muestran los resultados obtenidos en el fichero de Simulink del PC explicado en el apartado 4.3.3.1. Durante la realización de este ensayo, se ha podido observar que pese a haber conseguido la instalación de un paquete ROS y de un nodo completamente funcional en la Raspberry Pi, existen ciertos problemas de conexión que es necesario solucionar.

Concretamente, se ha observado que desde Simulink se publica el valor escogido en el *topic /Calculo*, pero no se recibe respuesta alguna del *topic /Respuesta*. Más tarde, se comprobó que la información publicada en */Calculo* solo era visible si se realizaba un *rostopic echo /Calculo* desde el propio Matlab; pero que si se ejecutaba el mismo comando desde la Rapberry Pi, no aparecía valor ninguno publicado en */Calculo*. El problema estaba claro que residía en la conexión entre el PC y la Raspberry Pi, al no poderse actualizar el *topic /Calculo* en ambos dispositivos.

Una vez localizado el problema, se comenzó a realizar pruebas de conexión en las diferentes bandas Wifi (2,4GHz y 5GHz) y con diferentes routers del mercado. Se acabó comprobando que para que se pudiese establecer una **conexión fluida** entre los dos dispositivos (PC y Raspberry Pi) presentes en la **red ROS** es necesaria, como mínimo, una conexión por ambas partes en la **banda de 5GHz** facilitada por un **router de gama media-alta**.

En total, se han realizado pruebas hasta conseguir el funcionamiento correcto en dos modelos de router diferentes, los cuales son indicados a continuación, junto con el precio de venta al público estimado[5]:

- Linksys WRT3200ACM – 238€ (Amazon España).
- TP-Link Archer C50 – 29,89€ (Amazon España).

A continuación, en la Figura 62 y en la Figura 63, se puede comprobar cómo se ha conseguido que la conexión entre ambos dispositivos resulte efectiva; haciendo que el valor publicado por el PC en */Calculo* sea triplicado por la Raspberry Pi, publicado en */Respuesta*, y recibido de vuelta en el PC.

[5] PVP a fecha de 10/08/2020.

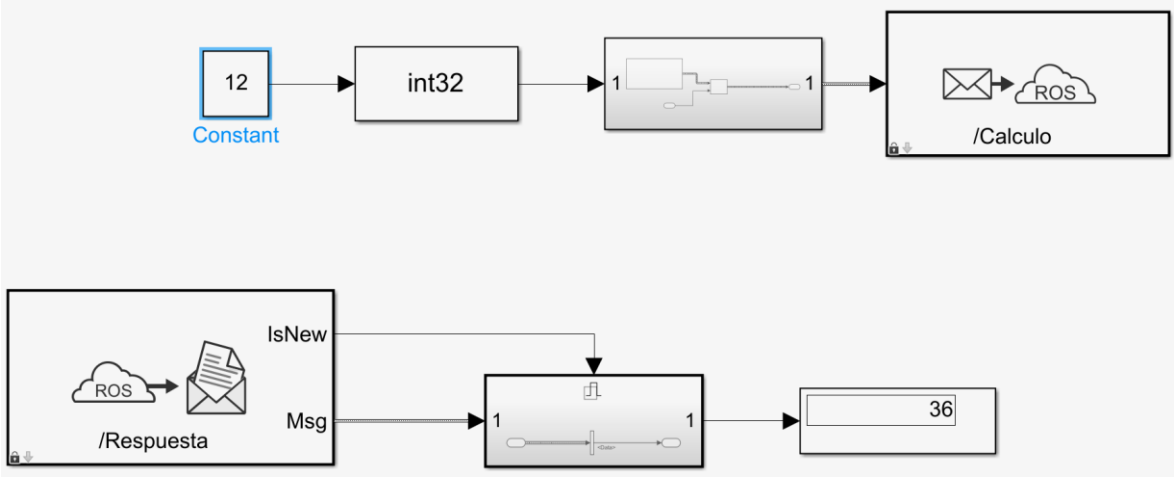


Figura 62: Prueba 1. Ensayo Simulink - ROS.

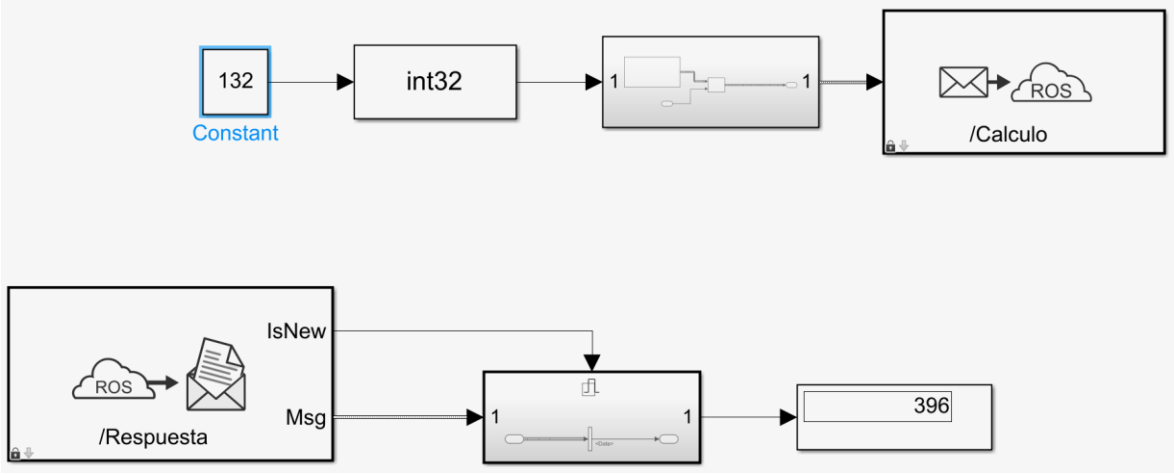


Figura 63: Prueba 2. Ensayo Simulink - ROS.

5.2.3 MEDICIÓN DE LA IMU

En este apartado, se procede a presentar los resultados obtenidos en la IMU mediante el uso del paquete ROS mencionado en el apartado 4.3.4. En el gráfico de la XXX se muestran las mediciones obtenidas en los tres acelerómetros presentes en la MPU-6000. Tras intentar sin éxito la modificación de parámetros de calibrado presentes en el paquete, la Figura 64 muestra con claridad que los datos de aceleración obtenidos finalmente distan mucho de la realidad física. Las medidas de aceleración poseen unos offset inaceptables, siendo el del eje X (valor medio aproximado de 1 m/s^2) y el del eje Y (valor medio de unos 5 m/s^2). Esto es

debido a que dentro de la configuración de calibrado del driver se opta por sustraer el valor de aceleración ejercido por la gravedad; efecto que funciona con relativo éxito en el eje Z, pero que desvirtúa completamente los otros dos.

