COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

SISTEMA DE CONTROL PARA EL SEGUIMIENTO DE PARED BASADO EN UN SENSOR LIDAR 2D

Autor: Jesús Vidal Sánchez

Director: Diego Cubillo Llanes

Co-Director: Juan Luis Zamora Macho

Madrid

Julio de 2024

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

Sistema de control para el seguimiento de pared basado en un sensor Lidar 2D

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

curso académico 2023/2024 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.: Jesús Vidal Sánchez Fecha: 01/07/2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Diego Cubillo Llanes Fecha: 01/07/2024

Fdo.: Juan Luis Zamora Macho Fecha: 02/07/2024

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

SISTEMA DE CONTROL PARA EL SEGUIMIENTO DE PARED BASADO EN UN SENSOR LIDAR 2D

Autor: Jesús Vidal Sánchez

Director: Diego Cubillo Llanes

Co-Director: Juan Luis Zamora Macho

Madrid

Julio de 2024

SISTEMA DE CONTROL PARA EL SEGUIMIENTO DE PARED BASADO EN UN SENSOR LIDAR 2D

Autor: Vidal Sánchez, Jesús.

Co-Director: Cubillo Llanes, Diego.

Co-Director: Zamora Macho, Juan Luis.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El objetivo del proyecto es el desarrollo de un control de seguimiento de pared para implantarlo en un vehículo terrestre. Los datos de las medidas se obtendrán de un sensor LiDAR 2D. Tras su posterior procesado, se utilizarán en el sistema de control para que el vehículo sea capaz de recorrer un circuito con tramos tanto rectos como curvos, adaptando su distancia a la pared para respetar el valor de referencia fijado. La puesta en marcha del LiDAR, el procesado de sus datos y su respectivo envío por protocolo TCP/IP, se realizará en una Raspberry Pi 4. Por otra parte, todos los algoritmos y procedimientos referidos al control de seguimiento se realizarán en una Raspberry Pi 3B+.

Palabras clave: Seguimiento de pared, ROS2, Python, LiDAR, Matlab, Simulink, Raspberry Pi, navegación autónoma, vehículo autónomo.

1- Introducción:

La transformación de la industria, en búsqueda de un modelo cada vez más eficiente, ha impulsado nuevas disciplinas como la navegación autónoma. Debido a la necesidad de automatizar los procesos de producción, el desplazamiento autónomo de las máquinas que participan en ellos ha cobrado gran importancia. Ejemplo de esto son los Vehículos de Guiado Autónomo o AGVs. Estas máquinas son capaces de obtener datos de su entorno gracias a una gran variedad de sensores, que les permiten seguir trayectorias previamente definidas y tomar decisiones acerca de su desplazamiento teniendo en cuenta los posibles obstáculos que pueden llegar a encontrarse [1]. Además, la cuarta revolución industrial explora técnicas de mejora de la navegación autónoma aplicando tecnologías emergentes, como el Internet of Things (IoT) o la Inteligencia Artificial

Es debido al creciente interés en esta materia que este proyecto pretende diseñar un sistema de control de seguimiento de pared para un vehículo terrestre usando como fuente de información sobre el entorno un sensor LiDAR, una tecnología cada vez más presente en los proyectos de navegación autónoma.

2- Objetivos del proyecto

Para la realización del proyecto se definieron los siguientes objetivos:

- Obtención de un sistema funcional en tiempo real que informe de la distancia y ángulo relativo a la pared. El sensor LiDAR es la principal fuente de información de la posición relativa del vehículo a la pared que se pretende seguir. El hecho de

que estás variables representen fielmente la posición real del vehículo sienta las bases para el correcto funcionamiento de los algoritmos de control, ya que son precisamente las variables que se pretenden regular.

- Generación de un software basado en ROS2 capaz de gestionar el funcionamiento del sensor y la comunicación de medidas con el ordenador de control del vehículo. Se desarrollará un software basado en nodos ROS que será capaz de gestionar la puesta en marcha del sensor LiDAR, la recepción de los datos de interés, su posterior formateo y envío al ordenador de control del robot.

- Correcto funcionamiento del robot en el seguimiento autónomo de pared mediante la comprobación por ensayos en el laboratorio. Una vez acabada la implementación del proyecto se comprobará su correcto funcionamiento mediante pruebas, asegurando que el comportamiento es el esperado. El vehículo deberá ser capaz de recorrer un circuito con tramos de pared tanto rectos como curvos. Estas características afectan al control de ángulo y distancia a la pared. Además, los tramos de pared recta presentan secciones inclinadas, que ponen a prueba la capacidad del control de velocidad del vehículo de lograr un avance a la velocidad de referencia especificada independientemente de la inclinación de la pista.

3- Solución

El proyecto está basado en un vehículo de tracción diferencial equipado con diferentes componentes hardware. Esto incluye un sensor RPLIDAR modelo A2M8 de SLAMTEC, encargado de medir la distancia del vehículo a la pared de referencia. También incluye dos Raspberry Pi, una modelo 3B+ y otro modelo 4, una IMU (Inertial Measurement Unit), diversos pilotos LED para dar información del estado del vehículo, dos motores de corriente continua para el movimiento de las ruedas y tres pulsadores para su operación por parte del usuario, entre otros.

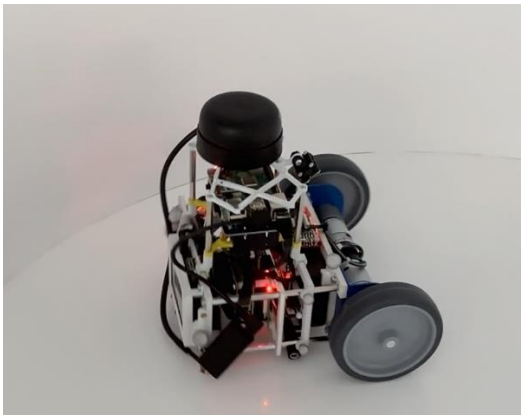


Figura 1. Vehículo de tracción diferencial respondiendo ante la curva del circuito de ensayos.

En cuanto a la etapa de desarrollo del software podemos distinguir dos partes bien diferenciadas: la obtención y envío de las medidas del LiDAR por un lado y los algoritmos de control por otro.

La primera parte, se basa en nodos ROS2 (Robot Operating System), que son programas en este caso escritos en Python. Hacen posible la puesta en marcha del LiDAR, la obtención de medidas de este, la transformación de los datos a un formato deseado y su posterior envío a la Raspberry Pi 3B+ para seguir con el control.

La segunda, hace uso del software MATLAB/SIMULINK. Recibe la información del mensaje de medidas ya formateado y hace uso de esta para estimar la distancia y ángulo relativo del coche respecto a la pared. Estas estimaciones se usarán como señales dentro de los algoritmos de control que, a su vez, modificarán la tensión en los motores para orientar al coche en la dirección correcta.

4- Resultados

Durante el desarrollo del proyecto, se ha verificado el funcionamiento de los componentes del sistema con diferentes pruebas:

En primer lugar, antes de proceder con el diseño del control de pared, se comprobó que la información que enviaba el sensor LiDAR era coherente. Para ello se generó un mapa del circuito de pruebas para ver si el contorno era reconocible. Además, se monitorizaron las dos señales esenciales para los algoritmos de control, la distancia y el ángulo relativo del coche a la pared.

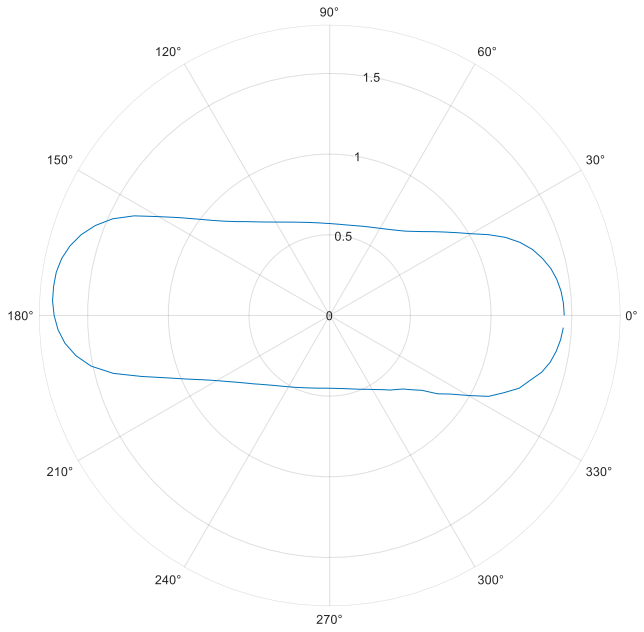


Figura 2. Mapa del circuito de ensayos del laboratorio

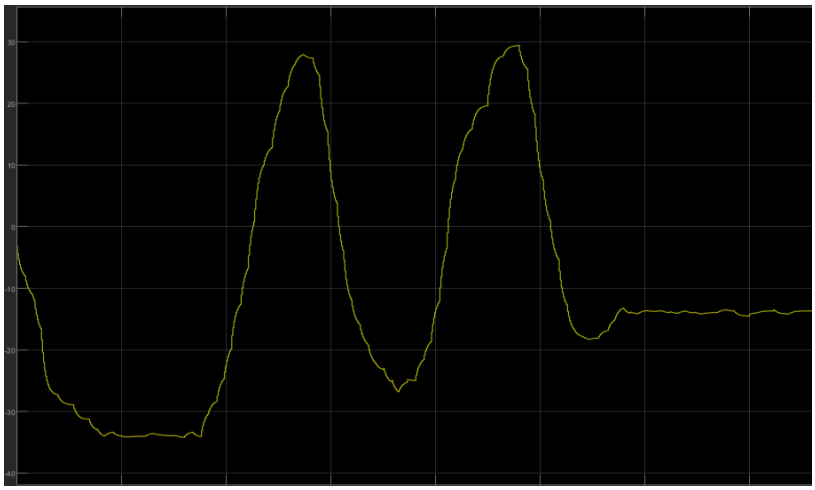


Figura 3. Ensayo de monitorización de la señal de distancia a la pared.

En segundo lugar, antes de dar por finalizado el proyecto, se comprobó que el control respondía correctamente. Se capturó la respuesta a una referencia variable para ver cómo se adaptaba la señal de distancia a la pared a los cambios en referencia. El resultado fue positivo. Como podemos ver en las gráficas obtenidas, tanto el control de velocidad, gráfica Forward Velocity, como el control de distancia de pared, Wall Dist, funcionan siguiendo la referencia especificada para cada caso.

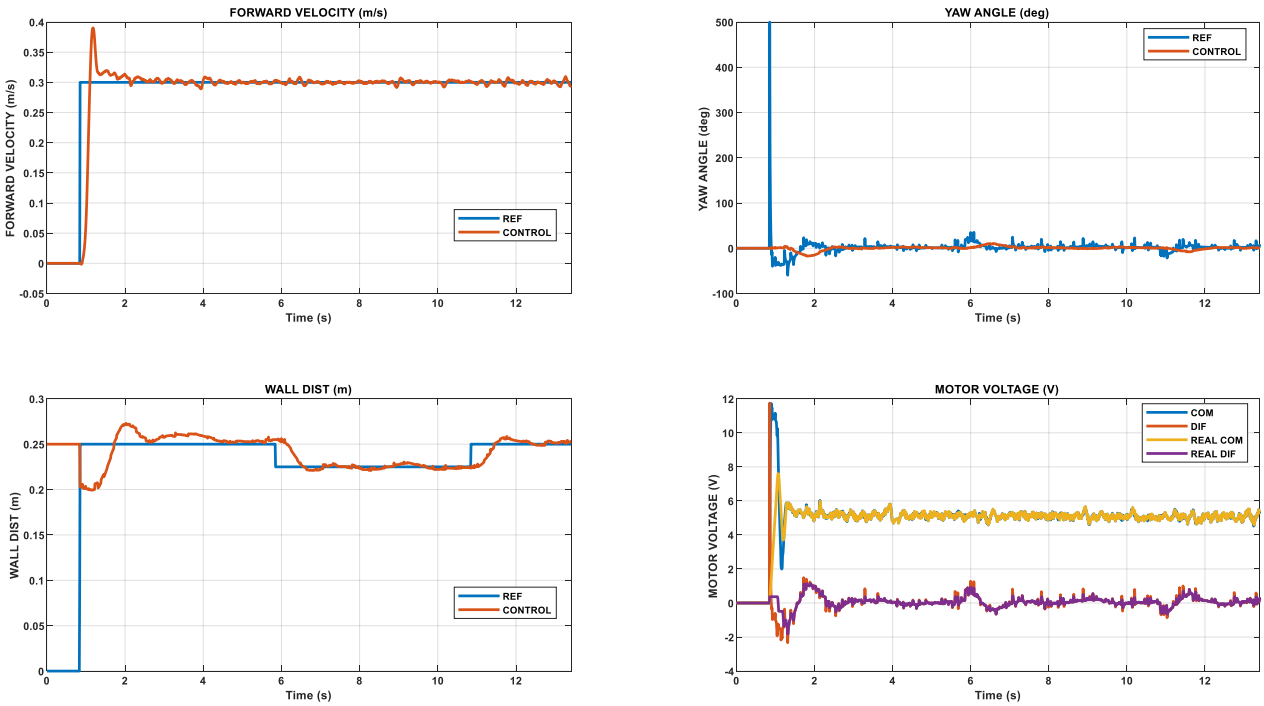


Figura 4. Ensayo con referencia variable para comprobar transitorios.

5- Conclusión

Podemos concluir tras los resultados obtenidos que los objetivos marcados al inicio del proyecto se han completado con éxito.

El software desarrollado consigue obtener medidas coherentes del sensor haciendo posible obtener información fiable de la morfología del entorno.

A su vez, el control de seguimiento de pared también es correcto. Los ensayos de seguimiento en el circuito de ensayos han sido exitosos y el vehículo consigue mantener constante su separación con la pared y recuperar su orientación en los tramos curvos para poder evitar el choque con esta.

Para trabajos futuros se propone explorar nuevas técnicas de creciente popularidad en proyectos de navegación autónoma como SLAM (Simultaneous Location and Mapping). Esto permitiría al robot entender la morfología de su entorno y ser capaz de trazar trayectorias considerando todo el tramo de pared que debe seguir. Otra alternativa sería implementar nuevos sensores en el vehículo, como cámaras de video, para poder obtener más información del entorno.

6- Referencias

[1] Madrigal Moreno, S. A., & Muñoz Ceballos, N. D. (2019). *Vehículos de guiado autónomo (AGV) en aplicaciones industriales: una revisión.* Revista Politécnica, 15(28),117-137.[fecha de Consulta 25 de Febrero de 2024]. ISSN: 1900-2351. Recuperado de: https://www.redalyc.org/articulo.oa?id=607866567012
Último acceso: 04/2024

WALL TRACKING CONTROL SYSTEM BASED ON A 2D LIDAR SENSOR

Author: Vidal Sánchez, Jesús.

Codirector: Cubillo Llanes, Diego.

Codirector: Zamora Macho, Juan Luis.

Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

The aim of the project is the development of a wall tracking control to be implemented in a land vehicle. Measurement data will be obtained from a 2D LiDAR sensor. After subsequent processing, these data will be used in the control system to enable the vehicle to traverse a circuit with both straight and curved sections, adjusting its distance to the wall to respect the set reference value. The startup of the LiDAR, processing of its data, and its respective transmission via TCP/IP protocol will be carried out on a Raspberry Pi 4. Furthermore, all algorithms and procedures related to the tracking control will be executed on a Raspberry Pi 3B+

Keywords: Wall tracking control, ROS2, Python, LiDAR, Matlab, Simulink, Raspberry Pi.

1- Introduction:

The transformation of the industry, in pursuit of an increasingly efficient model, has driven the emergence of new disciplines such as autonomous navigation. Due to the need to automate production processes, autonomous robot movement and navigation have gained great importance. An example of this is Autonomous Guided Vehicles (AGVs). These machines can gather data from their environment through a variety of sensors, which enables them to follow predefined trajectories and make decisions regarding their movement accounting for possible obstacles they may encounter [1]. Moreover, the fourth industrial revolution is exploring improvement techniques based on emerging technologies such as the Internet of Things (IoT) or Artificial Intelligence (AI).

It is due to the growing interest in this field that this project aims to design a wall tracking control system for a land vehicle using a LiDAR sensor as a source of environmental information. LiDAR technology is increasingly prevalent in autonomous navigation projects.

2- Project Objectives:

For the completion of the project, the following objectives were defined:

- Development of a real-time functional system that provides information on the distance and relative angle to the wall. The LiDAR sensor is the main source of information about the relative position of the vehicle in regard to the wall. The fact that these variables accurately represent the true position of the vehicle is the

foundation for the proper functioning of the control algorithms, as these are the variables to be controlled.

- Creation of ROS2-based software capable of managing the sensor's operation and communication with the vehicle's control computer. Dynamic ROS nodes-based software will be developed to handle the startup of the LiDAR sensor, reception of relevant data, their subsequent formatting, and transmission to the robot's control computer.

- Ensuring the robot's correct operation in autonomous wall tracking through laboratory testing. Once the project implementation is completed, its proper functioning will be verified through tests, ensuring that the behavior meets the expected standards. The vehicle will have to be able to traverse a circuit with both straight and curved wall sections. These curvature features affect the wall following control, forcing the robot to adapt its orientation to modify its distance. Moreover, the straight-wall sections feature inclined segments, which test the vehicle's velocity control to keep the velocity to the constant reference value regardless of the circuit inclination.

3- Solution

The project is based on a differential drive vehicle equipped with various hardware components. This includes an RPLIDAR model A2M8 from SLAMTEC, responsible for measuring the distance from the vehicle to the reference wall. Additionally, two Raspberry Pi boards are employed, one model 3B+ and the other model 4, an Inertial Measurement Unit (IMU), several LED indicators to provide vehicle status information, two DC motors for wheel movement, and three push buttons for the user operation interface, among other components.

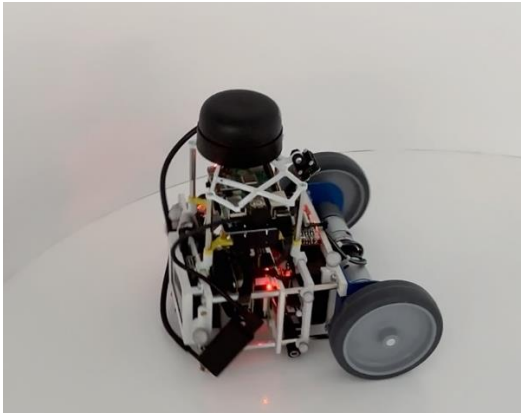


Figure 1. Differential traction vehicle facing a curved section of the trial circuit.

Regarding the software used, there are two distinct parts: obtaining and sending LiDAR measurements on one hand, and control algorithms on the other.

The first part is based on ROS2 (Robot Operating System) nodes, which are programs written in Python in this case. They enable the startup of the LiDAR, obtain measurements from it, transform the data into the desired format, and subsequently send it to the Raspberry Pi 3B+ to continue with the control process.

The second part utilizes MATLAB/Simulink software. It receives the formatted measurement message and uses this information to estimate the distance and relative angle of the car with respect to the wall. These estimates are then used as signals within the control algorithms, which in turn adjust the voltage to the motors to steer the car in the correct direction.

4- Results

During the project development, the progress was monitored through various tests:

Initially, before proceeding with the wall control design, it was verified that the information provided by the LiDAR sensor was coherent. To do this, a map of the test circuit was generated to check if the contour was recognizable. Additionally, the two crucial signals to the control algorithms, the distance and the car's relative angle to the wall, were monitored.

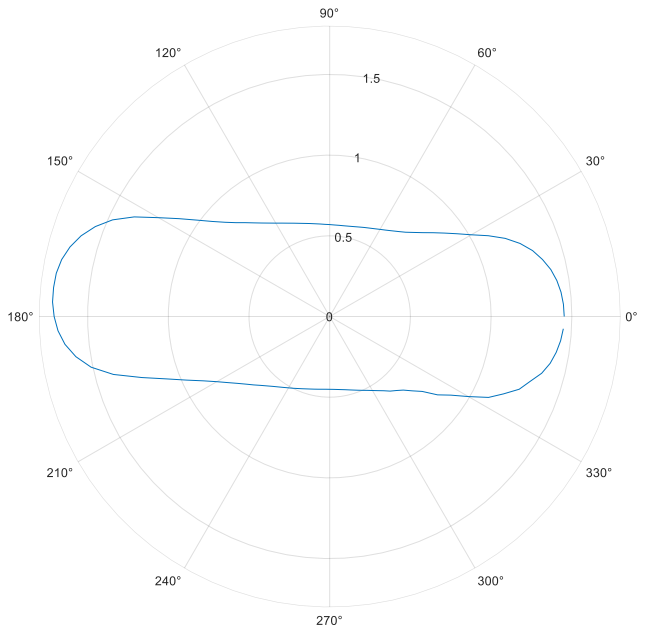


Figure 2. Map of the tests circuit

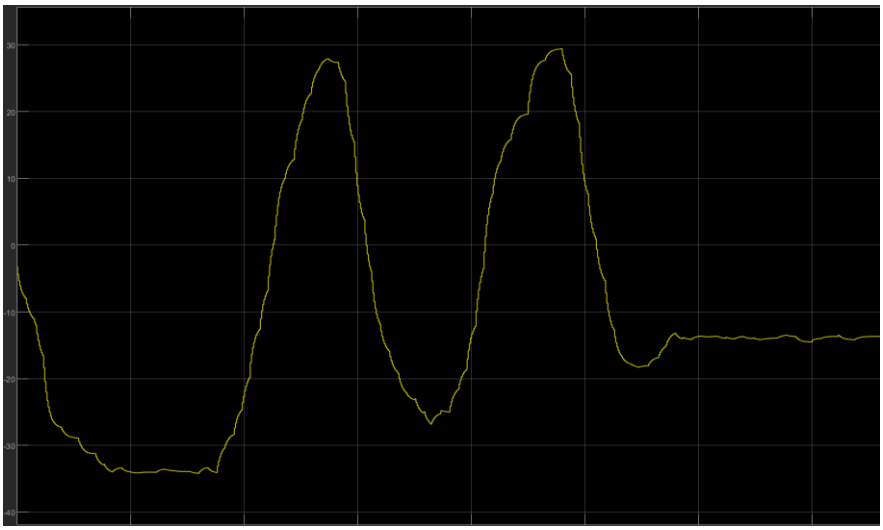


Figure 3. Test monitoring the distance signal.

Secondly, before concluding the project, it was ensured that the control system responded correctly. The response to a variable reference was captured to observe how the distance-to-wall signal adapted to changes in the reference. The result was positive. As we observe in the obtained plots, both the forward velocity control and the wall distance control work as expected, following the specified reference value for each magnitude.

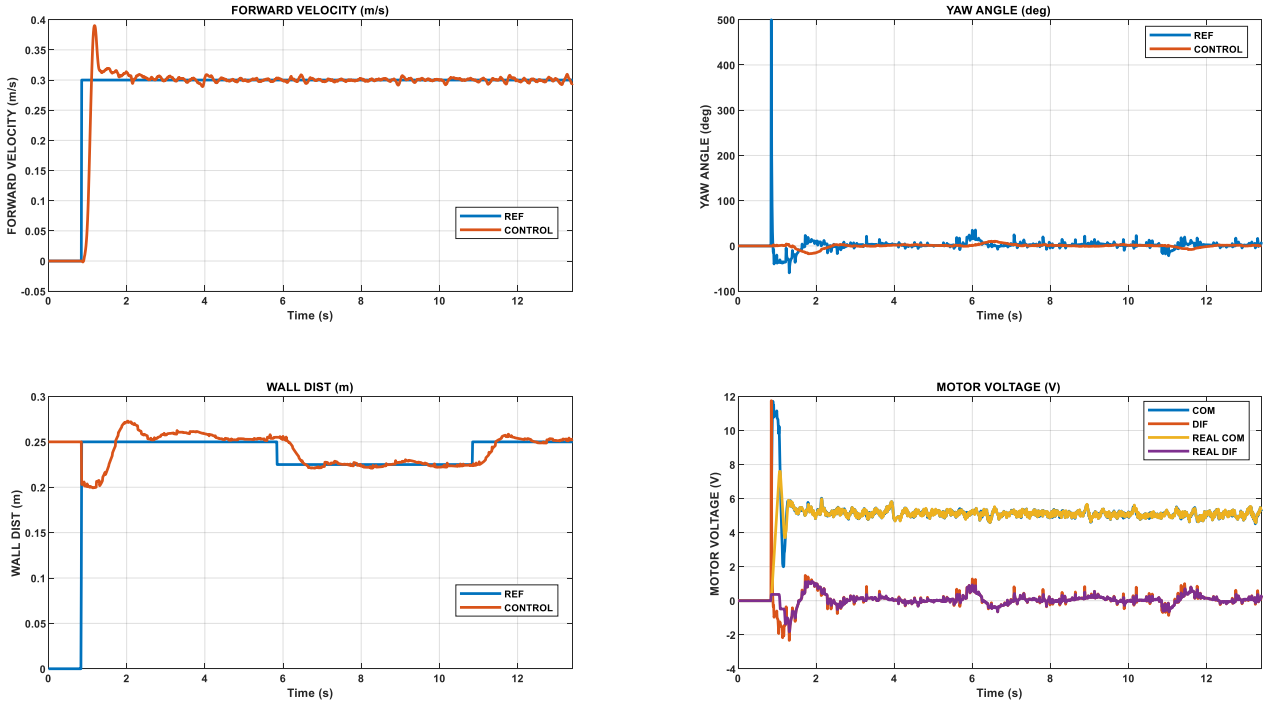


Figure 4. Test with a variable reference to check the transient response.

5- Conclusion

Based on the results obtained, we can conclude that the objectives set at the beginning of the project have been successfully achieved.

The developed software successfully obtains coherent measurements from the sensor, enabling reliable information about the environment's morphology to be obtained.

Additionally, the wall tracking control is also successful. The tracking tests on the test circuit have been successful, and the vehicle manages to regain its orientation on curved sections to avoid collision with the wall.

For future projects, it is proposed to explore new techniques that are gaining popularity in autonomous navigation projects, such as SLAM (Simultaneous Location and Mapping). This would allow the robot to understand the morphology of its environment and be able to plot trajectories considering the entire stretch of wall it must follow. Another alternative would be to implement new sensors in the vehicle, such as video cameras, to obtain more information about the environment.

6- References

[1] Madrigal Moreno, S. A., & Muñoz Ceballos, N. D. (2019). *Vehículos de guiado autónomo (AGV) en aplicaciones industriales: una revisión.* Revista Politécnica, 15(28),117-137.[fecha de Consulta 25 de Febrero de 2024]. ISSN: 1900-2351. Recuperado de: https://www.redalyc.org/articulo.oa?id=607866567012
Último acceso: 04/2024

Agradecimientos

Estoy especialmente agradecido a Diego Cubillo Llanes y a Juan Luis Zamora Macho, directores del proyecto, por estar siempre atentos y dispuestos a resolver cualquier duda, explicar cualquier concepto y hacer posible este trabajo.

También quería dar las gracias a mis padres y a mi hermana, mis mayores referentes, por el apoyo incondicional. Por último, a mis amigos, tanto los que me han acompañado en mi paso por ICAI como los que venían de antes, gracias por estar ahí siempre.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

El auge de nuevas tecnologías como la Inteligencia Artificial (IA) o el Internet de las cosas (IoT) han impulsado en gran medida el desarrollo de máquinas cada vez más inteligentes y dotadas de mayor percepción de su entorno. Además, la necesidad de una industria cada vez más eficiente, hace necesario que el grado de autonomía de estas máquinas sea cada vez mayor, así es como surgen los robots autónomos. Un robot autónomo es capaz de realizar la tarea para la que ha sido diseñado sin la necesidad de intervención humana.

Las grandes ventajas de esta nueva generación de robots han convertido esta tecnología en un factor clave en muchas industrias como son el transporte, la aeronáutica, la fabricación o la agricultura. Las distintas aplicaciones de estas máquinas incluyen fábricas inteligentes, redes de telecomunicación y vehículos autónomos [1].

Ejemplo de la automatización de vehículos son los ya comúnmente utilizados Vehículos de Guiado Autónomo o AGVs por sus siglas en inglés. Estas máquinas, a pesar de no ser capaces de entender la morfología de su entorno, sí pueden seguir trayectorias definidas y evitar colisiones y son un claro ejemplo de solución para procesos productivos que requieren trasladar objetos de un lugar a otro de forma manual o automática [2]. Además, la llegada de la cuarta revolución industrial, lo que conocemos hoy en día como industria 4.0, ha introducido nuevas tecnologías punteras que hacen que el grado de autonomía de estas máquinas se incremente a la vez que cobran mayor consciencia sobre su entorno.

En el contexto de este trabajo nos centraremos en un vehículo terrestre cuyo objetivo es desplazarse en una trayectoria marcada por una pared. Esto quiere decir que el robot, gracias a sus sensores integrados, deberá reconocer la pared y desplazarse a lo largo de la misma, manteniendo una distancia de separación constante marcada como referencia.

Se ha elegido un sensor LiDAR 2D para dotar al robot de información sobre su entorno. Este sensor permitirá obtener un mapa 2D en 360 grados de las proximidades, y entender así el espacio que le rodea.

1.1 Motivación

El escenario actual de las máquinas autónomas hace que proyectos como el propuesto cobren un gran interés.

En entornos industriales podría usarse esta técnica para hacer que diversos robots siguiesen una trayectoria determinada y de interés, marcada por un elemento similar a la pared. De esta manera, nos aseguraríamos de que las máquinas navegan con cierta autonomía una vez esa referencia está fijada, rediseñado así el flujo de trabajo de las plantas de fabricación asumiendo tareas relacionadas con el transporte de productos o materias primas y haciendo posible la reducción de costes asociados al personal de las fábricas [3].

El objetivo del proyecto es optimizar el modelo de robot usado en los laboratorios de control de la Universidad Pontificia Comillas, sustituyendo sus sensores de distancia laterales originales por un LiDAR 2D. Esta modificación permite obtener distancias en un plano 360 de los alrededores del robot. Esto amplía el rango de 'visión' del vehículo en comparación con el sensor original, que solo aportaba medidas referentes al lado del robot en el que estaba instalado. El hecho de usar la opción 2D de dicho sensor tiene la ventaja de ser más económico, pero a su vez tiene algunas limitaciones respecto a su versión 3D. Las medidas de distancias quedan restringidas a un plano de altura fija, por tanto, objectos fuera de este rango no serían detectados [4]. Sin embargo, para este proyecto, la pared que marca nuestra referencia de trayectoria sí es detectada a esta altura y, por tanto, un sensor 2D cumple con los requisitos que necesitamos.

1.2 Objetivos del proyecto

El objetivo final del trabajo es el diseño de un sistema de control de distancia que hace uso de un sensor LiDAR 2D como fuente de información acerca del entorno. Como objetivos en la realización del proyecto se marcarán los siguientes:

- Obtención de un sistema funcional en tiempo real que informe de la distancia y ángulo relativo a la pared. El sensor LiDAR es la principal fuente de información de la posición relativa del vehículo a la pared que se pretende seguir. El hecho de

que estás variables representen fielmente la posición real del vehículo sienta las bases para el correcto funcionamiento de los algoritmos de control, ya que son precisamente las variables que se pretenden regular.

- Generación de un software basado en ROS2 capaz de gestionar el funcionamiento del sensor y la comunicación de medidas con el ordenador de control del vehículo. Se desarrollará un software basado en nodos ROS que será capaz de gestionar la puesta en marcha del sensor LiDAR, la recepción de los datos de interés, su posterior formateo y envío al ordenador de control del robot.

- Correcto funcionamiento del robot en el seguimiento autónomo de pared mediante la comprobación por ensayos en el laboratorio. Una vez acabada la implementación del proyecto se comprobará su correcto funcionamiento mediante pruebas, asegurando que el comportamiento es el esperado. El vehículo deberá ser capaz de recorrer un circuito con tramos de pared tanto rectos como curvos. Estas características afectan al control de ángulo y distancia a la pared. Además, los tramos de pared recta presentan secciones inclinadas, que ponen a prueba la capacidad del control de velocidad del vehículo de lograr un avance a la velocidad de referencia especificada independientemente de la inclinación de la pista.

1.3 Metodología

Para el desarrollo experimental del trabajo se seguirá un esquema de trabajo estructurado de la siguiente manera:

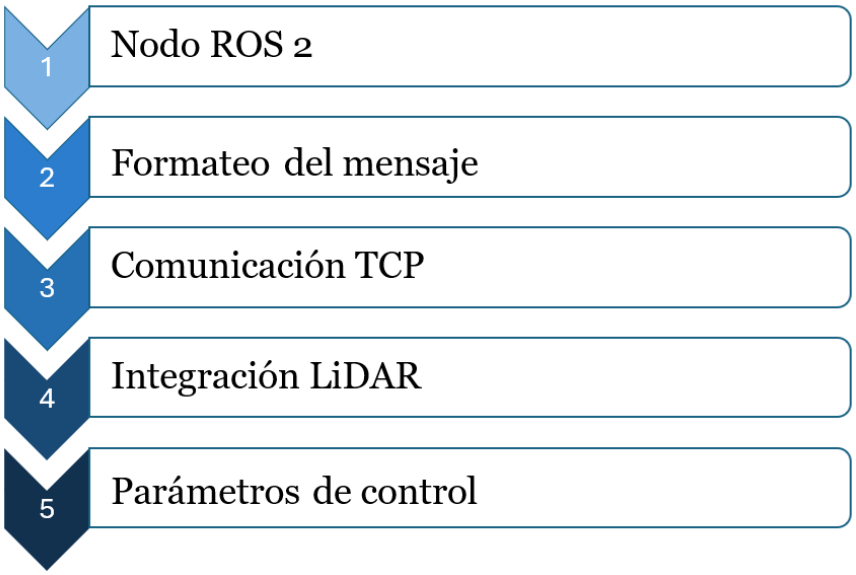


Figura 1. Metodología del proyecto. Fuente: Elaboración propia.

El trabajo comienza por el desarrollo de los programas basados en nodos ROS 2 que gestionan el correcto funcionamiento del sensor y su capacidad para obtener y enviar medidas. Se desarrollarán dos programas independientes, recogidos en el Anexo I y II de esta memoria. En este punto, deberíamos ser capaces de poner en marcha el sensor y de recoger las medidas de ángulo y distancia del LiDAR a los elementos del entorno.

Una vez obtenidas las medidas, el siguiente paso es pasar de medidas en bruto a un formato adecuado y estructurado, para el posterior uso de estas en los algoritmos de control.

Una vez dotados de un formato correcto, el siguiente paso es enviar los datos, por protocolo TCP, al ordenador de control, que los usará en los correspondientes algoritmos para gobernar el vehículo.

Si estas fases se completan de manera correcta, deberíamos poder integrar correctamente el sensor LiDAR con el resto de los elementos del vehículo. Procederemos a la realización de ensayos estáticos de coherencia de las medidas respecto al sistema de referencia del robot. Esto es una fase previa a los ensayos del robot en movimiento. Se pretende alterar la referencia con algún elemento móvil que simule la pared, y monitorizar las señales de distancia y ángulo una vez ya procesadas, para corroborar que la orientación del sensor es la esperada por el control y por tanto la integración ha sido correcta.