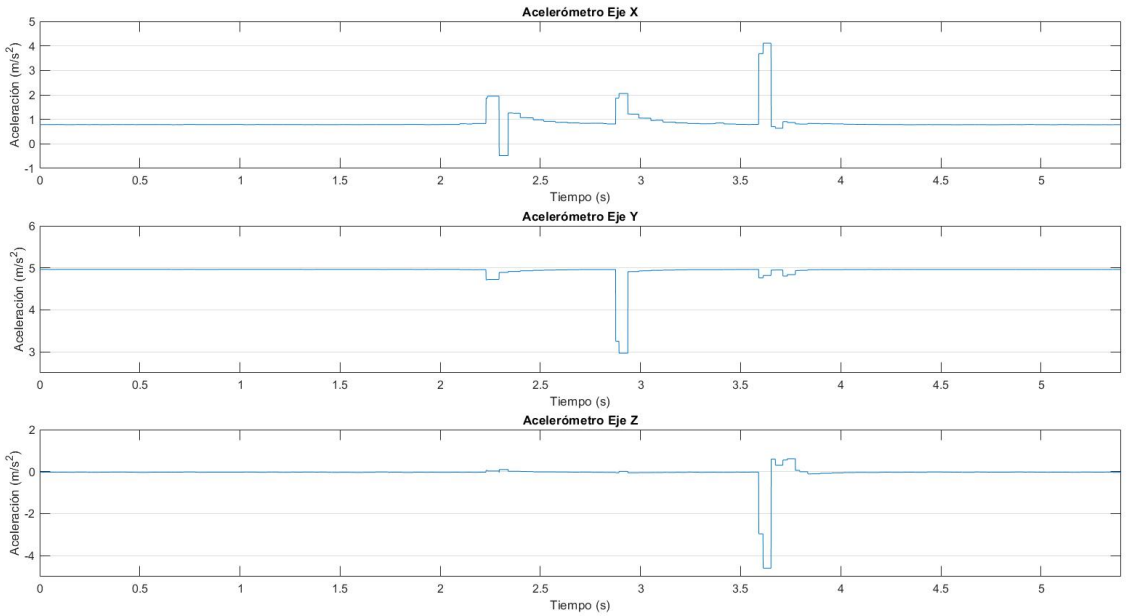


Figura 64: Medición de los acelerómetros de la IMU mediante el paquete ROS de chrisspen.

5.2.4 MEDICIÓN DEL ENSAYO DE SIMULINK CON PAQUETE DE SOPORTE PARA RASPBERRY PI A ROS

Como se ha comentado anteriormente en el apartado 4.3.4.2, los resultados del ensayo mostrado en esta sección han sido verificados de forma correcta tanto en Raspberry Pi OS como en Ubuntu Mate 18.04.2. El ensayo, consiste en la lectura del registro *WHO_AM_I* perteneciente a la IMU, localizado en su dirección 0x75. Como se puede comprobar en la Figura 65, el nodo volcado en la Raspberry Pi, con un bloque perteneciente al paquete de soporte para Raspberry de Simulink, devuelve con éxito al PC el valor predeterminado alojado en el registro *WHO_AM_I* de la MPU-6000.

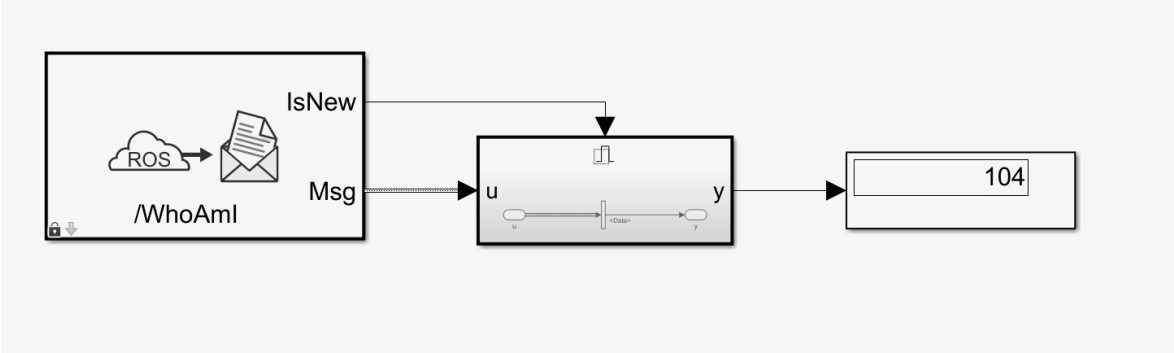


Figura 65: Resultado estructura para PC de nodo ROS con paquete de soporte para Raspberry Pi.

El resultado satisfactorio de este ensayo es de vital importancia para el desarrollo de futuros proyectos; ya que multitud de drivers desarrollados por el laboratorio de la universidad en Simulink (de los que su buen funcionamiento ha sido comprobado) pueden ser ahora volcados y ejecutados como nodos ROS si el proyecto así lo requiere.

Capítulo 6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1 LOGROS GENERALES DEL PROYECTO

- Diseño del vehículo.
- Análisis exhaustivo de las diferentes alternativas hardware en el mercado.
- Instrumentación y adecuación eléctrica de los componentes hardware.
- Adaptación y ensayo del correcto funcionamiento del driver de la IMU MPU-6000 del vehículo a través del paquete de soporte para Raspberry de Simulink.
- Diseño y ensayo del correcto funcionamiento del driver de los MD25 y los motores EMG30 del vehículo a través del paquete de soporte para Raspberry de Simulink.
- Filtrado de las medidas procedentes de los encoders y adecuación de las mismas para su incorporación como variables de control del vehículo.
- Adaptación del driver de los pulsadores y de la máquina de estados en la que participan, a través de Simulink.
- Adaptación y ensayo del correcto funcionamiento del driver del LIDAR presente en el vehículo a través de ROS.
- Adaptación y ensayo del funcionamiento de algoritmos LIDAR SLAM en ROS.
- Análisis extenso y pruebas de funcionamiento y compatibilidad de los diferentes sistemas operativos disponibles para Raspberry Pi con ROS.
- Estudio de integración y compatibilidad entre Simulink y ROS.
- Resolución documentada de diferentes problemas de compatibilidad entre Simulink y ROS en diferentes sistemas operativos para Raspberry Pi.

- Estudio y ensayo de diferentes propuestas de integración de drivers para los diversos componentes hardware de futuros proyectos.

6.2 RECOMENDACIONES

Sin duda uno de los logros más importantes del proyecto ha sido la consecución de la correcta instrumentación eléctrica y computacional de los componentes hardware integrados en el vehículo. Sin embargo, el aspecto más destacable, debido a su influencia en futuros proyectos, es el estudio realizado y los ensayos llevados a cabo sobre las diferentes posibilidades de desarrollo de drivers y su correspondiente viabilidad. Durante este proyecto, se ha trabajado siempre buscando una mejora de los recursos actualmente disponibles en el laboratorio; con el objetivo de generalizar la adaptación de drivers en futuros proyectos, indicando las soluciones más prácticas para cada caso.

Tras la demostración de la viabilidad de conversión de ficheros Simulink a nodos ejecutables en ROS, se recomienda lo siguiente:

Debido a que también ha sido comprobado con éxito el volcado a nodos ROS de diagramas de Simulink con bloques pertenecientes al paquete de soporte para Raspberry tanto en Ubuntu como en Raspberry Pi OS; para proyectos que no requieran de un alto nivel computacional, se recomienda adaptar los drivers existentes en el laboratorio para ser convertidos a nodos ROS exclusivos para el uso de Raspberry Pi.

Sin embargo, la opción más practica y general, dado que sería capaz de funcionar con cualquier tarjeta SBC (lo cual abre la puerta a proyectos más complejos), es la de modificar los drivers existentes **sin hacer uso de ningún paquete de soporte de hardware** para poder ser convertidos a nodos ROS utilizables por cualquier dispositivo. Esta opción debe incluir un concentrador hardware tipo Arduino en el que se realice la programación del conexionado y obtención de datos en crudo de cada componente hardware; para que pueda ser transmitida posteriormente a través de comunicación en serie a cualquier dispositivo con ROS. Esta opción es especialmente interesante porque permitiría el uso de tarjetas SBC más potentes,

capaces de procesar algoritmos de Inteligencia Artificial y modelos de clasificación de imágenes, detección de objetos, segmentación, y procesamiento digital de voz. Además, esta opción sería compatible también con el uso de Raspberry Pi para proyectos más sencillos; eso sí, con la diferencia frente a la opción anterior de tener que añadir un concentrador hardware.

6.2.1 ROS vs ROS2

De cara al desarrollo de futuros proyectos, es importante destacar ciertos problemas presentes en ROS:

- La conexión entre nodos y dispositivos no es 100% en tiempo real; ya que hace uso de TCP y UDP como protocolos de comunicación.
- No posee la opción de establecer niveles de prioridad o *Quality of Service* (QoS).
- Requiere de una conexión de red muy rápida y estable para poder funcionar.
- Los registros de todos los nodos ROS se realizan a través del ROS Master; y si este falla, toda la red ROS cae con él. Esto es comúnmente conocido como *Single Point of Failure*.

Con el objetivo de solucionar los problemas señalados, nace ROS2 en 2013; que además de solventar dichos problemas gracias al uso del protocolo de comunicación DDS, permite la conexión y control de varios robots a la vez (modo "enjambre"). Sería interesante realizar una serie de ensayos para comprobar si las ventajas ofrecidas en la nueva versión compensan el salto de ROS a ROS2; ya que gracias a la existencia de herramientas como *ROSBrigde*, actualmente la conversión de paquetes desarrollados en ROS para que sean compatibles con ROS2 es sencilla.

6.3 FUTUROS PROYECTOS

Dada la instrumentación realizada en este proyecto y la valiosa información obtenida acerca de los mecanismos de compatibilidad entre Simulink y ROS, sería interesante dotar al

vehículo autónomo de ciertos sensores extra. La inclusión de un sistema de dos cámaras, una RGBD y una *Tracking Camera* (ver apartado 2.1.5), habilitaría al vehículo autónomo de la posibilidad de implementar algoritmos avanzados de procesado de imágenes y detección de obstáculos. Sin embargo, para poder conseguir esto, es necesaria la utilización de una tarjeta SBC más potente; en la que se recomienda encarecidamente el uso de una *Jetson Nano* o de un *Intel NUC*, como los propuestos en el apartado 3.2.1.

Otro proyecto interesante para realizar paralelamente al anterior es el estudio del motor físico incluido en ROS: Gazebo. Debido a que es posible customizar, de forma prácticamente infinita, los mapas de simulación con diferentes obstáculos, paredes, carriles, etc.; un trabajo que podría resultar de gran ayuda en el futuro sería el de documentar la integración en un vehículo prediseñado los diferentes sensores, drivers y controles presentes en el laboratorio, con el fin de simular el comportamiento de dicho vehículo en diferentes condiciones físicas. Esto sería de gran ayuda en drones para simular, por ejemplo, la repuesta de los controles a diferentes rachas de viento; pero también sería de gran interés para el desarrollo e implementación de algoritmos de reconocimiento de carriles y señalización visual en un futuro proyecto de vehículo autónomo dotado de cámaras.

Capítulo 7. BIBLIOGRAFÍA

[1] International, SAE, «Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles,» Technical Report J3016_201806, 2018.

[2] NHTSA, «NHTSA Grants Nuro Exemption Petition for Low-Speed Driverless Vehicle,» News, Washington, DC, February 6, 2020.

[3] Nuro, «Delivering Safety: Nuro's Approach,» Technical Brouchure, 2019.

[4] P. Corke, Robotics, Vision and Control. Fundamental Algorithms in MATLAB®, B. Siciliano y O. Khatib, Edits., Springer International Publishing AG, 2017.

[5] The Mathworks, Inc., «Understanding sensor fusion and tracking,» Octubre 2019. [En línea]. Available: https://es.mathworks.com/videos/series/understanding-sensor-fusion-and-tracking.html.

[6] RealPars B.V., «What is the Difference between Absolute and Incremental Encoders?,» 10 Junio 2019. [En línea]. Available: https://www.youtube.com/watch?v=-Qk--Sjgq78. [Último acceso: 23 Mayo 2020].

[7] M. Mihelj, T. Bajd, A. Ude, J. Lenarčič, A. Stanovnik, M. Munih, J. Rejc y S. Šlajpah, Robotics (Second Edition), Springer International Publishing AG, 2019.

[8] Wikipedia Foundation, Inc., «Rotary encoder,» 21 Mayo 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Rotary_encoder. [Último acceso: 1 Junio 2020].

[9] Keyence Corporation, «¿Qué es un senor?,» [En línea]. Available: https://www.keyence.com.mx/download/download/confirmation/?dlAssetId=AS_99468. [Último acceso: 15 Junio 2020].

[10] ArcBotics, «Ultrasonic Range Finder. Learn about Sparki's Ultrasonic Range Finder.,» [En línea]. Available: http://arcbotics.com/products/sparki/parts/ultrasonic-range-finder/. [Último acceso: 15 06 2020].

[11] Robot Gear, «Ultrasonic distance range finder sensors,» Robot Gear Australia, [En línea]. Available: https://www.robotgear.com.au/Category.aspx/Category/24-Ultrasonic-Range-Finders?page=2. [Último acceso: 15 06 2020].

[12] Renishaw plc., «Interferometry explained,» [En línea]. Available: https://www.renishaw.com/en/interferometry-explained--7854. [Último acceso: 01 07 2020].

[13] Zygo Corporation, «3D Optical Profilers,» [En línea]. Available: https://www.zygo.com/?/met/profilers/. [Último acceso: 01 07 2020].

[14] Robot Gear, «Infrared (IR) Distance and Movement Sensors,» [En línea]. Available: https://www.robotgear.com.au/Category.aspx/Category/23-Infrared. [Último acceso: 01 07 2020].

[15] Motorpasión España, «Qué es un LIDAR, y cómo funciona el sensor más caro de los coches autónomos.,» 28 Septiembre 2017. [En línea]. Available: https://www.motorpasion.com/tecnologia/que-es-un-lidar-y-como-funciona-el-sistema-de-medicion-y-deteccion-de-objetos-mediante-laser. [Último acceso: 01 07 2020].

[16] Intel Corporation, «Stereo Depth,» [En línea]. Available: https://www.intelrealsense.com/stereo-depth/. [Último acceso: 10 07 2020].