En caso de una respuesta correcta de dichas señales a los cambios en el entorno, podremos proceder a los ensayos dinámicos. Se usará un circuito con tramos tanto curvos como rectos para poner a prueba los dos sistemas de control que tiene el vehículo. Debido a los cambios en curvatura del circuito, el sistema deberá adaptar su velocidad y rumbo para no colisionar con la pared, haciendo uso de su control de distancia. Además, los tramos de pared recta son tramos inclinados. Esto hace que el robot tenga que hacer uso de su control de velocidad para mantener su avance constante a pesar de la inclinación de la pista. En estos ensayos finales se pretende modificar los parámetros del propio sistema de control del vehículo para obtener una respuesta robusta y rápida ante las características del circuito.

Una vez superada esta última fase, deberíamos tener un sistema completamente funcional, capaz de recorrer el circuito adaptando su velocidad y rumbo a medida que detecta como cambia su distancia a su referencia, la pared.

Capítulo 2. Estado del arte

La Revolución Industrial trajo consigo un gran avance que ha impactado la manera en que funcionamos, ayudándonos a optimizar el tiempo y los recursos: la automatización.

El auge de las industrias manufactureras hizo que se optase por automatizar muchos trabajos manuales. De esto son ejemplo el telar mecánico de Edmund Cartwright y la máquina de vapor de James Watt. Posteriormente, "la línea de ensamblaje por parte de Henry Ford en 1913" sentó las bases de la automatización industrial [5].

Históricamente muchas de las labores a las que se han destinado los robots han sido labores de fabricación y ensamblaje. Sin embargo, el auge en la escena tecnológica ha hecho que otras labores claves en la operación de muchas industrias, sobre todo en cuestiones logísticas, sean un nuevo objetivo que una nueva generación de robots debe cubrir [6].

Así surgen los vehículos autónomos. Fuera del sector industrial, hay más ejemplos de aplicación para estas máquinas como pueden ser robots de exploración en áreas que pondrían en peligro la integridad de un explorador humano. Disciplinas como la vulcanología o la exploración espacial se ven en gran medida beneficiadas de esta nueva aplicación tecnológica [6].

Otra industria interesada en este desarrollo tecnológico es la industria del transporte. Varias empresas han comenzado a implantar sistemas de navegación autónoma en vehículos terrestres, marítimos y aéreos. Algunas de las empresas más relevantes en la industria de los viajes compartidos, como lo son Uber o Lyft, ya están intentando implementar esta nueva tecnología en su modelo de negocio. Con esta iniciativa se pretende optimizar la operación de estos vehículos, reduciendo el coste por espacio no utilizado en cada viaje y mejorando la eficiencia en las rutas elegidas, maximizando así el beneficio de estas compañías [7].

En el caso de la movilidad terrestre, la navegación autónoma unida a técnicas de inteligencia artificial puede transformar por completo este sector hacia un modelo más sostenible y responsable con el medio ambiente. En cuanto a los vehículos autónomos

podemos encontrar una clara escala que los diferencia según el nivel de autonomía que consiguen en su operación [8]:

0	No hay Automatización
	No se automatizan las tareas del conductor. El vehículo puede incluir sensores o radares que notifican al conductor de eventos y objetos cercanos.
1	Conducción asistida
	Incluyen sistemas que controlan la dirección, la velocidad y el frenado. Algunos ejemplos son el control de distancia anticolisión o el control de carril.
2	Automatización parcial de la conducción.
	El vehículo integra sistemas que automatizan en el desplazamiento tanto longitudinal como lateral. Sin embargo, se requiere que el conductor este atento en todo momento y a los mandos del control del vehículo, aunque no tenga que preocuparse por tareas de movimiento.
3	Automatización condicionada de la conducción.
	El vehículo integra sistemas que automatizan el desplazamiento tanto longitudinal como lateral de forma simultánea. El conductor no requiere tomar los mandos como norma general pero sí que es necesario estar preparado para tener que hacerlo en caso de que el vehículo así lo indique.
4	Automatización elevada de la conducción.
	En este nivel desaparece la figura del conductor. El sistema tiene un protocolo de respaldo en caso de que se produzca algún imprevisto. Sin embargo, sigue habiendo algunas limitaciones y se pueden encontrar situaciones en las que continuar con la conducción no sea posible.
5	Automatización completa de la conducción.
	En este nivel ya no hay situaciones que puedan limitar el funcionamiento normal del vehículo. Este por tanto podría seguir conduciendo en toda situación.

Tabla 1. Niveles de autonomía de vehículos. Fuente: Elaboración propia.

2.1 Contexto histórico del coche autónomo

La historia del coche autónomo tiene su origen en el año 1925, cuando el ingeniero eléctrico Francis Houdina, muestra el primer concepto de coche autónomo circulando por las calles de Manhattan. Este prototipo era controlado por radio, sin la necesidad de un conductor físico en el interior del vehículo [9].

Fue en 1986 cuando el ingeniero Ernst Dickmanns equipó una furgoneta Mercedes Benz con todo tipo de cámaras y sensores. Un año más tarde, tras pruebas por un recinto especializado de la Universidad de Múnich, esta furgoneta alcanzó sin incidentes los 90 km/h por un tramo de Autobahn [10].

A las contribuciones de Dickmanns para el desarrollo de esta nueva tecnología se le suman las siguientes. En 1994, dos vehículos gemelos condujeron más de mil kilómetros en una autopista de París con 3 carriles, en días habituales y con tráficos intensos a velocidades de 130 km/h. El año siguiente, en 1995, Dickmanns modificó un Mercedes-Benz Clase S que consiguió completar un viaje de ida y vuelta entre las ciudades de Múnich y Copenhague, haciendo uso de técnicas de visión por ordenador. Se necesitaron pequeñas intervenciones humanas, pero el porcentaje de conducción autónoma en dicho viaje fue del 95% [10].

Ya en la década de los 90 y continuadamente en los 2000, empiezan a surgir sistemas de estacionamiento automático, sentando las bases de los vehículos autónomos que conocemos hoy en día [9].

A partir de 2009, Google comienza su proyecto de Google Car, queriendo impulsar el desarrollo y fiabilidad de los coches autónomos. Así, en diciembre de 2016, el proyecto toma el nombre de Waymo y se postula como una compañía independiente a Google dentro del grupo Alphabet.

2.2 El coche autónomo en la actualidad

La realidad es que actualmente la conducción autónoma no está ampliamente normalizada. Aunque ya hay ejemplos de ciudades que permiten la conducción de este tipo de vehículos e incluso empresas que ofrecen servicios de viajes en vehículos

autónomos, está tecnología ha tenido que luchar con diversos factores en los últimos años para hacerse un hueco en el sector del transporte. A pesar de que los avances técnicos evolucionan a un ritmo desmesurado, son factores políticos, jurídicos y regulatorios los principales responsables que obstaculizan la implantación final de esta tecnología.

A lo largo de estos años han surgido estudios que evalúan el estado de preparación de distintos países ante la creciente implantación de los vehículos autónomos. Este es el caso del *Índice 2018 de adaptación a vehículos autónomos*, emitido por la firma consultora KPMG. El índice evalúa 4 pilares que son integrales a la capacidad de cada país para adoptar e integrar vehículos autónomos tales como política y legislación; tecnología e innovación; infraestructura; y aceptación del consumidor [11].

Overall rank	Country	Total score	Policy and legislation		Technology & innovation		Infrastructure		Consumer acceptance	
			Rank	Score	Rank	Score	Rank	Score	Rank	Score
1	The Netherlands	27.73	3	7.89	4	5.46	1	7.89	2	6.49
2	Singapore	26.08	1	8.49	8	4.26	2	6.72	1	6.63
3	United States	24.75	10	6.38	1	6.97	7	5.84	4	5.56
4	Sweden	24.73	8	6.83	2	6.44	6	6.04	6	5.41
5	United Kingdom	23.99	4	7.55	5	5.28	10	5.31	3	5.84
6	Germany	22.74	5	7.33	3	6.15	12	5.17	12	4.09
7	Canada	22.61	7	7.12	6	4.97	11	5.22	7	5.30
8	United Arab Emirates	20.89	6	7.26	14	2.71	5	6.12	8	4.79
9	New Zealand	20.75	2	7.92	12	3.26	16	4.14	5	5.43
10	South Korea	20.71	14	5.78	9	4.24	4	6.32	11	4.38
11	Japan	20.28	12	5.93	7	4.79	3	6.55	16	3.01
12	Austria	20.00	9	6.73	11	3.69	8	5.66	13	3.91
13	France	19.44	13	5.92	10	4.03	13	4.94	10	4.55
14	Australia	19.40	11	6.01	13	3.18	9	5.43	9	4.78
15	Spain	14.58	15	4.95	16	2.21	14	4.69	17	2.72
16	China	13.94	16	4.38	15	2.25	15	4.18	15	3.13
17	Brazil	7.17	20	0.93	18	0.86	19	1.89	14	3.49
18	Russia	7.09	17	2.58	20	0.52	20	1.64	18	2.35
19	Mexico	6.51	19	1.16	17	1.01	17	2.34	19	2.00
20	India	6.14	18	1.41	19	0.54	18	2.28	20	1.91

Tabla 2. Índice 2018 de adaptación a vehículos autónomos. Fuente: KPMG

Entre los países a la cabeza en cuanto a preparación ante esta nueva generación de vehículos destacan algunos como Países Bajos, Singapur o Estados Unidos. De la lista, México y Brasil son los únicos representantes de América Latina. Hay una gran representación de países europeos entre los primeros puestos, así como los dos grandes países norteamericanos y las principales potencias tecnológicas de Asia. Estos estudios

confirman la tendencia del sector del transporte hacia un nuevo modelo con los vehículos autónomos como protagonistas en el futuro próximo.

No obstante, el auge de la digitalización también ha traído nuevas vulnerabilidades para los sistemas tecnológicos y los coches autónomos no son una excepción. La preocupación por la ciberseguridad no ha hecho más que crecer en los últimos años. Hoy en día son muchos los sistemas que almacenan información en línea o que están conectados a la red para acceder a datos necesarios para su operación. El Internet de las cosas o IoT, es una gran tecnología que facilita muchas tareas de nuestro día a día brindando a los sistemas una avanzada capacidad de comunicación que propone como solución la conexión de estos a Internet. Sin embargo, cualquier dispositivo conectado a la red es vulnerable de sufrir un ataque de seguridad. En el caso de los vehículos autónomos, muchos necesitan esta conexión para recibir información crucial para la conducción, pero esto es una puerta de entrada directa para ciberdelincuentes que podría manipular el funcionamiento de los sistemas digitales integrados en los vehículos. Es por ello que la preocupación por crear sistemas que garanticen la seguridad de los tripulantes a bordo de estos vehículos ha aumentado, junto a los esfuerzos para solucionar este problema. Algunos fabricantes, como Chrysler y Tesla, ya ofrecen jugosas recompensas a hackers que les notifiquen vulnerabilidades [12].

El coche autónomo desarrollado por Google, Waymo, no necesita la red de redes para circular con toda seguridad por las carreteras. Si el coche autónomo quiere ser una realidad tendrá que garantizar la máxima seguridad a los futuros compradores y esta podría ser una buena solución al problema [12].

2.3 Sensores populares en vehículos autónomos

Para ser capaces de interactuar con su entorno, los vehículos autónomos necesitan recibir información acerca del mundo físico que les rodea. Por ello, es necesario equiparlos con diferentes sensores que les dotarán de conocimiento sobre su situación.

2.3.1 Cámaras

La captación de imágenes del entorno es crucial para la operatividad de un vehículo autónomo. De hecho, en mayo de 2018 el Departamento de Seguridad del Transporte, la

Administración Nacional de Seguridad en las Carreteras (NHTSA) de Estados Unidos hizo obligatorio por ley, que todos los vehículos fabricados a partir de dicha fecha incluyesen cámaras de visión trasera. Con esta iniciativa se pretende disminuir el número de accidentes por atropellos en la conducción marcha atrás [13].

Este tipo de sensores abre la posibilidad de realizar el reconocimiento de objetos del entorno del vehículo. Los datos recopilados por cámaras pueden fusionarse con algoritmos de inteligencia artificial para realizar una clasificación de objetos y así poder distinguir desde líneas de carril, para la asistencia de dirección de carril, hasta peatones cruzando delante del vehículo en entornos urbanos o distintos objetos relevantes para la circulación como semáforos o señales de tráfico [14].

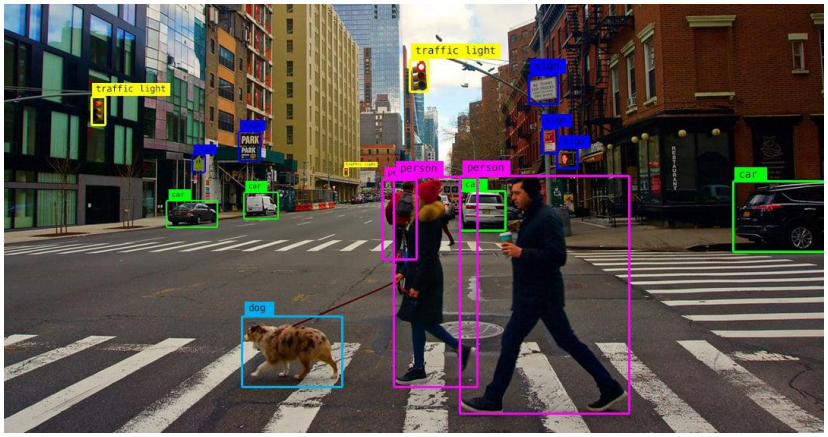


Figura 2. Ejemplo de clasificación de objetos en un entorno urbano con técnicas de clasificación de imágenes. Fuente: Algotive.

Sin embargo, las cámaras tienen problemas determinando la distancia a los objetos que identifican. Saben que están presentes, pero no cuánto espacio los separa de ellos con exactitud.

Por otro lado, es cierto que las cámaras de luz visible presentan limitaciones similares a las del ojo humano. En situaciones meteorológicas adversas o de poca visibilidad, estos sistemas pierden eficacia y fiabilidad. Además de esta tipología de cámara, también existen otras variantes como las cámaras infrarrojas, que son particularmente útiles en la visión nocturna [14].

2.3.2 Radar

Este tipo de sensor tiene una historia que se remonta décadas atrás, cuando su uso se popularizó en el ámbito militar, primero en aeronaves y en el transporte marítimo. Cubren la limitación de las cámaras de determinar la distancia a objetos de interés.

Su funcionamiento se basa en ondas de radio, que una vez emitidas por el sensor, rebotan en los objetos próximos y viajan de nuevo al sensor. Este, tomando el tiempo entre emisión y recepción, puede calcular la distancia a los objetos del entorno.

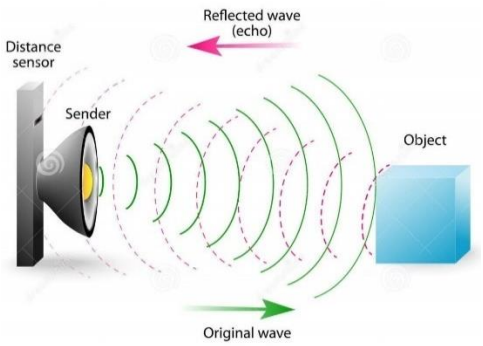


Figura 3. Diagrama de funcionamiento de un sensor radar. Fuente: WordPress

La ecuación empleada para el cálculo de distancia estimada usando un sensor radar es la siguiente:

$$d = \frac{c * t}{2}$$

Ecuación 1.Cálculo de distancia a objeto usando sensor radar.

Donde d es la distancia al objeto, t es el tiempo entre emisión y recepción del pulso original y c es la velocidad de la luz en el vacío, con un valor de 299,792,458 m/s.

En el caso de los vehículos, estos sensores suelen colocarse en la parte delantera y así poder identificar no solo la distancia sino también la velocidad de distintos objetos en el campo de visión frontal del automóvil, aunque es cierto que hay modelos donde se encuentran en partes posteriores o laterales. Los radares funcionan correctamente en situaciones de baja visibilidad, a diferencia de las cámaras. Sin embargo, a pesar de ser eficientes en cálculos de distancia, tienen dificultades reconociendo la forma de los objetos, por tanto, es difícil hacer uso de ellos para la clasificación tanto de vehículos como de personas o señales [15].

2.3.3 LiDAR

El funcionamiento del sensor LiDAR (light detection and ranging) es análogo al del radar. La diferencia es que la onda emitida en este sensor es una señal laser en vez de una onda de radio. Esto hace que la respuesta de esta tipología de sensor sea mucho más compleja y detallada. Con todos los puntos recogidos puede generarse una nube de puntos, que representa fielmente el 'esqueleto' o silueta del entorno que rodea al sensor. Estos datos tienen una gran calidad de precisión, profundidad y forma de los objetos [15].

Existen distintas versiones de este tipo de sensores ofertadas por distintos fabricantes, pero las dos más relevantes son la versión 2D y 3D. La diferencia entre estos dos tipos de LiDAR es que la versión 2D escanea un solo plano, dando lugar a mapas de puntos en dos dimensiones, mientras que la versión 3D escanea varios planos, dando lugar a volúmenes. Según la aplicación para la que se destine el sensor, la versión 2D puede no ofrecer la suficiente información que necesitamos. En muchos vehículos hay sistemas de LiDAR 2D instalados destinados a funciones de mapeo del entorno o de seguridad, sin embargo, para aplicaciones más avanzadas como la detección de personas o volúmenes con una forma concreta, la versión bidimensional ofrece un funcionamiento limitado, requiriéndose la versión 3D del sensor [16].

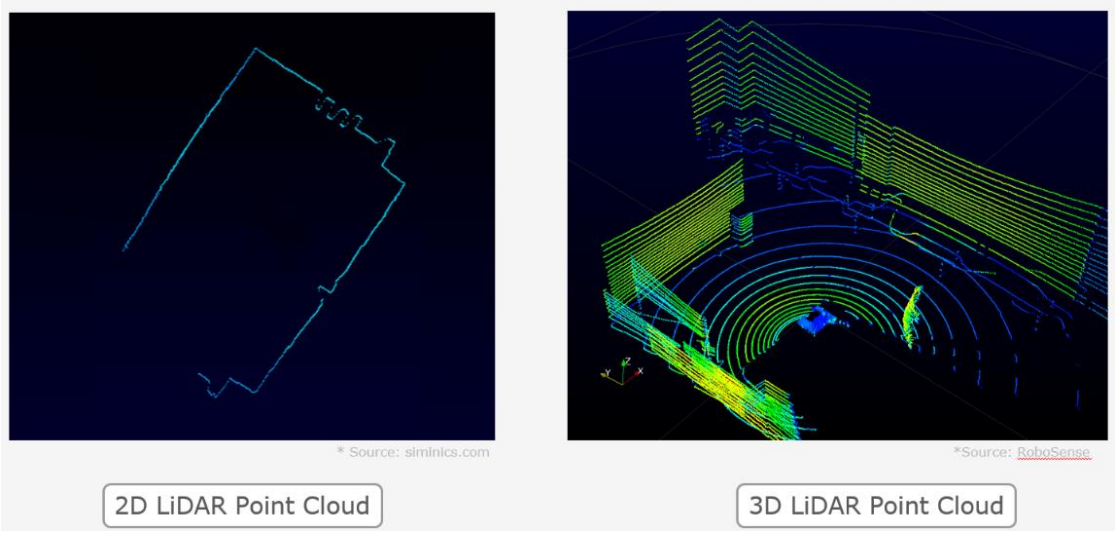


Figura 4. Comparación entre nubes de puntos de un LiDAR 2D y 3D. Fuente: Robosense.

2.3.4 GNSS

Se denomina así a los sistemas globales de navegación por satélite. Está tecnología está basada en una extensa red de satélites que emiten señales de posición y tiempo desde el espacio a los distintos receptores GNSS que solicitan estos datos. Esta tecnología ofrece conexión global y de esto son ejemplo Galileo en Europa, GPS en Estados Unidos, GLONASS en Rusia y BeiDou en China [17].

Usando técnicas de triangulación se puede determinar la posición de un objeto en el espacio tridimensional, pero esta tecnología tiene un inconveniente. Las señales emitidas por las satélites en órbita son débiles y pueden ser modificadas por interferencias en la atmósfera de la Tierra, rebotar en edificios y otros elementos o incluso interferirse fácilmente, lo que se conoce como suplantación. Es por ello por lo que generalmente se complementa el uso de estos sistemas con otros como las IMU. Estos sistemas más avanzados pueden conseguir una precisión en el resultado de posición de hasta 1cm [14].

2.3.5 IMU

Las siglas IMU (del inglés unidad de medición inercial) hacen referencia a un tipo de sensores cuyo objetivo es ayudar en el posicionamiento del objeto en el que se implanten. Están compuestos de acelerómetros y giroscopios.

En el caso de los vehículos autónomos, la función de estos sensores es detectar los movimientos a base de determinar la orientación y la velocidad con la que se mueve el vehículo. El inconveniente de usar una IMU aislada es que a medida que el vehículo se desplaza se van acumulando pequeños errores en la medición, que, a pesar de ser pequeños, acumulándose en el tiempo pueden crear una distorsión no despreciable en el resultado de medida. Es por ello por lo que se suelen combinar con otros sistemas como la tecnología GNSS. Así, actuando de forma combinada, se potencia la fiabilidad de las medidas cubriendo una tecnología las limitaciones de la otra [15].

2.3.6 Sensores ultrasónicos

Con un funcionamiento muy similar al radar, tenemos los sensores de ultrasonidos. En vez de ondas de radio estos sensores emiten ondas sonoras, que una vez rebotan en objetos

del entorno, vuelven al sensor, pudiendo así calcular la distancia del sensor al objeto, al igual que pasaba en la tecnología radar. Estos sensores se inspiraron en la ecolocalización presente en ejemplares de la naturaleza como los murciélagos, que también hacen uso de ondas sonoras para orientarse.

Este tipo de sensores es un clásico implantado en muchos modelos de automóviles desde los 90, a raíz del desarrollo de los sistemas de estacionamiento asistido. La razón, es que estos sensores son más baratos y funcionan con eficacia máxima a bajas velocidades y con objetos relativamente cercanos. Por ello las técnicas de estacionamiento, que se efectúan a velocidades razonablemente lentas, eran óptimas para el uso de este tipo de sensores [18].

2.4 Algoritmos usados en navegación autónoma

La información recogida por los diversos sensores integrados en los vehículos autónomos necesita de cierto procesado para ser útil. Es por ello por lo que se han desarrollado multitud de algoritmos que hacen uso de estos datos para tomar decisiones sobre la conducción del vehículo.

2.4.1 Técnicas de Inteligencia Artificial

El auge de las tecnologías de inteligencia artificial en los últimos años también ha impactado el desarrollo de los coches autónomos. Los diferentes algoritmos de inteligencia artificial se clasifican en dos grandes grupos: técnicas de aprendizaje supervisado y no supervisado.

Las técnicas de aprendizaje supervisado usan conjuntos de datos con unas etiquetas conocidas. Se pretende que el modelo use los datos para predecir un resultado preciso o clasificar nuevos datos en categorías conocidas [19].

Por otro lado, las técnicas de aprendizaje no supervisado no tienen etiquetas fijadas. Estos modelos son capaces de descubrir patrones entre los datos sin necesidad de una supervisión humana [19].

Algunas de las tareas que abordan distintos algoritmos de inteligencia artificial en materia de vehículos autónomos son las siguientes [20]:

	Aprendizaje Supervisado	Aprendizaje No Supervisado	
1	Reconocimiento de Objetos	Detección de Anomalías	1
	Ser capaz de clasificar distintos objetos haciendo uso de datos de video provenientes de cámaras integradas en el vehículo. Distinguir otros vehículos, personas, semáforos o señales de tráfico.	Se han desarrollado sistemas que, mediante técnicas de aprendizaje no supervisado, han conseguido una gran eficiencia para detectar eventos inesperados y responder de manera rápida. Ejemplos son un peatón cruzando de manera inesperada o un giro brusco de otro conductor.	
2	Modelado de Posibles Escenarios	Clustering	2
	Ser capaz de generar modelos complejos para prevenir posibles eventos adversos que afecten a la circulación. Este cálculo de probabilidades se basaría en la información recibida por los sensores en tiempo real, por ejemplo, imágenes de mucho tráfico, información meteorológica, etc.	Estas técnicas hacen posible que el vehículo asocie condiciones de conducción similares a una situación común. Esto ayuda a los sistemas de conducción a saber simplificar situaciones complejas al tener una guía de conducta ante ciertas situaciones que reconoce.	
3	Predicción de Comportamiento	Extracción de Características	3
	Usando algoritmos avanzados de entrenamiento y aprendizaje, estos sistemas son capaces de predecir el comportamiento de los usuarios del entorno. Esto incluye tanto a peatones, en entornos urbanos, como a otros conductores en la carretera.	Estos algoritmos son una parte clave del proceso de selección de los datos más significativos recogidos por los sensores. Son capaces de identificar cuáles de todas las medidas obtenidas tienen mayor importancia y centrar los esfuerzos en el procesado de estas, mejorando la percepción del entorno	

Tabla 3. Técnicas de IA usadas en la conducción de vehículos autónomos. Fuente: Elaboración propia.

Estos ejemplos de aplicación dejan ver que implementar técnicas de inteligencia artificial hace mucho más eficiente la conducción autónoma. Al cobrar los vehículos una mayor capacidad de percepción y entendimiento de las situaciones a las que se enfrentan, las decisiones son más acertadas debido al fundamento detrás de estas.

2.4.2 Filtro de Kalman

Debido a que el correcto funcionamiento de los vehículos de conducción autónoma depende de la información recogida por sus sensores integrados, es necesario que esta sea lo más precisa posible. Un inconveniente muy común a la hora de implantar sensores es el ruido. En la mayoría de los casos, es muy complicado obtener medidas de un sensor sin obtener pequeñas perturbaciones debido al ruido. Si estas perturbaciones se acumulan por el uso de medidas distintas provenientes de diferentes sensores, la incertidumbre total puede ser suficientemente grande como para afectar a la fiabilidad de la información que recibe el sistema y, por tanto, hacer que el vehículo no perciba de forma precisa el entorno que le rodea. Es por esta razón que los sistemas a bordo hacen uso de los filtros de Kalman, ya que permiten obtener medidas más precisas que las señales en crudo generadas por los sensores [21].

El filtro de Kalman es un algoritmo que permite estimar el estado de un sistema dinámico haciendo uso de un conjunto de medidas afectadas por ruido. Puede usar información proveniente de varios sensores para combinarla y ayudarse en su estimación [22]. Su funcionamiento está basado en métodos de teoría de probabilidad y análisis de señales.

En el caso de la navegación autónoma de vehículos, estos algoritmos son de gran utilidad para estimar variables como la posición, velocidad y orientación del vehículo en cada instante de tiempo. El modelo de transición de estado puede ser un modelo cinemático simple del vehículo, mientras que el modelo de observación puede incluir mediciones de GPS, IMU, LIDAR, cámaras, entre otros. [23].

Existen variantes del filtro para naturalezas distintas de casos. La mayor segmentación de tipos tiene que ver con el caso de sistemas no lineales. En los casos de uso en el campo de la navegación autónoma, muchas de las medidas recogidas por los sensores no son

lineales. Por ello, el algoritmo debe modificarse para poder adaptar su funcionamiento a estos tipos de sistemas. El resultado de esta modificación se conoce como Filtro Extendido de Kalman o Filtro de Kalman- Schmidt.

2.4.3 Técnicas SLAM

Las siglas SLAM, del inglés simultaneous localization and mapping, se refieren a un método que permite generar un mapa del entorno y localizar a un vehículo autónomo en dicho mapa de manera simultánea [24].

Los vehículos autónomos son expuestos a situaciones de operación en entornos cambiantes. Estas técnicas pretenden obtener información de las características del entorno en un primer escaneado para así, una vez obtenida toda esa información, poder tomar decisiones en base a la morfología y posibles obstáculos que puedan presentarse. De esta manera es posible optimizar las rutas, lo que se conoce como Path Planning, y así la eficiencia del desplazamiento del vehículo. Además, debido a que se conoce el mapa del entorno por el que el vehículo va a desplazarse, es posible localizarlo a la vez que se desplaza [25].

Es importante decir que los sistemas SLAM no son una tecnología específica. Existen multitud de algoritmos para abordar el problema de la localización y mapeado simultaneo de robots, cada uno con un enfoque diferente. El término SLAM agrupa todas estas técnicas bajo un concepto general. Aun así, podría decirse que un sistema SLAM está formado por dos componentes o etapas principales, independientemente de los algoritmos específicos utilizados en cada una de ellas [26].

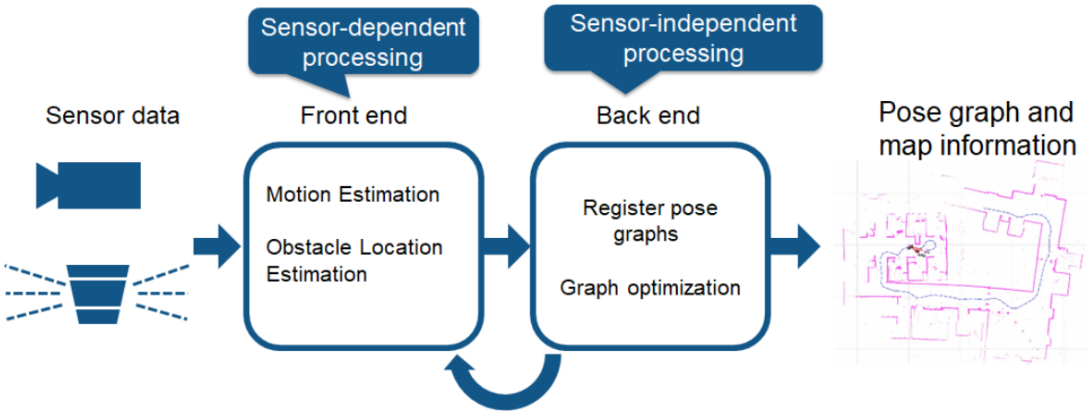


Figura 5. Diagrama de funcionamiento de un sistema SLAM. Fuente: Mathworks.

- **Percepción del entorno o front end:**

  En esta primera etapa depende directamente de los sensores equipados en el vehículo. Recoge toda información generada por estos que sea relevante para la estimación de variables referentes tanto a la localización, como pueden serlo la cuenta de revoluciones de las ruedas, como para el mapeado, como imágenes de cámaras o datos de sensores LiDAR [24].

- **Extracción de datos o back end:**

  La información recogida por los sensores es analizada e interpretada. El objetivo final de la gran variedad de algoritmos que pueden implementarse en esta etapa es usar la información extraída por los sensores para identificar la morfología del entorno por el que se desplaza el robot [26].

Para poder conseguir un sistema SLAM completamente funcional es necesario que la comunicación entre estos dos componentes, y también entre los diferentes sistemas hardware y software integrados en el vehículo, sea fluida y responda de forma correcta. Todos estos sistemas están compartiendo información de unos a otros de forma constante y, además, necesitan esos datos para poder efectuar correctamente su tarea. Por tanto, la integración de los diferentes elementos en el sistema conjunto es tan crucial para el funcionamiento de un sistema SLAM como el funcionamiento individual de cada una de las etapas.

Las aplicaciones SLAM son muy variadas y no se reducen sólo al campo de los vehículos autónomos más conocidos que son las aplicaciones dedicadas al sector del transporte. Ya hay ejemplos de aplicación de esta tecnología en robots del sector de la domótica, que asisten en tareas del hogar. El hecho de tener un mapa claro de la superficie que deben limpiar les permite planear rutas óptimas para cubrir toda la superficie sin pasar por una misma zona dos veces, ahorrando así tiempo.

En el sector del entretenimiento, ya son varios los espectáculos nocturnos que se han llevado a cabo con drones, siendo capaces de crear figuras de luces en tres dimensiones. Esto es posible ya que las técnicas SLAM, al dar información de la localización de cada robot en el mapa generado, permiten la coordinación entre varios de ellos, permitiendo a cada uno trazar rutas de desplazamiento específicas para formar la figura deseada y a la vez, evitando la colisión con cualquier otro robot que se mueve en su entorno [26].

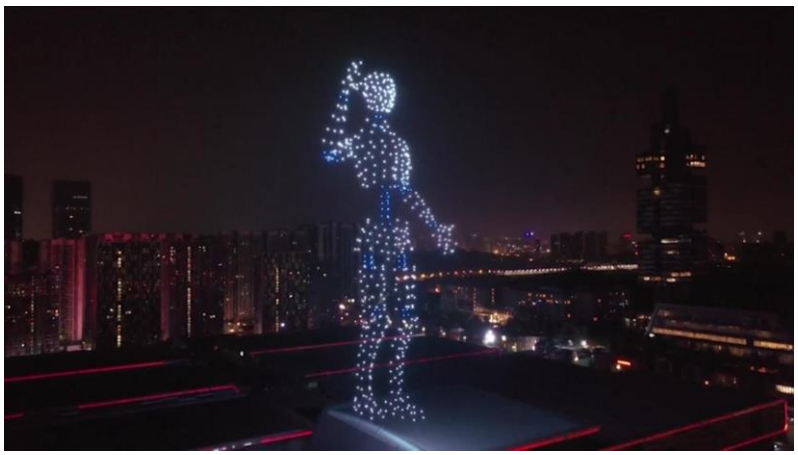


Figura 6. Espectáculo de drones usando coordinación con métodos SLAM.

Por último, los sistemas SLAM también tienen aplicación en el campo de la medicina. Gracias a estos dispositivos, es posible realizar cirugías o exámenes médicos de una manera mucho menos invasiva que la cirugía tradicional. Los doctores pueden obtener información del paciente gracias a modelos generados en 3D de la zona examinada sin la necesidad de realizar ningún corte [26].

2.4.4 Técnicas Path Planning

Cuando hablamos de la tecnología SLAM, es importante mencionar el Path Planning. Así se conoce al método de planificación de rutas en la navegación autónoma de robots. En estos casos, se usan los sistemas SLAM con el objetivo de recopilar información del entorno del robot para tomar decisiones basadas en la morfología del espacio por el que el vehículo se está desplazando. De esta manera es posible trazar rutas de desplazamiento óptimas que permitan evitar obstáculos detectados en el mapeado inicial que se ha efectuado, consiguiendo que el vehículo llegue al destino de una manera más eficaz y segura [25].

Existe una gran variedad de algoritmos de Path Planning que se basan en distintas estrategias para resolver este problema. Algunos de los grupos más relevantes son los siguientes:

- **Grid-based search algorithms:**
  Estos algoritmos dividen el mapa generado con las técnicas de SLAM en una cuadrícula y calculan el costo asociado a una posible trayectoria recorriendo cuadricula a cuadricula en la dirección a la posición final. Son especialmente útiles para aplicaciones de movimiento en dos dimensiones. La contrapartida es que requieren de una gran cantidad de memoria debido a las divisiones que generan. En problemas de más dimensiones, la cantidad de memoria necesaria puede llegar a ser excesiva y, por tanto, puede no ser la mejor opción para implementar [27].

- **Sampling-based search algorithms:**
  Estos algoritmos muestrean todas las posibles combinaciones de posición y orientación del robot en el mapa generado. Luego, construyen un árbol de decisión empezando desde el estado (posición y orientación) inicial hasta el estado final, que es el objetivo final del desplazamiento. En cada iteración añaden un estado siguiente al estado de la iteración anterior de forma aleatoria y así son capaces de encontrar una ruta del punto inicial al final. La desventaja de este tipo de algoritmos es que no hay garantía de que la ruta generada sea la óptima, en especial en entornos con pasajes estrechos o características parecidas [27].