[17] Intel Corporation, «Open-source SLAM with Intel RealSense depth cameras,» 1 Julio 2019. [En línea]. Available: https://www.youtube.com/watch?v=tcJHnHpwCXk. [Último acceso: 10 07 2020].

[18] C. Pao, «How are Visual SLAM and LIDAR used in Robotic Navigation?,» CEVA's Experts blog, 11 Junio 2019. [En línea]. Available: https://www.ceva-dsp.com/ourblog/how-are-visual-slam-and-lidar-used-in-robotic-navigation/. [Último acceso: 10 07 2020].

[19] The Mathworks, Inc., «Understanding Kalman Filters,» 2017. [En línea]. Available: https://www.youtube.com/playlist?list=PLn8PRpmsu08pzi6EMiYnR-076Mh-q3tWr. [Último acceso: 16 07 2020].

[20] Open Source Robotic Foundation, Inc., «es,» [En línea]. Available: wiki.ros.org/es. [Último acceso: 02 08 2020].

[21] Quanser Inc., «QCAR - Sensor-rich autonomous vehicle for self-driving applications,» Product Datasheet, 2020.

[22] Open Source Robotics Foundation, Inc., «TurtleBot,» [En línea]. Available: https://www.turtlebot.com/. [Último acceso: 18 07 2020].

[23] ROBOTIS, «TurtleBot3,» [En línea]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Último acceso: 18 07 2020].

[24] Raspberrry Pi Foundation, «Raspberry Pi 3 Model B+ product brief,» 03 2018. [En línea]. Available: https://static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+3+Model+B+plus+Product+Brief+PRINT&DIGITAL.pdf. [Último acceso: 23 07 2020].

[25] Infotechnology.com, «Cuáles son las diferencias entre una red Wifi 2.4GHz y una 5GHz y cuál cinviene usar,» [En línea]. Available: https://www.infotechnology.com/online/Cuales-son-las-diferencias-entre-una-red-WiFi-2.4-GHz-y-una-5-GHz-y-cual-conviene-usar-20181022-0008.html. [Último acceso: 24 07 2020].

[26] Wikipedia Foundation, Inc., «LPDDR,» [En línea]. Available: https://en.wikipedia.org/wiki/LPDDR. [Último acceso: 24 07 2020].

[27] Raspberry Pi Foundation, «Raspberry Pi 4 Model B product brief,» [En línea]. Available: https://static.raspberrypi.org/files/product-briefs/200521+Raspberry+Pi+4+Product+Brief.pdf. [Último acceso: 24 07 2020].

[28] Seed Studio, «Best Single Board Compters of 2020,» [En línea]. Available: https://www.seeedstudio.com/blog/2019/11/20/best-single-board-computers-of-2019/. [Último acceso: 25 07 2020].

[29] The Construct, «Using NVIDIA Jetson Nano with ROS,» 16 01 2020. [En línea]. Available: https://www.youtube.com/watch?v=3a_6ENUjgSM. [Último acceso: 25 07 2020].

[30] NVIDIA Corporation, «Módulos y kits de desarrollo Jetson Nano,» [En línea]. Available: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/. [Último acceso: 24 07 2020].

[31] Intel Corporation, «Intel NUC Board NUC8CCHB,» [En línea]. Available: https://www.intel.com/content/www/us/en/products/boards-kits/nuc/boards/nuc8cchb.html. [Último acceso: 24 07 2020].

[32] Intel Corporation, «Intel NUC Programming for the Custom Solutions Header,» Whitepaper v1.0, 2016. [En línea]. Available:

https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/custom-solutions-header-whitepaper.pdf. [Último acceso: 25 07 2020].

[33] J. L. Gómez, «Control de un vehículo autónomo en un entorno limitado,» Universidad Pontificia Comillas, Madrid, 2018.

[34] Alchemy Power Inc., «Pi-EzConnect Product Brief,» 2016. [En línea]. Available: https://cdn-shop.adafruit.com/product-files/2711/EzConnect.pdf. [Último acceso: 27 07 2020].

[35] MikroElectronica , «Pi 3 Click Shield,» [En línea]. Available: https://www.mikroe.com/pi-3-click-shield. [Último acceso: 27 07 2020].

[36] Shanghai Slamtec Co., Ltd.; RoboPeak Team, «RPLIDAR A1 Introductioin and Datasheet,» rev.1.1, 23 03 2018. [En línea]. [Último acceso: 28 07 2020].

[37] Shanghai Slamtec Co., Ltd.; RoboPeak Team, «RPLIDAR A2 Introduction and Datasheet,» rev.1.0, 15 05 2017. [En línea]. [Último acceso: 28 07 2020].

[38] Robot-Electronics, «MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive,» [En línea]. Available: https://www.robot-electronics.co.uk/htm/md25tech.htm. [Último acceso: 28 07 2020].

[39] Robot-Electronics, «EMG30, mounting bracket and wheel specification,» [En línea]. Available: https://www.robot-electronics.co.uk/htm/emg30.htm. [Último acceso: 28 07 2020].

[40] Hobbyplay, «Batería LiPO GENSAce Tattu 1300mAh 11,1V 45C,» [En línea]. Available: https://www.hobbyplay.net/baterias/bateria-lipo-gensace-1300-mah-11v1v-45c. [Último acceso: 29 07 2020].

[41] RS PRO, «SPDT PCB ON-(ON) Push Button Switch,» Datasheet, [En línea]. Available: https://docs.rs-online.com/7601/0900766b81587592.pdf. [Último acceso: 29 07 2020].

[42] RS PRO, «Miniature Toggle Switches,» Datasheet, [En línea]. Available: https://docs.rs-online.com/a996/0900766b81587626.pdf. [Último acceso: 29 07 2020].

[43] Linuxize.com, «How to Add Swap Space on Ubuntu 18.04,» [En línea]. Available: https://linuxize.com/post/how-to-add-swap-space-on-ubuntu-18-04/. [Último acceso: 20 07 2020].

[44] Naciones Unidas, «Objetivos de Desarrollo Sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/. [Último acceso: 29 08 2020].

[45] M. Prieto, «Los robots logísticos de Amazon llegan a España,» Expansión - Unidad Editorial Información General, S.L.U., 26 04 2017. [En línea]. Available: https://www.expansion.com/economia-digital/companias/2017/04/26/590061c2268e3eed088b4679.html. [Último acceso: 29 08 2020].

ANEXO I

Instalación de Ubuntu Mate (se va a instalar la versión 18.04.4) en Raspberry Pi:

1. Descargar la imagen de Ubuntu Mate para Raspberry Pi de 64-bit desde la dirección: https://ubuntu-mate.org/download/

2. Descargar el programa balenaEtcher (Windows/MacOS/Linux) desde la dirección: https://www.balena.io/etcher/

3. Abrir el programa balenaEtcher y seleccionar la imagen de Ubuntu Mate descargada para *flashearla* en la tarjeta microSD. Una vez finalizado el proceso, introducir la tarjeta en la ranura correspondiente en la Raspberry Pi y proceder a su encendido.

4. Es necesario el uso de ratón y teclado en esta parte del proceso. Tras elegir el idioma del sistema, la distribución del teclado (Español – Español), conectarse a la red wifi deseada (se recomienda de una red de 5GHz) y seleccionar la ubicación del usuario en el mapa; introducir un usuario y una contraseña para finalizar la configuración del sistema.

Habilitación de interfaces de conexión de la Raspberry Pi:

```
sudo raspi-config
```

Una vez dentro de la interfaz, habilitar el uso de I2C, SSH y Remote GPIO. Tras esto, reiniciar la Raspberry Pi. Aunque se supone que con estos cambios que se han realizado, el SSH Server debería estar activo; este no se ha habilitado, para ello es necesario seguir los siguientes pasos:

```
sudo apt install openssh-server

sudo apt install sshguard

sudo dpkg-reconfigure openssh-server

sudo reboot
```

Instalación de ROS *Melodic Morenia* desde fuente en la Raspberry Pi:

1. Dirigirse a la siguiente página web: http://wiki.ros.org/melodic/Installation/Ubuntu

2. Seguir los pasos descritos en la web, procediendo a la descarga de *Desktop-Full Install* o *Desktop Install*.

Creación de un espacio de trabajo en ROS, comúnmente conocido como *catkin workspace*:

```
mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make
```

Para no tener que estar actualizando constantemente las dependencias de ROS de nuestro *catkin workspace* cada vez que se abre una terminal nueva, es muy recomendable automatizar dicha actualización mediante el siguiente comando:

```
echo "source /home/javiglez/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

Esto, añade la ubicación del archivo *setup.bash* de nuestro *catkin workspace* (cambiar el nombre de usuario) al final del archivo *.bashrc*.

ANEXO II

Antes de comenzar la instalación, es necesario ampliar el tamaño de la *Swapfile* en Raspberry Pi OS (ampliar el uso de memoria RAM auxiliar alojada en la tarjeta microSD) siempre que no se disponga de un mínimo de 2GB de memoria RAM normal. Para ello se seguirán los siguientes pasos:

Cambiar la configuración de la carpeta **/etc/dphys-swapfile* *:

```
sudo nano /etc/dphys-swapfile
```

El valor por defecto en Raspberry Pi OS (antiguo Raspbian) es:

```
CONF_SWAPSIZE=100
```

Es necesario cambiar este valor al siguiente, para añadir un 1GB extra:

```
CONF_SWAPSIZE=1024
```

Después, será necesario reiniciar el servicio que gestiona el uso de la *Swapfile* en Raspberry Pi OS:

```
sudo /etc/init.d/dphys-swapfile stop

sudo /etc/init.d/dphys-swapfile start
```

Es posible verificar la cantidad de memoria *swap* añadida y disponible en el dispositivo mediante el siguiente comando:

```
free -m
```

El output en pantalla debería parecerse al siguiente:

```
              total        used        free      shared  buff/cache   cached
Mem:            926         154         556          21         215         698
Swap:          1023           0        1023
```

Después de haber realizado este cambio en la memoria *swap*, se procede a la **instalación de ROS *Melodic Morenia* en Raspberry Pi OS**, proceso que durará unas 4-5 horas.

1. **Descargar las dependencias *Bootstrap* y descargar los paquetes:**

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

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt-get update

sudo apt-get install -y python-rosdep python-rosinstall-generator python-wstool
python-rosinstall build-essential  cmake
```

Inicializar *rosdep* y actualizarlo:

```
sudo rosdep init

rosdep update
```

Crear un *catkin workspace* exclusivamente dedicado para la instalación de ROS y acceder a su ubicación:

```
mkdir ~/ros_catkin_ws

cd ~/ros_catkin_ws
```

Instalar la versión de escritorio (recomendada) de ROS *Melodic Morenia*:

```
rosinstall_generator desktop --rosdistro melodic --deps --wet-only --tar >
melodic-desktop-wet.rosinstall

wstool init -j8 src melodic-desktop-wet.rosinstall
```

Si *wstool init* falla o es interrumpido, se puede reanudar la descarga mediante el siguiente comando:

```
wstool update -j4 -t src
```

2. Compilar y ejecutar la instalación:

Usar *rosdep* para instalar las siguientes dependencias:

```
rosdep install --from-paths src --ignore-src --rosdistro melodic -y
```

Compilar e instalar los paquetes *catkin*:

```
sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release -
-install-space /opt/ros/melodic -j2
```

Una vez instalado ROS *Melodic Morenia* en Raspberry Pi OS, se hará source de la nueva instalación en el archivo *~/.bashrc*:

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

3. Crear un *catkin workspace* en el que trabajar y hacer source del mismo:

```
mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make

echo "source /home/pi/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

ANEXO III

Para ampliar el tamaño de la *Swapfile* en Ubuntu 18.04 (ampliar el uso de memoria RAM auxiliar alojada en la tarjeta microSD) es necesario seguir los siguientes pasos:

Comprobar que la instalación de Ubuntu realizada no posee ninguna *Swapfile* habilitada, escribiendo:

```
sudo swapon --show
```

Si el output resulta vacío, significa que el sistema no tiene instalada ninguna memoria swap adicional. Si por el contrario, se obtiene algo parecido a lo mostrado a continuación, es preferible borrar la *swapfile* actual para crear una nueva. (Proceso explicado al final).

```
NAME      TYPE      SIZE USED PRIO
/dev/sda2 partition 1.9G   0B   -2
```

Crear una *swapfile*:

El usuario debe tener acceso a privilegios sudo para poder crear la *swapfile*. Se va a proceder a añadir 1GB de *swap*, si se quisiese añadir más memoria, reemplazar el *1G* por el espacio requerido por el usuario.

```
sudo fallocate -l 1G /swapfile
sudo chmod 600 /swapfile
```

Usar el comando *mkswap* para para habilitar el area swap dentro de la carpeta creada.

```
sudo mkswap /swapfile
```

Activar la *swapfile* mediante el comando:

```
sudo swapon /swapfile
```

Para realizar los cambios de forma permanente, abrir la carpeta */etc/fstab* mediante el comando:

```
sudo nano /etc/fstab
```

y pegar el siguiente contenido:

```
/swapfile swap swap defaults 0 0
```

Verificar que la memoria *swap* ha sido creada con éxito usando el comando *swapon* o el comando *free*, tal y como se muestra a continuación:

```
sudo swapon --show
NAME      TYPE  SIZE   USED PRIO
/swapfile file 1024M 507.4M   -1
```

```
sudo free -h
              total        used        free      shared  buff/cache   available
Mem:           488M        158M         83M        2.3M        246M        217M
Swap:          1.0G        506M        517M
```

Ajustar el valor de *swappiness* a 60:

Swappiness en Linux es una propiedad que define la frecuencia con la que el sistema recurre al uso de la memoria *swap*. Puede tener un valor entre 0 y 100. Cuanto más pequeño sea el valor, más se intentará evitar el uso de dicha memoria, y viceversa.

Para cambiar el valor de *swappiness* a 60, introducir el siguiente comando:

```
sudo sysctl vm.swappiness=10
```

Para hacer este valor permanente, introducir la siguiente línea de código en la carpeta *etc/sysctl.conf* :