- **Trajectory optimization algorithms:**

  Esta clase de algoritmos aborda el problema como si se tratase de un problema de optimización. Toman en cuenta las restricciones del problema y consideran todas las soluciones factibles, muchas de las cuales son de naturaleza no lineal. Mediante técnicas de optimización son capaces de encontrar la solución óptima [27].

Capítulo 3. Hardware

En este capítulo de la memoria se van a explicar los distintos componentes hardware que han sido necesarios en la realización del proyecto, así como la función que cada uno desempeña.

3.1 Vehículo de tracción diferencial

Todo el proyecto está basado en el control de trayectoria del siguiente vehículo:

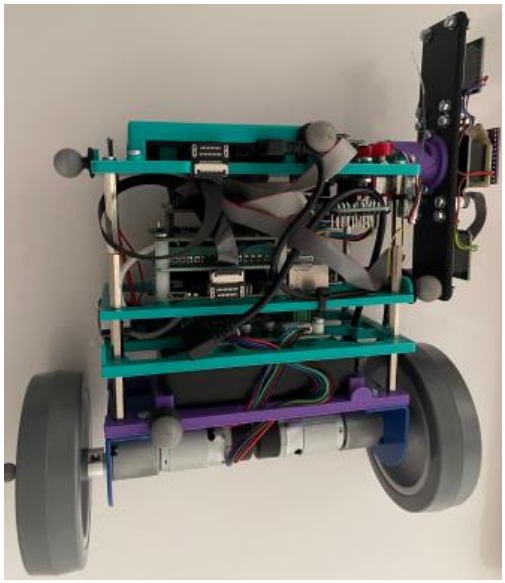


Figura 7. Vehículo de tracción diferencial usado en el proyecto. Fuente: ICAI.

Este, es un vehículo de tracción diferencial equipado con diversos sensores usados comúnmente en gran variedad de vehículos autónomos, que le permite obtener información de su entorno. Los sensores con los que cuenta el vehículo incluyen los siguientes, entre otros: dos motores de corriente continua modelo EMG30, un sensor LiDAR 2D, un Single Board Computer (SBC) Raspberry Pi 3B+, un miniordenador Raspberry Pi 4, encoders magnéticos, una IMU encargada de las medidas de velocidad y aceleración, un módulo de comunicación de radio y una serie de pulsadores y pilotos LED, que actúan como interfaz física con el usuario.

El robot es utilizado en la parte experimental de distintas asignaturas en la Universidad Pontificia Comillas, que se enfocan en el diseño de sistemas de control para poder controlar el desplazamiento de este.

3.2 Motor de corriente continua EMG30

El vehículo tiene equipados dos motores de corriente continua modelo EMG30, uno en cada una de sus ruedas traseras.

Figura 8. Motor de corriente continue EMG30.

Estos motores se usan para controlar tanto la velocidad de avance como el ángulo de giro del vehículo. Están alimentados por una batería de tensión 12V. Si definimos la tensión a la que está sometida cada una de las ruedas de la siguiente manera:

$$u_r: tensión\ de\ la\ rueda\ derecha$$

$$u_l: tensión\ de\ la\ rueda\ izquierda$$

La tensión común, que queda definida por:

$$u_c = \frac{u_l + u_r}{2}$$

Esta tensión se aplica a ambos motores y define la velocidad de avance del vehículo. A mayor tensión el vehículo avanzará linealmente con mayor velocidad.

La velocidad de giro, sin embargo, queda definida por la tensión diferencial entre ambos motores:

$$u_d = \frac{u_l - u_r}{2}$$

La rueda que tenga mayor tensión girará a mayor velocidad, lo que le hace recorrer más distancia que la rueda que gira más despacio y por tanto fuerza al vehículo a girar sobre esta última.

Si ambas ruedas comparten el mismo valor de tensión, y por tanto la tensión diferencial es nula, el vehículo avanzará en trayectoria recta sin girar.

Es así como conseguimos un sistema que nos permite controlar la velocidad de avance y de giro del vehículo mediante el control de dos variables, la tensión de cada una de las ruedas. Son los algoritmos de control que ejecuta el miniordenador Raspberry Pi 3B+ los encargados de la manipulación de estas variables para ajustar la trayectoria del vehículo a la deseada.

3.2 RPLIDAR A2M8 SLAMTEC

Se ha optado por utilizar un LiDAR modelo RPLIDAR A2M8 del fabricante SLAMTEC como sensor para recopilar los datos de posición respecto a la pared.

Figura 9. Sensor RPLIDAR A2M8 de SLAMTEC. Fuente: SLAMTEC.

El sensor funciona de manera común a los distintos sensores con tecnología LiDAR que se han explicado con más detalle en la sección del estado del arte de esta memoria. Este sensor es adecuado para la aplicación del proyecto ya que su especificación de rango de medida máxima y mínima respeta las dimensiones del circuito en el que el vehículo operará. Estas especificaciones se recogen más adelante en la memoria.

El sensor está equipado con un motor que hace girar la carcasa que contiene tanto al emisor como al receptor y permite así un escaneado en 360 grados. Utilizando encoders, se puede determinar el ángulo relativo a una referencia fija del sensor en el que se ha tomado cada una de las medidas de distancia. Esto permite obtener medidas definidas por

un dato de distancia y otro de ángulo. Este formato es muy útil ya que es la principal manera de identificar un punto en un sistema de referencia expresado en coordenadas polares.

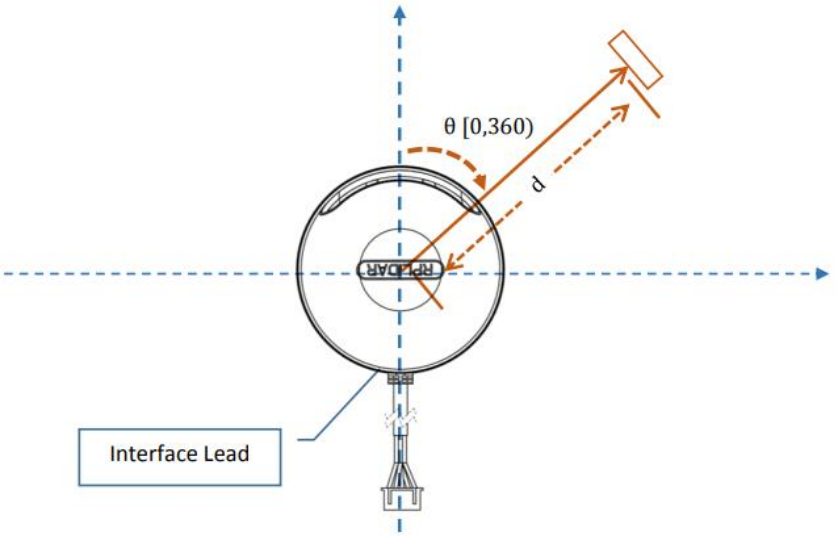


Figura 10. Esquema del sensor mostrando el punto de ángulo de referencia y medidas de ángulo y distancia a un punto. Fuente: SLAMTEC.

Debido a que el sistema de referencia del sensor está fijo, con los diferentes puntos, correspondientes cada uno a una medida, es posible generar mapas que nos dan información de la morfología del entorno.

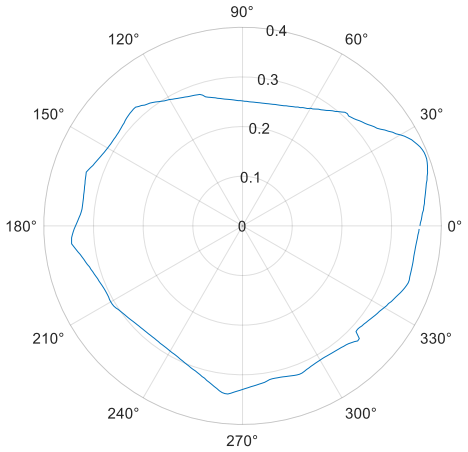


Figura 11. Mapa generado en pruebas que muestra un circuito artificial hecho con papel. Fuente: Elaboración propia.

Las especificaciones proporcionadas por el fabricante SLAMTEC pueden dividirse en dos grupos, características de las medidas y características de potencia del haz laser [28].

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	0.15	-	8	Based om white objects with 70% reflectivity
Angular Range	Degree	-	0-360	-	-
Distance Resolution	mm	-	<0.5 <1% of the distance	-	<1.5 meters All distance range*
Angular Resolution	Degree	0.45	0.9	1.35	10Hz scan rate
Sample Duration	Millisecond(ms)	-	0.25	-	-
Sample Frequency	Hz	2000	4000	4100	
Scan Rate	Hz	5	10	15	The rate is for a round of scan. The typical value is measured when RPLIDAR takes 400 samples per scan

Tabla 4. Especificaciones de calidad de las medidas del sensor RPLIDAR A2M8. Fuente: SLAMTEC.

Es importante recalcar que el sensor tiene unos límites mínimos y máximos de la distancia que es capaz de reconocer, un dato relevante que se discutirá en el apartado de Desarrollo Experimental.

Item	Unit	Min	Typical	Max	Comments
Laser wavelength	Nanometer(nm)	775	785	795	Infrared Light Band
Laser power	Milliwatt (mW)	-	3	5	Peak power
Pulse length	Microsecond(us)	60	87	90	-
Laser Safety Class	-	-	FDA Class I	-	-

Tabla 5. Características de potencia de láser del sensor RPLIDAR A2M8 de SLAMTEC. Fuente: SLAMTEC.

El valor de potencia recogido en las especificaciones corresponde al valor máximo del láser. El fabricante recalca que el valor de operación medio es mucho menor [28].

La conexión del sensor se efectúa mediante un conector XH2.54-5P. Esto permite alimentar los diferentes componentes del sensor, como el motor, y establecer un canal de comunicación por puertos serie que permite controlar la puesta en marcha y el apagado del LiDAR desde un ordenador, además de poder recibir datos de las medidas de escaneo. El diagrama del conector es el siguiente:

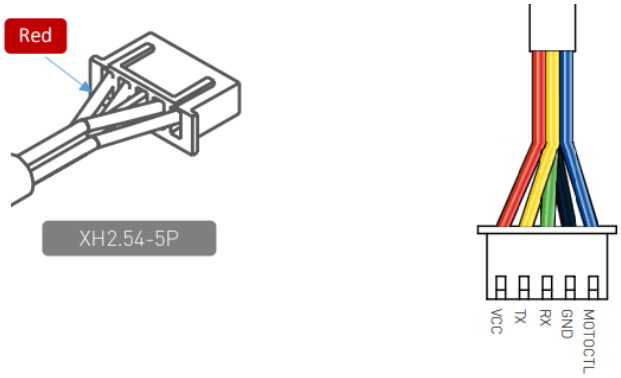


Figura 12. Diagrama de conector XH2.54-5P del sensor RPLIDAR. Fuente: SLAMTEC.

La información detallada del propósito de cada una de las líneas se especifica en la siguiente tabla:

Color	Signal Name	Type	Description	Min	Typical	Max
Red	VCC	Power	Total Power	4.9V	5V	5.5V
Yellow	TX	Output	Serial port output of the scanner core	0V	3.3V	3.5V
Green	RX	Input	Serial port input of the scanner core	0V	3.3V	3.5V
Black	GND	Power	GND	0V	0V	0V
Blue	MOTOCTL	Input	Scan motor /PWM Control Signal (active high, internal pull down)	0V	3.3V	5V

Tabla 6. Función de cada una de las líneas del conector del sensor RPLIDAR. Fuente. SLAMTEC.

3.3 Raspberry Pi 3B+

El vehículo tiene equipado un SBC Raspberry Pi 3B+ a bordo. El objetivo de este componente es claro y crucial. Este dispositivo es el único encargado de gobernar el movimiento del vehículo. Para ello, debe gestionar toda la información proveniente de los distintos sensores. Una vez recibida esta información, mediante el uso de distintos

algoritmos de control para asegurar una trayectoria adecuada, genera ordenes que envía a los distintos actuadores para así poder corregir el rumbo y velocidad del robot.

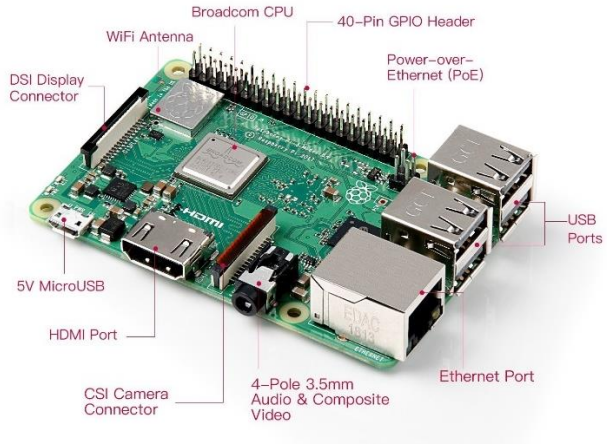


Figura 13. Distintos componentes del miniordenador Raspberry Pi 3B+. Fuente: Euclide.

Podríamos establecer un símil entre los componentes hardware descritos en la memoria como el músculo del robot. Para la coordinación y funcionamiento de todos estos componentes se necesita un 'cerebro' que sea capaz de controlar todo el flujo de información y dar directrices estos sistemas para que puedan trabajar conjuntamente. Esta tarea la realizan los microcontroladores o miniordenadores. Son placas con un procesador y módulos muy diversos que les permiten comunicarse con el mundo exterior mediante GPIOs, puertos de entrada y salida de uso general, que sirven de conexión a una gran variedad de sensores y actuadores. Los miniordenadores programables más conocidos son Nvidia Jetson, Raspberry Pi y Arduino [29].

El proyecto Raspberry Pi tiene su origen en febrero del año 2012, cuando la Raspberry Pi Foundation lanzó los dos modelos originales, versión A y B. El objetivo principal era fomentar la enseñanza y experimentación de las bases de las ciencias de la computación tanto en escuelas como en universidades del Reino Unido. Al poco tiempo de su lanzamiento, el producto ya tenía una gran cantidad de seguidores y desarrolladores que compartían sus proyectos con el mundo. El éxito de este miniordenador fue debido en gran parte a su bajo costo, al alcance de mucha gente para implementar en proyectos desde los niveles más básicos a los más avanzados, gracias a su gran versatilidad. Además, el sistema operativo en el que se basa el ordenador es Linux, que tiene una gran comunidad de desarrolladores por ser un sistema operativo de software libre [30].

El éxito del proyecto hizo que nuevos modelos de Raspberry Pi salieran al mercado adaptándose a las necesidades específicas que demandaban los clientes. El modelo 3B+ nace como la versión mejorada del original modelo 3B al que se le compatibilidad con redes wifi a 5GHz y no solo a 2.4GHz.

Las especificaciones proporcionadas por Raspberry Pi para el modelo 3B+ son las siguientes [31]:

- Procesador Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC de 1.4GHz.
- Memoria SDRAM LPDDR2 de 1GB.
- Conexiones inalámbricas estándar a 2.4GHz y 5GHz LAN, Bluetooth 4.2, BLE.
- Conexión por cable Gigabit Ethernet por puerto USB 2.0 (salida máxima de 300 Mbps).
- Banco de 40 pines GPIO para la conexión con sensores y actuadores.
- Puerto HDMI.
- 4 puertos USB 2.0.
- Puerto de conexión CSI para conectar cámara externa a la Raspberry.
- Salida de audio estéreo a través de conector de 4 polos y puerto de video.
- Puerto Micro SD para almacenamiento y carga de sistema operativo.
- Puertos de alimentación de corriente continua a 5/2.5V.
- Permite alimentación por cable Ethernet (se necesita un módulo PoE adicional).

3.4 Raspberry Pi 4

El vehículo cuenta con otro miniordenador de la familia Raspberry Pi, en este caso es el modelo Raspberry Pi 4. La tarea que desempeña este ordenador es muy distinta a la de del otro ordenador explicado anteriormente. La versión 4 no es responsable de ninguna acción directamente relacionada con el movimiento del vehículo. Su tarea se centra en el control del sensor LiDAR 2D y podría dividirse en tres fases diferenciadas: obtención de los datos, formateo y envío.

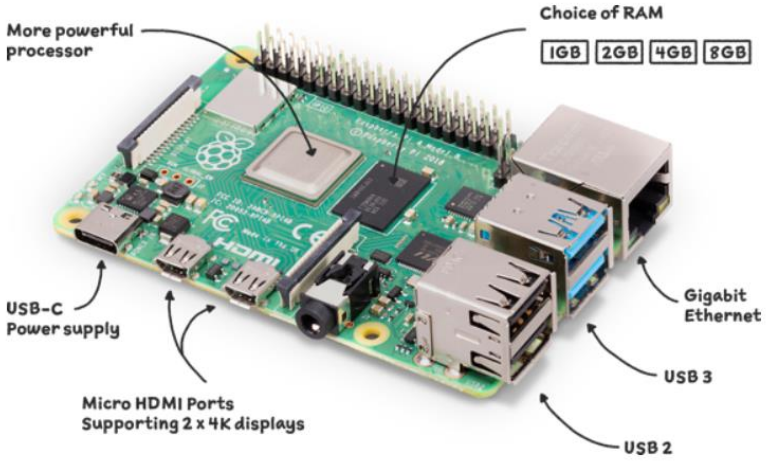


Figura 14. Raspberry Pi 4. Fuente: Raspberry Pi.

Este ordenador es el encargado de ejecutar el programa desarrollado especialmente para este proyecto, que está basado en nodos ROS2 y escrito en Python y que se explicará más adelanta en la memoria. En líneas generales, el objetivo del programa y por tanto de la Raspberry Pi 4 es dar órdenes al sensor LiDAR para controlar el escaneo y obtención de datos de medidas. Además, una vez que el sensor envía los datos en crudo a la Raspberry Pi 4, esta se encarga de darles el formato adecuado para conseguir una estructura que los algoritmos de control puedan entender. Finalmente, una vez que el mensaje de medidas esté listo, este ordenador será el encargado de enviarlo por comunicación TCP al ordenador encargado del control del vehículo, la Raspberry Pi 3B+.

Las especificaciones de este dispositivo proporcionadas por el fabricante son las siguientes [32]:

- Procesador Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC de 1.8GHz.
- Memorias LPDDR4-3200 SDRAM de 1GB, 2GB, 4GB o 8GB (dependiendo del modelo).
- Conexiones inalámbricas estándar a 2.4GHz y 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Puerto Gigabit Ethernet.
- 2 puertos USB 3.0 y 2 puertos USB 2.0.
- Banco de 40 pines GPIO para la conexión con sensores y actuadores.

- 2 puertos micro-HDMI.
- Puerto para pantalla MIPI DSI (Display Serial Interface).
- Puerto para cámara MIPI CSI (Camera Serial Interface).
- Salida de audio estéreo a través de conector de 4 polos y puerto de video.
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode).
- OpenGL ES 3.1, Vulkan 1.0 APIs gráficas.
- Puerto Micro SD para almacenamiento y carga de sistema operativo.
- Conector USB-C de alimentación de 5V en continua (mínimo 3A).
- Conector GPIO de alimentación de 5V en continua (mínimo 3A).
- Permite alimentación por cable Ethernet (se necesita un módulo PoE adicional).
- Temperatura ambiente de operación entre 0-50 C.

Capítulo 4. Software

En este capítulo se van a explicar las diferentes herramientas software utilizadas a lo largo del proyecto y la tarea que cada una de ellas desempeña en el funcionamiento del sistema completo.

4.1 ROS

El nombre ROS (Robot Operating System), se refiere a un conjunto de librerías software y de herramientas *open source* destinadas al desarrollo de proyectos de robótica. Incluye distintos drivers, algoritmos y multitud de herramientas que pueden ayudar a distintos desarrolladores a llevar a cabo proyectos de naturalezas muy diversas. Gracias a su gran comunidad, se ha convertido en una herramienta estandarizada tanto en el prototipado rápido e investigación de productos robóticos como en la producción de sistemas con aplicación en la industria [33].

ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Su base es la arquitectura de grafos del área de las matemáticas. Esto quiere decir que los distintos programas ROS funcionan como nodos, que pueden mandar recibir o multiplexar mensajes provenientes de otros nodos a los que se conectan [34].

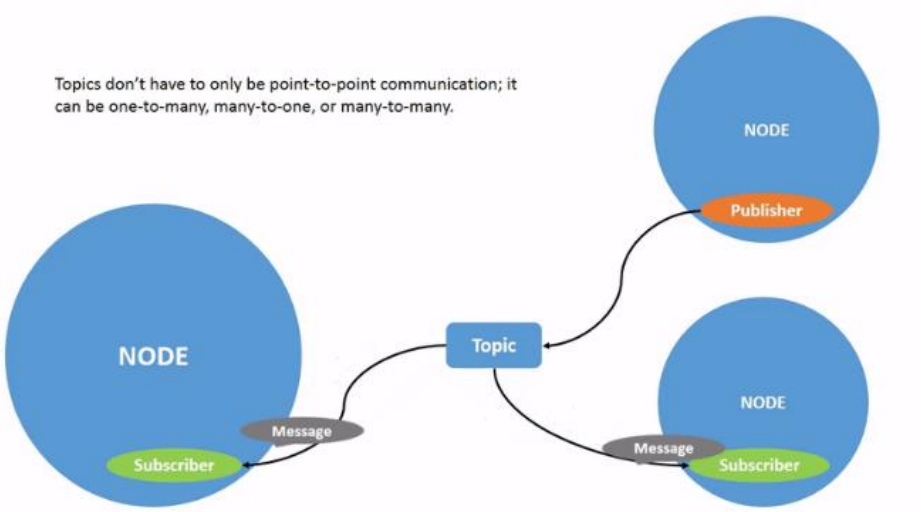


Figura 15. Ejemplo de nodo ROS publisher conectado a dos nodos subscriber. Fuente: ROS.

Cada uno de los nodos es un programa activo con un objetivo específico. Cada nodo puede comunicarse con otros nodos por medio de mensajes. Estos mensajes se envían y reciben por medio de un *topic*, un bus de comunicación. Cuando un nodo está interesado en enviar mensajes a través de un *topic*, actuará como *Publisher* de ese *topic*. En caso de estar interesado en recibir mensajes que se estén publicando en un *topic*, el nodo actuará como *Subscriber* del *topic*. Así se crea una red interconectada de programas que envían y reciben información unos a otros.

A esta estructura básica se le añaden más niveles de complejidad, pero en lo que ocupa a este proyecto, la estructura esencial para el funcionamiento del sistema desarrollado son los nodos, mensajes y *topics*.

El objetivo del proyecto es crear un sistema con tres claros objetivos: obtener las medidas en crudo del sensor LiDAR 2D, modificar su formato a uno entendible por los algoritmos de control y finalmente, enviar el mensaje de medidas con el formato correcto a la Raspberry Pi 3B+. Recordemos que este sistema basado en nodos ROS se estará ejecutando en la Raspberry Pi 4.

Por ello, nuestro proyecto contará con dos nodos y un *topic* por el que se enviarán mensajes de medidas provenientes del LiDAR. El primer nodo será el encargado de poner en marcha el sensor y publicar las medidas en crudo en nuestro canal de comunicación o topic, y el segundo, actuará como subscriptor del *topic* para poder escuchar esa información y será el encargado de formatearla y enviarla a la Raspberry Pi 3B+ por comunicación TCP/IP. Este canal de comunicación es necesario ya que la Raspberry Pi 3B+ no cuenta con ROS, por lo que no es capaz de ejecutar un nodo que reciba los mensajes publicados en el *topic* por el nodo *publisher*.

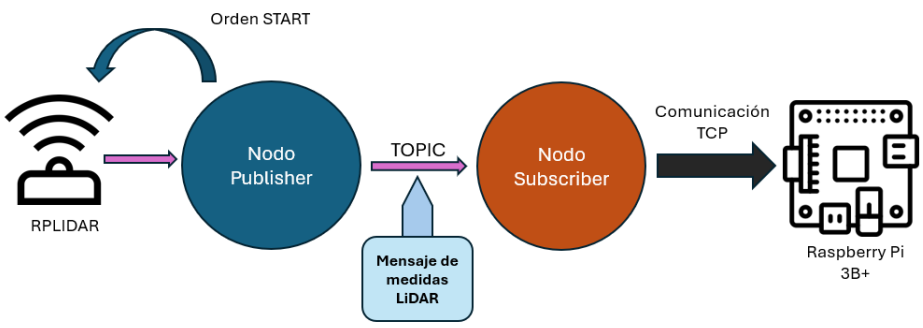


Figura 16. Diagrama de funcionamiento del programa ROS del proyecto.

4.1.1 Managed Life Cycle Nodes

El proyecto incluye una nueva estrategia de nodo conocido como managed life cycle nodes. Esto se refiere a un nodo que sigue un esquema de ciclo de vida con unas etapas conocidas de antemano.

La estrategia de un nodo life cycle permite configurar el nodo previamente a su puesta en marcha. Para un sistema con una complejidad considerable, este esquema permite asegurar que todo está preparado antes de que el nodo empiece a operar. Por ejemplo, que todos los sensores o actuadores de un robot están inicializados correctamente antes de que sea necesario usarlos. Además, en caso de encontrar un error sería posible detener todos los nodos hasta que el problema se haya resuelto gracias al estado de ShutDown [35].

Esta clase de nodos tiene un ciclo de vida similar a una máquina de estados:

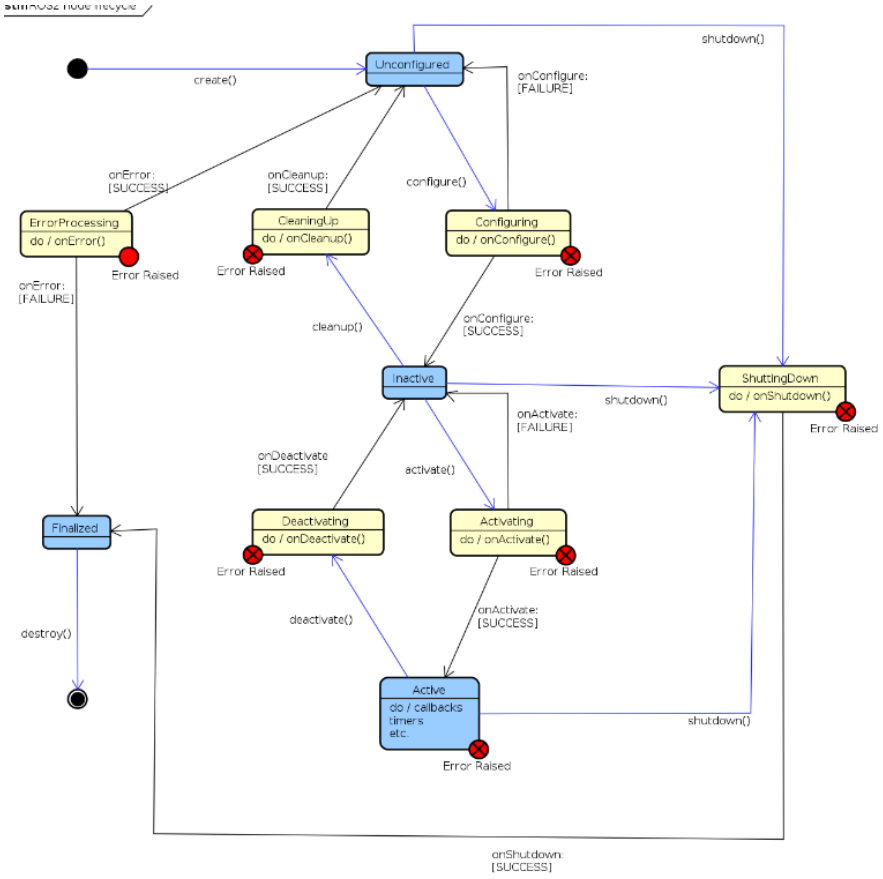


Figura 17. Diagrama de flujo del funcionamiento de un managed life cycle node. Fuente: ROS2.

Los estados se relacionan entre sí por medio de unas transiciones también conocidas. Estas transiciones pueden ser activadas por eventos clave que sean de interés y sean decisivos en el funcionamiento del nodo.

En este proyecto, el nodo *Subscriber* ha sido desarrollado siguiendo la estructura de managed life cycle node.

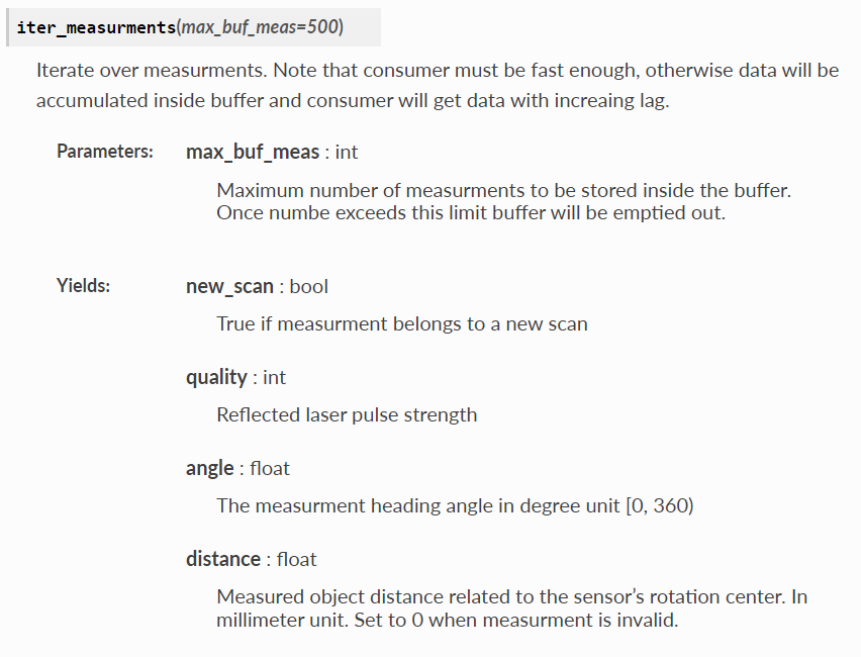
4.2 Pyhton

Los nodos ROS pueden escribirse en distintos lenguajes de programación como Java, Python o C++. Para este proyecto se ha escogido Python como lenguaje de desarrollo para todos los programas.

Python es un lenguaje de programación de alto nivel, lo que quiere decir que la sintaxis del código ha sido diseñada de tal manera que resulte entendible por un intérprete humano asimilándolo en la medida de lo posible al lenguaje natural. Además, es un lenguaje interpretado, es decir, la mayoría del código se ejecuta directamente sin necesidad de ser compilado previamente como es el caso de otros lenguajes de programación. Python se ha postulado como uno de los lenguajes más populares con una gran cantidad de colaboración por parte de muchos desarrolladores y es utilizado en muchos campos de las ciencias de la computación.

Para hacer que el desarrollo del código fuese más fluido se ha hecho uso de la librería de Python *rplidar* del usuario de GitHub SkoltechRobotics [36]. Esta librería ya tiene desarrolladas funciones compatibles con el modelo A2M8 que nos permiten inicializar el sensor y comenzar el escaneo de medidas de distancia y ángulo.

La función específica utilizada en el nodo encargado de obtener las medidas del LiDAR es la función iterativa *iter_measures*, por alguna razón el nombre de la función en la documentación de la librería no está actualizado y aparece como *iter_measurments*, lo que ocasionó errores en las primeras pruebas del nodo hasta que se descubrió que el nombre de la función real era *iter_measures* [37].



iter_measurments(*max_buf_meas=500*)

Iterate over measurments. Note that consumer must be fast enough, otherwise data will be accumulated inside buffer and consumer will get data with increaing lag.

Parameters: **max_buf_meas** : int

Maximum number of measurments to be stored inside the buffer. Once numbe exceeds this limit buffer will be emptied out.

Yields: **new_scan** : bool

True if measurment belongs to a new scan

quality : int

Reflected laser pulse strength

angle : float

The measurment heading angle in degree unit [0, 360)

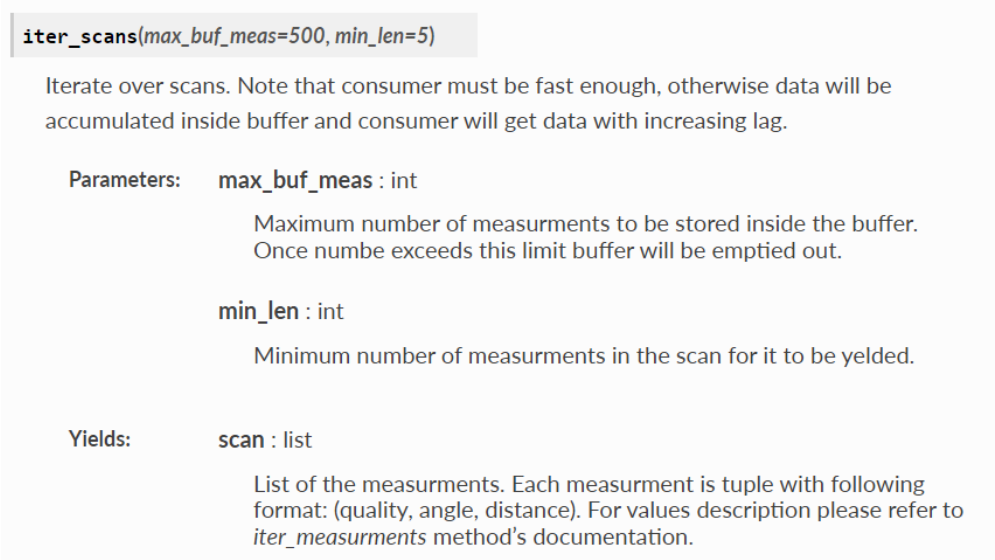
distance : float

Measured object distance related to the sensor's rotation center. In millimeter unit. Set to 0 when measurment is invalid.

Figura 18. Función de la librería rplidar encargada de obtener datos de medidas. Fuente: RPLIDAR.

Está función iterativa nos da información de cada una de las nuevas medidas que capta el sensor. Una medida está comprendida, como se ha mencionado en apartados anteriores, por una dupla de ángulo y distancia. Cada una de las iteraciones de esta nueva función nos proporcionará un nuevo punto del entorno del LiDAR. El parámetro *new_scan* hace referencia a que la medida pertenezca a un nuevo escaneado.

En relación con esto, la librería rplidar tiene otra función similar pero un poco diferente a la usada en el proyecto que es *iter_scans:*



iter_scans(*max_buf_meas=500, min_len=5*)

Iterate over scans. Note that consumer must be fast enough, otherwise data will be accumulated inside buffer and consumer will get data with increasing lag.

Parameters:

max_buf_meas : int

Maximum number of measurments to be stored inside the buffer. Once numbe exceeds this limit buffer will be emptied out.

min_len : int

Minimum number of measurments in the scan for it to be yelded.

Yields:

scan : list

List of the measurments. Each measurment is tuple with following format: (quality, angle, distance). For values description please refer to *iter_measurments* method's documentation.

Figura 19. Función de la librería rplidar que itera en escaneados. Fuente: RPLIDAR.

Durante el desarrollo del proyecto, se hicieron algunas versiones que usaban esta segunda función. Un escaneado, se entiende como una agrupación de medidas de una vuelta del sensor sobre sí mismo. Es posible monitorizar el número de medidas captadas por el sensor en cada vuelta y al final de esta, la función te devolverá una lista con todas las medidas recogidas. Cada una de las medidas sigue la misma estructura expuesta en la función *iter_measurments*.

Sin embargo, debido a que no toda la información escaneada era relevante y a la mayor complejidad de la estructura de los datos con esta segunda función, se decidió usar la primera opción para el programa final.

Es importante recalcar, que durante las pruebas se comprobó que la función iterativa de escaneados omite las medidas que están fuera del rango de distancia máximo y mínimo del sensor. Las medidas que no cumplían con este rango no se incluían en la lista de escaneado, mientras que el programa que usaba la función *iter_measurments* sí daba información de estas medidas, aunque la distancia de estas era nula.