```
vm.swappiness=10
```

Para comprobar el valor actual de *swappiness* del sistema, introducir el siguiente comando en la terminal:

```
cat /proc/sys/vm/swappiness
```

Borrar una *swapfile*:

Desactivar el espacio *swap* mediante el comando:

```
sudo swapoff -v /swapfile
```

Borrar de la carpeta */etc/fstab* la línea de código siguiente:

```
/swapfile swap swap defaults 0 0
```

Por último, borrar la actual *swapfile* mediante el comando:

```
sudo rm /swapfile
```

ANEXO IV

Para realizar una correcta conexión entre Matlab-Simulink y ROS en la Raspberry Pi, es necesario llevar a cabo los siguientes pasos:

1. Ejecutar en la terminal de la Raspberry Pi el siguiente código para obtener la dirección IP del dispositivo:

```
ifconfig
```

2. En Simulink, dirigirse a la pestaña *MODELING* y abrir *Model Settings*. Una vez allí, copiar la información mostrada en las siguientes Figuras. **Esta configuración es válida tanto para Ubuntu Mate como para Raspberry Pi OS.**

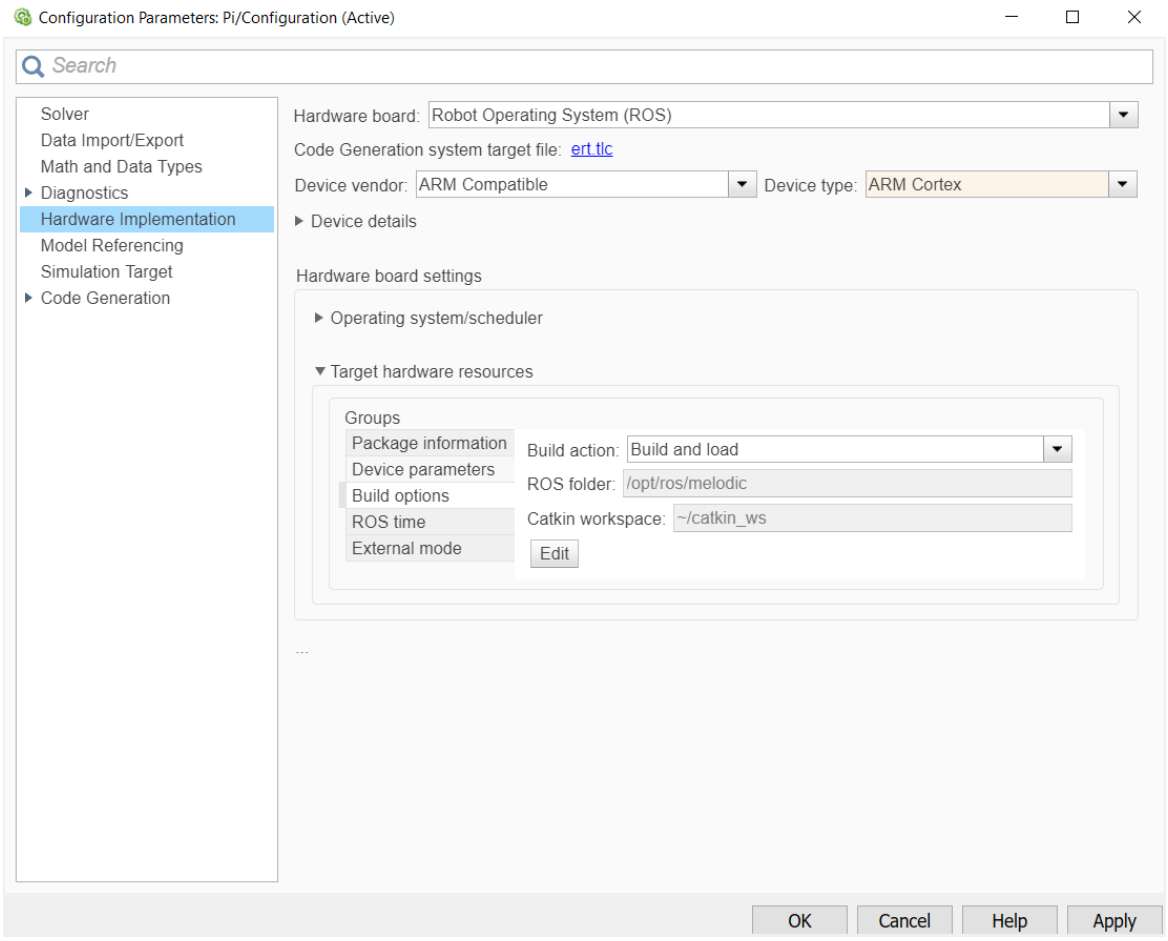


Figura 66: Hardware implementation.

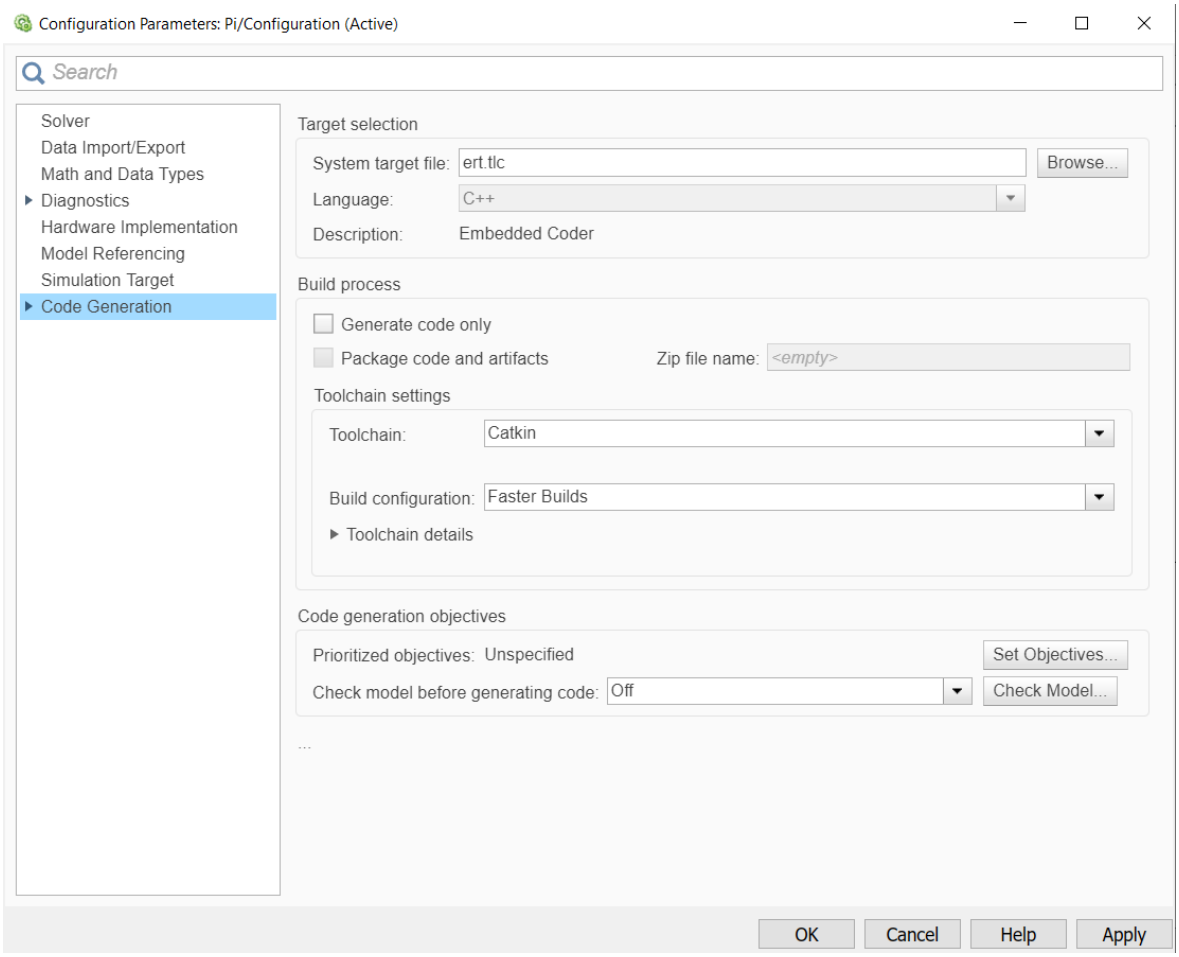


Figura 67: Code Generation.

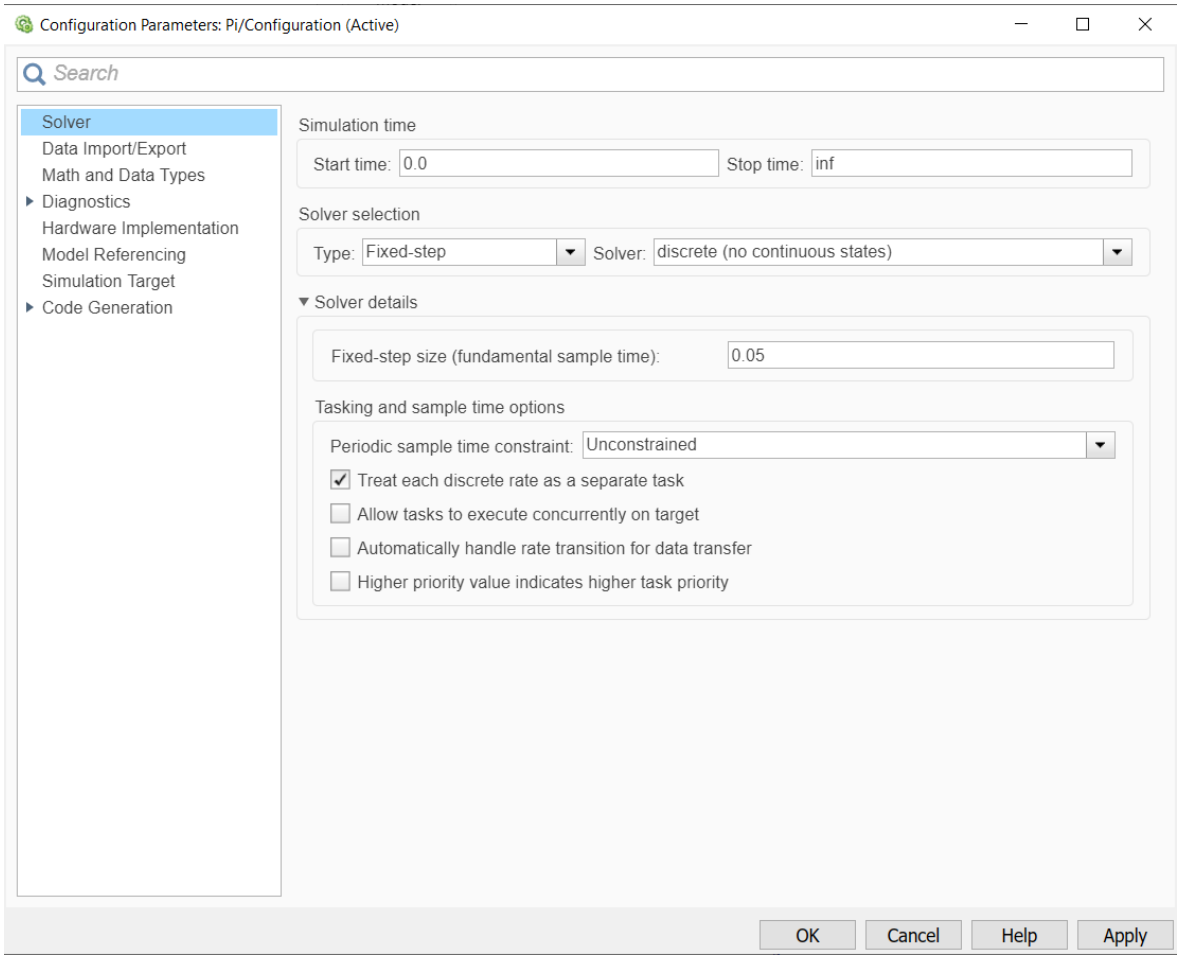


Figura 68: Solver.

3. Tras completar dicha configuración, aparecerá en Simulink una nueva pestaña llamada *ROBOT*, en la que se procederá a abrir la función *Connect to Robot*; y en la que se introducirán la dirección IP de la Raspberry hallada en el Paso 1, el usuario, la contraseña, la carpeta conteniente de ROS y la dirección del *catkin workspace*. El ejemplo de este proyecto puede apreciarse en la Figura 69 mostrada a continuación:

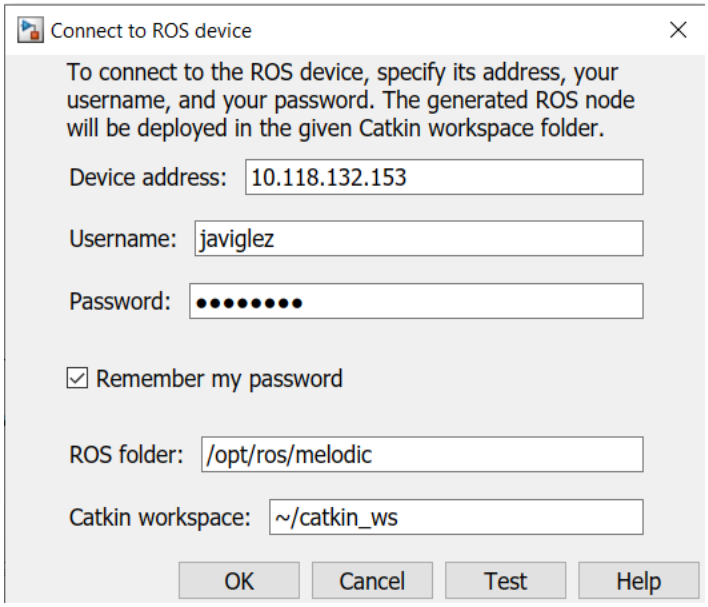


Figura 69: Connect to Robot.

4. Tras completar la información requerida, y darle a *OK*; se inicializará en la terminal de la Raspberry Pi un nodo Master escribiendo:

```
roscore
```

En la terminal aparecerá la dirección del ROS Master como *ROS_MASTER_URI*, de la cual solo interesará el puerto utilizado, que suele ser el *11311*.

5. Tras conocer el puerto del Ros Master, dirigirse a la pestaña de Simulink *Simulation*, en la cual se abrirá la función *ROS Network*. En ella se introducirá de nuevo la dirección IP de la Raspberry Pi y el puerto perteneciente al ROS Master, en el caso de este proyecto:

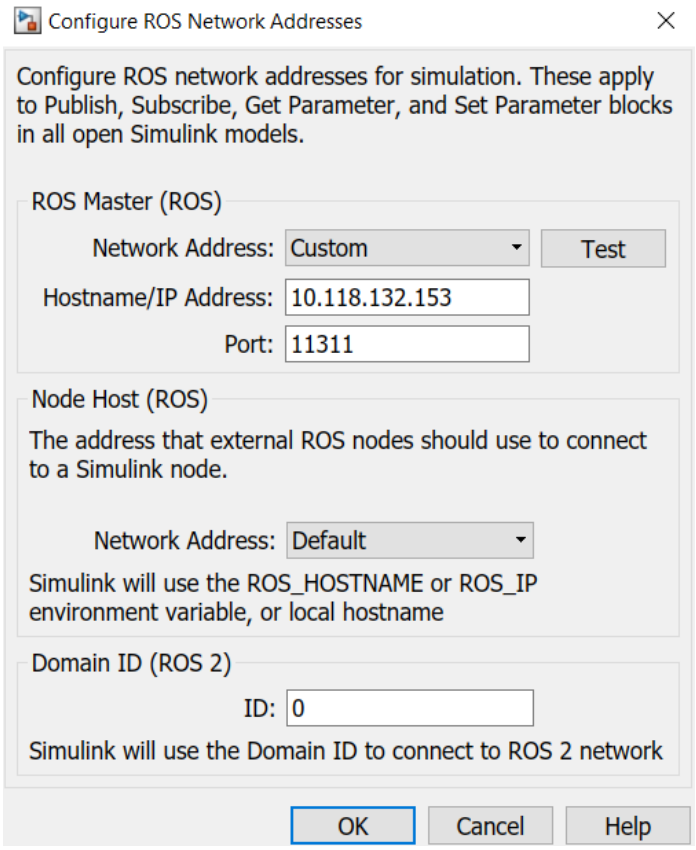


Figura 70: ROS Network.

6. Tras esto, verificar la conexión mediante la función *Test Connection* presente en la pestaña *ROBOT*. Dicho proceso deberá finalizar con la totalidad de la barra de procesos en azul y con el mensaje de *Done with connection test*, indicativo de que la conexión se ha establecido correctamente.

7. Por último es necesario que Matlab se conecte a la red de ROS como un nodo nuevo; para lo cual es necesario introducir en el *Command Window* el siguiente comando, introduciendo dentro de las comillas la dirección IP asignada a la Raspberry Pi utilizada, que en este proyecto ha resultado ser:

```
rosinit('10.118.132.153')
```

ANEXO A. EL PROYECTO Y EL DESARROLLO SOSTENIBLE DE LA ONU

En 2015, la ONU aprobó la Agenda 2030 sobre el Desarrollo Sostenible, que cuenta como parte protagonista con diecisiete Objetivos de Desarrollo Sostenible (ODS) enfocados a proporcionar una oportunidad para que los países y sus sociedades emprendan un nuevo camino con el que mejorar la vida de todos, sin dejar a nadie atrás. Los objetivos cubren problemas mundiales en un amplio espectro, desde la eliminación de la pobreza hasta el combate al cambio climático, la educación, la igualdad de la mujer, la defensa del medio ambiente o el diseño de las ciudades. [44]

Uno de los objetivos ODS apoyados por este proyecto es el correspondiente al número 4, encaminado a garantizar una educación inclusiva, equitativa y de calidad, y promover oportunidades de aprendizaje durante toda la vida para todos. En concreto, dado que el proyecto desarrollado consiste en la instrumentación de un vehículo de tamaño reducido enfocado a la investigación y desarrollo de las tecnologías de conducción autónoma, podría contribuir de forma activa en la educación de múltiples alumnos; asegurando la adquisición de conocimientos técnicos y prácticos necesarios para promover el desarrollo sostenible.

Un desarrollo sostenible que ayudaría a alcanzar principalmente dos de los diecisiete ODS propuestos por la ONU: el número 11, correspondiente a lograr ciudades más inclusivas, seguras, resilientes y sostenibles; y el número 9, enfocado a la construcción de infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación.

En concreto, en relación con el ODS 9, el proyecto podría resultar de especial utilidad a la hora de facilitar una modernización de la infraestructura de la industria, promoviendo el desarrollo de tecnologías industriales más eficientes, limpias y sostenibles; mejorando también la seguridad de algunos trabajadores. Es muy difícil cuantificar el impacto que

podría tener este proyecto en la implantación de robots autónomos en la industria; pero sí que se pueden extraer ciertos paralelismos respecto a los beneficios que han supuesto estos robots en la industria recientemente. Un ejemplo podrían ser los 3.500 robots autónomos especializados en logística introducidos por Amazon España, que no solo han aumentado la capacidad de almacenaje en más de un 50% en la planta de El Prat, debido a su capacidad de optimización del espacio disponible; sino que han permitido la creación de alrededor de 500 puestos de trabajo en los últimos tres años. [45]

Por último, al facilitar una plataforma sobre la que investigar y desarrollar nuevos algoritmos de conducción autónoma y contribuir al desarrollo de vehículos autónomos comerciales, se lograrán ciudades más seguras, sostenibles y respetuosas con el medio ambiente; metas incluidas en el ODS número 11 propuesto por la ONU. Es complicado cuantificar el impacto de este proyecto en el desarrollo del sector del automóvil autónomo; pero sí que se pueden realizar ciertas afirmaciones en relación con lo que supondría el establecimiento permanente de este tipo de vehículos en las ciudades. Los accidentes de tráfico se reducirían casi en su totalidad, y la contaminación causada por los coches con motor de combustión interna sería erradicada de las grandes urbes, ya que la industria del vehículo autónomo se encuentra estrechamente vinculada al sector del vehículo eléctrico; contribuyendo en gran medida a la conservación del medioambiente y a la salud de sus ciudadanos. La instauración de este tipo de vehículos como utilitarios habituales provocaría una expansión de las áreas urbanas al permitir aprovechar el tiempo utilizado en los desplazamientos en coche en tareas relacionadas con el ámbito laboral; permitiendo horarios más flexibles y provocando una reducción de densidad demográfica en las ciudades. Al permitir a los ciudadanos vivir en zonas residenciales a las afueras de las grandes urbes, no solo contribuiría a un aumento en su calidad de vida; sino que permitiría también una bajada de precios generalizada de las viviendas situadas en el centro de la ciudad, reactivando el sector urbanístico y mejorando el acceso a la vivienda del ciudadano medio.