4.3 Matlab y SIMULINK

Matlab es una plataforma de programación y cálculo numérico usada por millones de ingenieros y científicos para multitud de aplicaciones de sectores muy diversos [38]. Algunos ejemplos son los siguientes:

1	Ingeniería y Ciencias Aplicadas
	- Diseño y simulación de sistemas de control. - Análisis estructural y dinámico. - Desarrollo de algoritmos de visión por ordenador.
2	Investigación Académica
	- Analítica y visualización de datos en diversas materias de estudio. - Modelado y simulación de sistemas. - Desarrollo de algoritmos de inteligencia artificial.
3	Industria Financiera
	- Modelado y simulación de mercados. - Análisis de riesgo y gestión de carteras. - Desarrollo de algoritmos de predicción.
4	Educación
	- Desarrollo de material didáctico interactivo. - Herramienta para la enseñanza de conceptos matemáticos y técnicos. - Experimentos virtuales.
5	Medicina y Biología
	- Análisis de imágenes médicas. - Modelado de sistemas biológicos. - Análisis de datos biomédicos.

Tabla 7. Algunas aplicaciones del software Matlab en distintos campos profesionales. Fuente: Elaboración propia.

El desarrollo de este software fue producto del matemático e informático Cleve Moler. La idea de la herramienta nace de su tesis doctoral en la década de 1960. Sin embargo, Matlab no era todavía un lenguaje de programación como tal. En sus primeras versiones, funcionaba como una calculadora matricial. La primera versión de Matlab ve la luz a finales de la década de los 70 [39].

Fue en la década de los 80 cuando Cleve Moler junto con John N. Little, deciden reprogramar Matlab usando el lenguaje C, creando así el lenguaje de programación MATLAB y la disponibilidad de las *toolboxes*, paquetes de funciones y/o clases especializadas en cierta tarea [40].

En cuanto a Simulink, se trata de un entorno de programación gráfico basado en el entorno de Matlab. Permite programar en un nivel más alto de abstracción que el propio código de los archivos de texto de Matlab (.mat) ya que los programas se construyen mediante bloques que interactúan unos con otros. Es muy usado en materias de modelado de sistemas y simulación de estos, sobre todo en el ámbito de la ingeniería electrónica para el tratamiento de señales y en ingeniería de control y robótica.

Para este proyecto se ha trabajado sobre un sistema desarrollado en MATLAB-Simulink usado en las asignaturas de control en la Universidad Pontificia de Comillas. El funcionamiento del programa está compuesto por dos ficheros principales: el primero, se ejecuta en el PC del usuario y permite controlar la orden de inicio del programa además de comunicarse con el miniordenador del robot, el segundo, se ejecuta en la Raspberry Pi 3B+ y contiene los diversos drivers que interactúan con los componentes hardware del vehículo además de los algoritmos de control que toman las decisiones de rumbo y velocidad.

Para la integración del sensor LiDAR en este sistema, era preciso adaptar el mensaje de medida de distancias a lo que Simulink esperaba recibir. Para ello, dentro del bloque referente a la comunicación con el sensor LiDAR, tenemos un bloque de decodificación de mensaje.

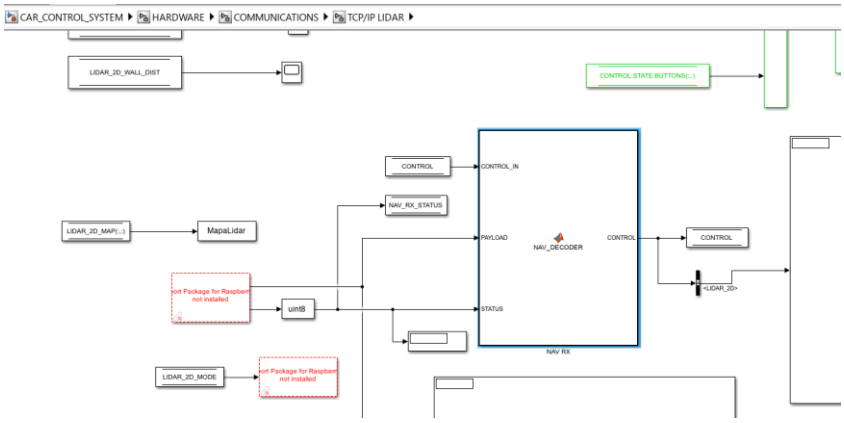


Figura 20. Bloque NAV_DECODER encargado de decodificar la información recibida desde la Raspberry Pi 4.

Dicho bloque está compuesto por un script de MATLAB que dicta su funcionamiento y es el siguiente:

```
function CONTROL  = NAV_DECODER(CONTROL_IN,PAYLOAD,STATUS)
% Initialization
CONTROL = CONTROL_IN;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MCS MESSAGES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% WRITE IN THE CONTROL BUS
if STATUS==1
    for nn = 1:20
        CONTROL.INPUT.LIDAR_2D(nn,1) = double(typecast(PAYLOAD(8*nn-4:-1:8*(nn-1)+1),'single'));
        CONTROL.INPUT.LIDAR_2D(nn,2) = double(typecast(PAYLOAD(8*nn:-1:8*nn-3),'single'));
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


return
```

Figura 21. Código del bloque NAV_DECODER.

Para poder enviar los datos de la Raspberry Pi 4 a la Raspberry Pi 3B+, que precisa de estos para poder ejecutar los algoritmos de control, se ha establecido una conexión física entre ambas mediante un cable Ethernet. Para poder enviar información es necesario transformar los datos a un formato bytes, ya que la manera de enviar información por un cable serie es mediante pulsos lógicos de ceros y unos. Una vez estos pulsos llegan a la Raspberry Pi 3B+, es necesario decodificar la información de formato bytes a un formato numérico que corresponda a los valores de las variables originales de distancia y ángulo.

El código del bloque del decodificador funciona de tal manera que crea una matriz de 20 filas y dos columnas. Cada fila corresponde a una medida donde la columna primera se identificará con el ángulo y la segunda con la distancia. Por eso, al enviar los datos desde

la Raspberry Pi 4 es necesario haberles aplicado un formato que se corresponda con esta estructura, ya que los algoritmos de Matlab esperan que los datos estén estructurados de esta manera. Por tanto, cada mensaje enviado de un ordenador a otro corresponderá a una matriz de tamaño 20:2, con el formato explicado anteriormente.

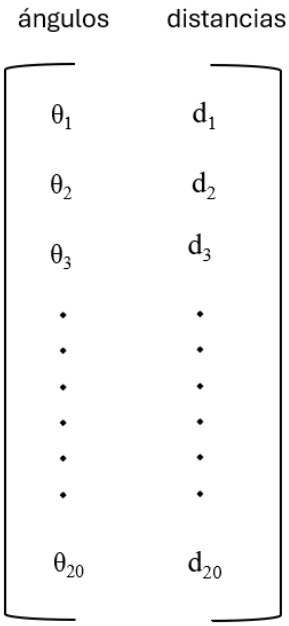


Figura 22. Formato de matriz de ángulos y distancias.

Capítulo 5. Desarrollo del trabajo experimental

El groso del proyecto es su desarrollo y parte experimental. Para poder explicar todas las etapas que se han seguido hasta obtener el resultado y diseño final, este capítulo constará de diferentes apartados que siguen las fases de desarrollo del proyecto en su orden de realización.

5.1 Nodos ROS

Se comienza con el primer objetivo del proyecto que corresponde al desarrollo de los dos nodos ROS.

- **Nodo Publisher**:
  Encargado de la obtención de las medidas en crudo
- **Nodo Lifecycle Subscriber:**
  Encargado del formateo y envío del mensaje de medidas

5.1.1 Instalación de ROS2 Humble

El primer paso es seguir la instalación de ROS2 cuyas indicaciones están recogidas en el Anexo III de esta memoria.

5.1.2 Obtención de medidas con Nodo Publisher

Este fue el punto de partida del desarrollo de los programas, poder poner en marcha el sensor de forma automática y poder leer la distancia a objetos colindantes.

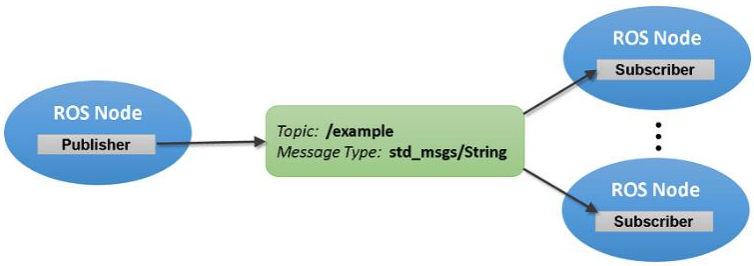


Figura 23. Diagrama de comunicación entre varios nodos ROS.

Como producto final queremos los datos en un formato de matriz 20:2 explicado anteriormente por lo que es necesario modificar el formato con el que el sensor nos da la información por defecto.

El código completo para el nodo *Publisher* desarrollado se ha ecogido en el ANEXO I. En este apartado se van a analizar las distintas partes de este para poder explicar cómo conjuntamente cumplen los objetivos de diseño marcados al comienzo del proyecto.

```
lidar = RPLidar("/dev/ttyUSB0")   # usar en Linux
#lidar = RPLidar('COM3')         #usar en Windows
```

Definimos el puerto al que conectaremos el sensor LiDAR. Según el sistema operativo donde ejecutemos el nodo, el nombre del puerto varía. Se dejaron ambas opciones debido a las distintas pruebas a lo largo del proyecto en ambos sistemas operativos, Linux y Windows. Adaptando la línea correspondiente, el código funcionó en ambos dispositivos.

```
class MinimalPublisher(Node):
    def __init__(self):
        super().init__("minimal_publisher")
        self.publisher_ = self.create_publisher(
            Float32MultiArray, "lifecycle_chatter", 10)
```

Definimos la clase del nodo ROS, en este caso un nodo *Publisher*, y algunos de sus valores característicos como el nombre identificador del nodo cuando se esté ejecutando, *minimal_publisher*, el nombre del topic y el tipo de mensaje que se va a enviar. En este caso, se ha optado por un mensaje tipo Float32MultiArray que es una matriz de datos tipo float.

```
    def timer_callback(self):
        try:
            # Creamos una matriz vacía que servirá como mensaje
            msg = Float32MultiArray()
            msg.data = []

            # Inicamos el escaneo y vamos rellenando la matriz del mensaje
            for mea in lidar.iter_measures(max_buf_meas=3000):
                msg.data.append(mea[2])  # ángulo en grados
                msg.data.append(mea[3] / 1000)  # distancia en metros

                # comprobación de las medidas que el sensor está midiendo
                print(mea[2], mea[3] / 1000, sep= " || ")

                if len(msg.data) >= 40:
                    # tenemos 20 medidas --> mandamos el mensaje y reseteamos
                    self.publisher_.publish(msg)
                    msg.data = []

        except KeyboardInterrupt:
```

```
            # Detener el escaneo y apagar el motor en caso de interrupción
            print("Escaneo interrumpido por el usuario")
```

Escribimos la función de obtención de medidas que se ejecutará indefinidamente mientras el nodo esté activo. Usaremos la librería rplidar que se ha mencionado en el apartado de Python de la memoria para poder usar la función *iter_measures*. Nos interesan las posiciones 3 y 4 de cada una de las medidas, que corresponden al ángulo y distancia respectivamente.

Es por esto, que al inicio del programa creamos una variable de tipo Float32MultiArray. Usaremos esta variable para guardar 40 datos, un ángulo y una distancia por cada una de las 20 medidas. Por ello, en cada iteración con una medida nueva, lo primero que hacemos es añadir el ángulo y distancia a nuestra variable mensaje.

Una vez el tamaño de nuestra variable mensaje llega a 40, significa que hemos guardado las 20 medidas que queríamos. Es ahora cuando hacemos uso de la principal funcionalidad del nodo ROS *Publisher.* Como ya tenemos nuestro mensaje preparado lo publicamos en el topic, el canal de comunicación que mencionábamos antes, para que el otro nodo ROS pueda recibirlo y posteriormente modificarlo. Una vez enviado, lo único que hacemos es resetear nuestro mensaje, vaciándolo para volver a empezar el proceso con nuevos datos.

```
finally:
        # nos aseguramos de desconectar y parar el LiDAR cuando el programa finalice
        print("Desconectamos")
        lidar.stop()
        lidar.stop_motor()
        lidar.disconnect()

        time.sleep(1)
        print("Desconectado nodo")
        minimal_publisher.destroy_node()
        rclpy.shutdown()
```

La parte final del código sirve para asegurarnos de que una vez se interrumpe la ejecución del nodo, el sensor se desconecta y se detiene, dejándolo listo para su nueva puesta en marcha cuando se vuelva a ejecutar el programa. En caso de no detener y desconectar el LiDAR, cuando volviésemos a ejecutar el programa obtendríamos errores, ya que no se podría establecer la conexión con un puerto que tiene ya un sensor conectado.

5.1.3 Envío de medidas con Nodo Lyfecycle Listener

Pasamos por tanto al segundo objetivo de nuestro programa ROS, la modificación del formato del mensaje y su posterior envío. El nodo actúa como *subscriber* o *listener* **del mismo topic** en el que se están publicando los mensajes de medidas del nodo *publihser*.

Vamos a analizar poco a poco el código, que se presenta completo en el Anexo II.

```
class LifecycleTalker(Node):
    modo = 'base'

    def __init__(self, node_name, **kwargs):
        self.subscription: Optional[Subscription] = None
        self.subscription  # prevent unused variable warning
        super().__init__(node_name, **kwargs)

    def listener_laser(self, msg):
        start = time.time()

        TIEMPO_MUESTRA = 0.01
        NUM_MEAS = len(msg.data)
        LEN_MSG = 20 * 2 #numero de medidas que queremos mandar por dos
        print('numero medidas: ', NUM_MEAS)

        #Creamos el mensaje de medidas
        message = numpy.full(LEN_MSG, -1.0)

        for i in range(LEN_MSG):
            if i < NUM_MEAS:
                message[i] = msg.data[i] #si no se rellena entero serán -1.0 que serán
elimindaos en matlab

        #Modificamos el mensaje
        a = bytearray(struct.pack('>ffffffffffffffffffffffffffffffffffffffff',
message[0],message[1],message[2],message[3],message[4],message[5],message[6],message[7],
message[8],message[9],message[10],message[11],message[12],message[13],message[14],messag
e[15],message[16],message[17],message[18],message[19],message[20],message[21],message[22
],message[23],message[24],message[25],message[26],message[27],message[28],message[29],me
ssage[30],message[31],message[32],message[33],message[34],message[35],message[36],messag
e[37],message[38],message[39])).hex()

        fin = bytes.fromhex(a) #transformamos a bytes y ya lo podemos mandar
        #mandamos mensaje por TCP

        if time.time()-start< TIEMPO_MUESTRA:
            time.sleep(TIEMPO_MUESTRA-(time.time()-start)) #esperamos hasta que toca
mandar el mensaje

        self.sock.send(fin)
        print('longitud mensaje: ', len(message))
        j = 0
```

```
        print('timepo: ', time.time()-start)
        for i in range(20):
            print(message[j])

            j+= 2
```

Definimos nuestra función de subscripción, que se ejecutará cada vez que escuche un mensaje publicado en el *topic* por nuestro nodo *publisher,* descrito anteriormente. Lo que hará esta función es recibir el mensaje matriz de 40 elementos que habíamos publicado, creará una variable que rellenará con esa información y una vez completa, la transformará a tipo *bytes,* que es el tipo de dato que necesitamos si queremos enviarlo por protocolo TCP/IP.

Antes de enviar el mensaje, debemos esperar hasta que pase el tiempo de muestreo definido para el proyecto, en este caso 10 ms. Así nos aseguramos de que cada mensaje nuevo se mande en un tiempo fijo. Una vez mandado, usando el comando self.sock.send(*mensaje*), el resto de las líneas funcionan como comprobación de los datos que estamos enviando.

```
    def on_configure(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_configure() is called.')

        #Seleccionamos el modo de funcionamiento del nodo que queremos
        fin = 0
        while(not fin):
            self.modo = input('LIDAR: \n')

            if(self.modo == 'LIDAR'):
                fin = 1
                print('LIDAR is saved')
                print('Vamos a establecer conexion')

                PORT = 7000
                SERVER = '192.168.2.123' #direccion IP Raspberry Pi 3B+
                ADDR = (SERVER,PORT)

                self.sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                connected = False

                while connected==False:
                    try:
                        print('Waiting...')
                        #Solicitamos unirse al servidor cada 5 segundos
                        time.sleep(5)
                        resp = self.sock.connect(ADDR)
                        if resp ==None:
```

```
                            connected = True

                    except TimeoutError:
                        print('Not connected yet...')
                    except ConnectionRefusedError:
                        print('Not connected yet...')
            else:
                fin = 0
                print('Again \n')

        return TransitionCallbackReturn.SUCCESS
```

Las siguientes funciones definidas dentro de la clase corresponden a las transiciones entre estados propias del nodo *lifecycle*. La primera de ellas corresponde a la transición de configuración.

Nuestro objetivo en esta etapa es dejar preparado el canal de comunicación entre ambas Raspberry Pi para que una vez se active el nodo, la conexión esté lista. Para ello necesitamos definir la dirección IP de la Raspberry de control, en este caso se le ha asignado una IP estática 192.168.2.123 como se explicará más adelante. Además, definimos el puerto de comunicación, en este caso el 7000, ya que es un puerto que no usa ningún otro proceso. Una vez hecho esto, simplemente solicitamos establecer la conexión cada 5 segundos hasta que esta sea exitosa.

Con esto dejaríamos preparado el canal de comunicación entre ambas Raspberry Pi y finalizaríamos el estado de configuración.

```
    def on_activate(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_activate() is called.')

        #iniciamos la subscripcion si asi lo hemos seleccionado en el estadod e
configuracion
        if (self.modo == 'LIDAR'):
            self.subscription = self.create_subscription(
                    Float32MultiArray,
                    'lifecycle_chatter',
                    self.listener_laser,
                    10)
            self.subscription

        return super().on_activate(state)
```

El objetivo de la transición de activación será crear y activar la subscripción para que el nodo comience a escuchar los mensajes que se publican en el topic. Lo que definimos es

el tipo de mensaje que tiene que recibirse, en este caso Float32MultiArray, el nombre del *topic* y la función que queremos definir como función de subscripción. Tanto el tipo de mensaje como el nombre del *topic* deben coincidir con lo establecido en el nodo *publisher* anterior, que publicaba el mensaje. Esto hará que cada vez que se reciba un mensaje por el canal especificado, se ejecute la función de subscripción que se ha explicado antes.

Una vez hecho esto, nuestro nodo ya se encuentra operativo hasta que se le dé la orden de desactivarse.

```
    def on_deactivate(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_deactivate() is called.')

        #Destruirmos la subscripción al pasar a inactivo
        self.destroy_subscription(self.subscription)

        return super().on_deactivate(state)
```

La función de desactivación simplemente destruye la subscripción para detener la funcionalidad del nodo. Sin embargo, el canal de comunicación establecido en la configuración lo mantendremos abierto. De esa manera, en el estado de desactivación podremos activar el nodo de nuevo sin tener que reconfigurarlo.

```
    def on_cleanup(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_cleanup() is called.')

        #Cerramos el socket si es que lo habiamos abierto
        if( self.modo == 'LIDAR'):
            print('closing socket')
            self.sock.close()

        return TransitionCallbackReturn.SUCCESS
```

Esta es la transición que utilizaremos en caso de querer reconfigurar el nodo. En este caso sí que cerramos el canal de comunicación establecido en la configuración inicial. Así dejaremos el nodo en el mismo estado que cuando iniciamos su ejecución.

```
    def on_shutdown(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_shutdown() is called.')

        if(self.modo == 'LIDAR'):
            print('closing socket')
            self.sock.close()
```

```
        return TransitionCallbackReturn.SUCCESS
```

Para el nodo de *shutdown* repetimos el procedimiento del nodo de reseteo, para dejar cerrado el canal de comunicación antes de detener el programa.

```
def main():
    rclpy.init()

    executor = rclpy.executors.SingleThreadedExecutor()
    lc_node = LifecycleTalker('lc_talker')
    executor.add_node(lc_node)
    try:
        executor.spin()

    except (KeyboardInterrupt, rclpy.executors.ExternalShutdownException):
        lc_node.destroy_node()
```

El *main* del programa sirve para poner en funcionamiento el nodo lifecycle con el comando spin(). Añadimos una excepción de interrupción para poder parar el programa de forma manual si queremos detener su ejecución en algún momento.

Con esto queda explicado el funcionamiento del programa que hace posible conseguir las medidas y enviarlas al ordenador que se encargará del propio control. Aquí finaliza la primera parte del proyecto, que usaba los nodos ROS 2 como herramienta.

5.2 Comunicación TCP/IP

Para que la comunicación entre ambos ordenadores funcione es necesario configurar unas direcciones IP estáticas. El proceso se recoge en detalle en el Anexo IV de la memoria.

5.3 Pruebas de coherencia de medidas

Antes de continuar con la siguiente parte del proyecto se comprobó que las medidas que recibía la Raspberry Pi 3 por el cable Ethernet eran coherentes. Para ello se decidió imprimir un mapa del circuito de ensayos para ver si la forma detectada por el sensor era similar a la real. Se guardó una variable con los datos recibidos de la Raspberry Pi 4 y se mostró en una gráfica el valor de distancias en forma polar atendiendo al ángulo de cada

medida. La variable en este caso se nombró como MapaLidar. Es importante recalcar que después de llevar a cabo este ensayo de obtención del mapa, **debe limpiarse la memoria de la Raspberry** para borrar las distintas variables generadas. Si no se borran estos datos, la memoria se llenará y no podrán realizarse más ensayos hasta liberar espacio. Se llegó a esta conclusión debido a errores al iniciar simulaciones para los ensayos ya que la memoria estaba llena.

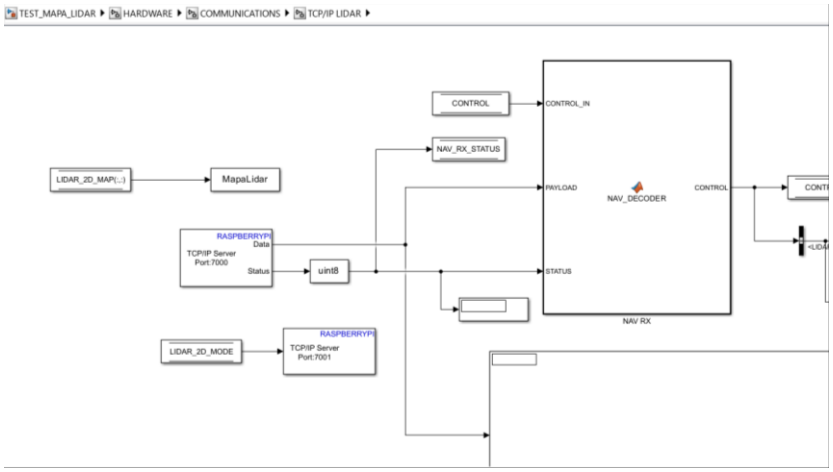


Figura 24. Archivo Simulink que muestra la variable del mapa guardada. Fuente: Elaboración propia.

Para obtener los datos del mapa es necesario realizar un ensayo ejecutando tanto los nodos ROS como los programas de Matlab/SIMULINK. Una vez esté lista la calibración de los sensores, que se lleva a cabo antes de cualquier ensayo, tendremos los datos guardados y listos para ser impresos.

En el desarrollo del proyecto, durante las primeras pruebas de generación de mapas de entorno no se tuvieron en cuenta los limites mínimos recogidos en las especificaciones del sensor LiDAR. El resultado es que cualquier objeto que se encuentre a una distancia inferior al mínimo no será detectado y el dato de distancia pasará a ser 0. Esto generaba mapas con espacios huecos que no mostraban de manera correcta la forma real del circuito que rodeaba al sensor:

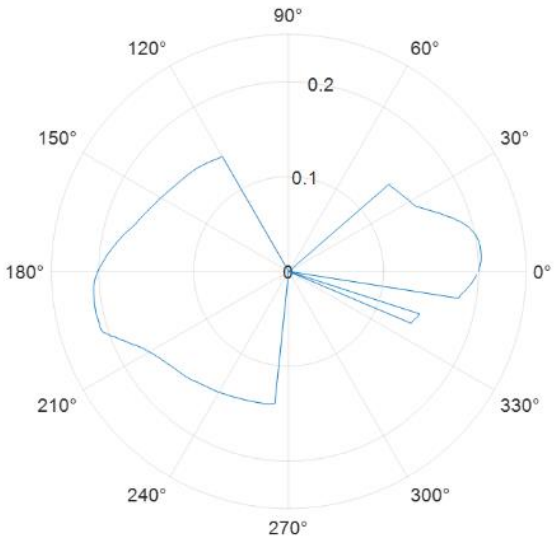


Figura 25. Mapa con espacios ciegos debido a no respetar el límite mínimo de distancia de 0.15m. Fuente: Elaboración propia.

Una vez adaptado a un circuito de mayor tamaño, el resultado del mapa obtenido fue el siguiente:

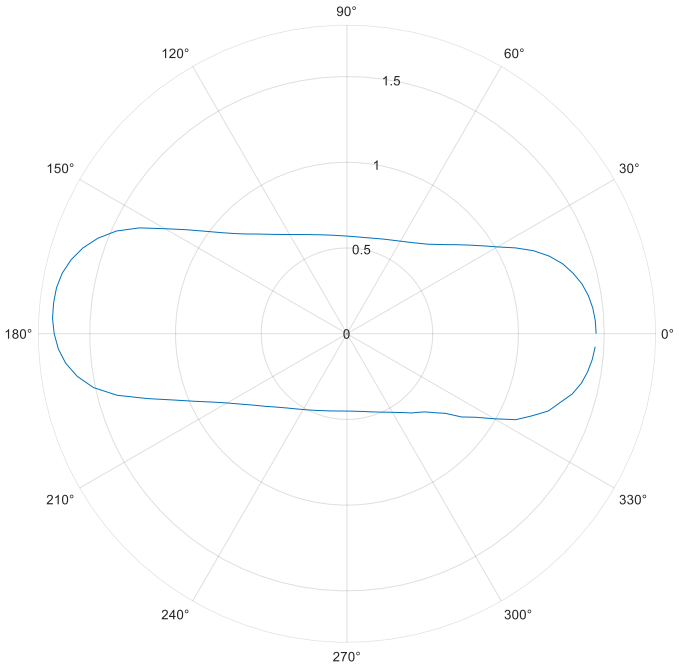


Figura 26. Mapa del circuito de ensayos

Figura 27. Foto de la situación del vehículo en el momento de captura del mapa del circuito.

También se llevó a cabo la monitorización de la señal de distancia a la pared antes de proceder con los ensayos de diseño del control. De esta manera, se aseguraba que los algoritmos de control estimaban de manera correcta la señal distancia, que es crucial en el propio algoritmo de control.

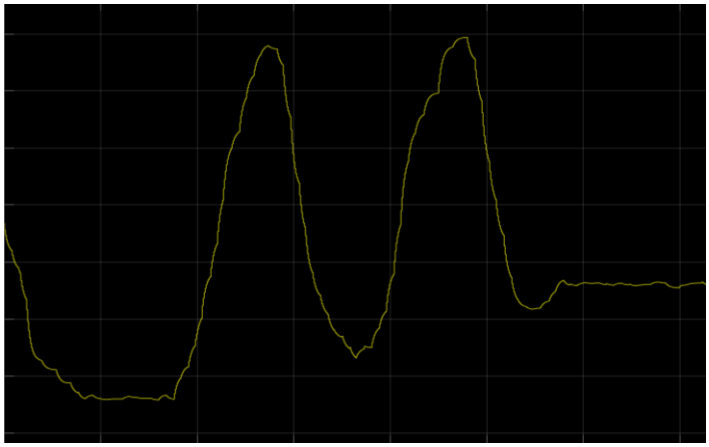


Figura 28. Monitorización de la señal distancia. Fuente: Elaboración propia.

La señal muestra la variación de distancia a la pared con el tiempo. El hecho de que el valor cambie es debido a que durante el ensayo se movió el vehículo de forma manual, modificando su posición respecto a la pared para comprobar que las lecturas eran coherentes con el movimiento real.

Se puede concluir que los resultados tienen sentido al mostrar fielmente la forma del circuito real y además ser coherente la señal de distancia. El sensor, por tanto, mide de forma correcta los objetos que tiene a su alrededor.

5.4 Procesamiento de las medidas

Una vez las medidas en crudo son recibidas por la Raspberry Pi 3B+, entran en juego los algoritmos de estimación de distancia a la pared.

5.4.1 Estimación por mínimos cuadrados

El objetivo de este proyecto es seguir una pared de referencia. Para ello nos interesa conocer la distancia del vehículo a la pared, entendiéndose esta como la distancia mínima y por tanto la distancia perpendicular a la pared pasando por el centro del vehículo. Podemos estudiar este problema con el siguiente diagrama:

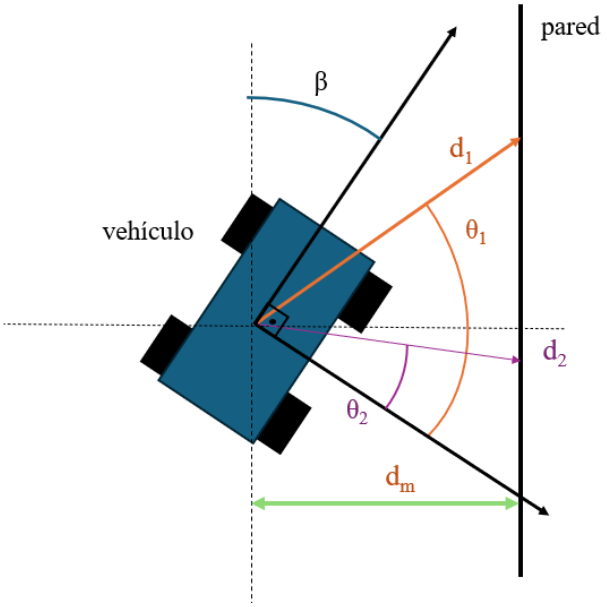


Figura 29. Diagrama del funcionamiento del método de mínimos cuadrados. Fuente. Elaboración propia.

Ahora contamos con medidas individuales, con sus respectivos ángulos y distancias, como son d_1 o d_2. Sin embargo, la distancia que necesitamos conocer de cara a los algoritmos de control es la distancia mínima entre pared y vehículo d_m, la distancia 'media'. El ángulo β corresponde al ángulo relativo entre la dirección del rumbo del vehículo y la dirección de la pared.

Si primero nos centramos en el caso de una sola medida, d_1 por ejemplo, podemos expresar, usando trigonometría básica, la distancia perpendicular como:

$$d_m = d_1 * \cos(\theta_1 - \beta)$$

Usando la fórmula trigonométrica del coseno de la resta de dos ángulos podemos transformar la expresión:

$$d_m = d_1 * (\cos(\beta) * \cos(\theta_1) + \sin(\beta) * \sin(\theta_1))$$

$$d_m = d_1 * cos\beta * cos\theta_1 + d_1 * sin\beta * sin\theta_1$$

Dividiendo ambos lados de la ecuación por la variable d_m obtenemos:

$$1 = d_1 * cos\theta_1 * \frac{cos\beta}{d_m} + d_1 * sin\theta_1 * \frac{sin\beta}{d_m}$$

Si lo expresamos de forma matricial:

$$1 = [d_1 * cos\theta_1 \quad d_1 * sin\theta_1] * \begin{bmatrix} \frac{cos\beta}{d_m} \\ \frac{sin\beta}{d_m} \end{bmatrix}$$

Esta expresión es el producto de dos matrices igualadas a uno. Si nos fijamos, la primera matriz contiene parámetros de medidas obtenidas directamente del sensor LiDAR, ángulo y distancia de un punto. Por otro lado, la segunda matriz es función de parámetros que queremos estimar, la distancia perpendicular del coche a la pared y el ángulo relativo del rumbo del vehículo con la dirección de la pared.

Ahora podemos generalizar el problema para más de una medida. Si sustituimos la primera matriz con una nueva que contenga la expresión para las 20 (o n) medidas diferentes que tenemos podremos usarlas para la estimación de las variables de distancia y ángulo relativo.

Una vez conocemos la relación entre medidas y variables a estimar, utilizamos un algoritmo de mínimos cuadrados para resolver el problema. Este algoritmo trata el problema como un problema de optimización e intenta minimizar el cuadrado del error entre medida y estimación tal que así:

$$\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

Una vez resuelto, ya tenemos el valor estimado de distancia a la pared y ángulo relativo, variables que juegan un papel fundamental en los algoritmos de control para dar órdenes a los actuadores y poder ajustar la trayectoria del vehículo a la referencia.

Es importante mencionar que, aunque el sensor LiDAR pueda obtener datos en los 360º a su alrededor, debido a que en este proyecto la pared de referencia se encuentra solo en un lado del vehículo, se pretende reducir el ángulo de visión del sensor. Definiremos una zona de ángulos relevantes para el cálculo de distancia comprendida entre los 60º y -30º, correspondientes al criterio siguiente:

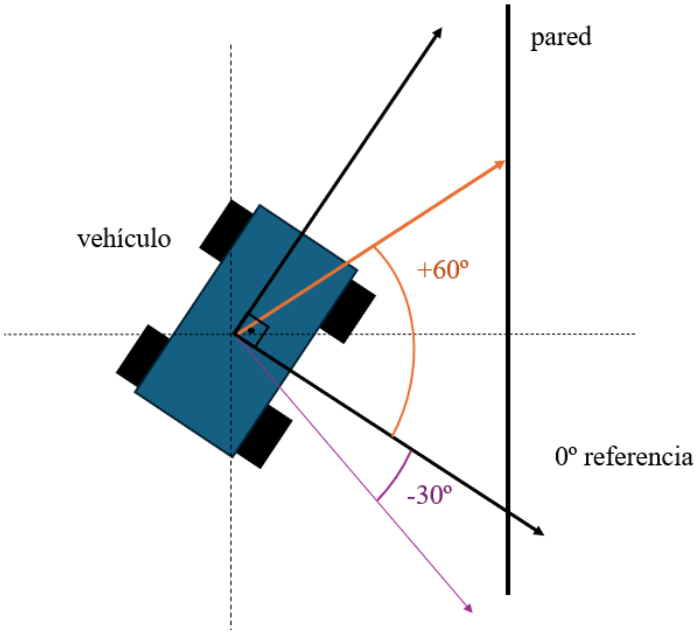


Figura 30. Rango de visión reducido del sensor. Fuente: Elaboración propia.

La razón de que el ángulo delantero sea mayor es que haciendo pruebas, se llegó a la conclusión de que aumentando este ángulo el sensor era capaz de detectar antes futuros cambios en la curvatura de la pared y, por lo tanto, responder corrigiendo su trayectoria de una forma más rápida. Las pruebas iniciales tenían ángulos simétricos respecto a la perpendicular a la pared de 45º y -45º pero la respuesta era demasiado lenta y el vehículo

tardaba demasiado en corregir su rumbo al encontrarse con una curva, llegando en algunos casos a colisionar con la pared.

El código del componente de MATLAB encargado del cálculo de mínimos cuadrados es el siguiente:

```
%-------------------------------------------------------------------------
% WALL FOLLOWER: DISTANCE AND ANGLE ESTIMATION USING THE LIDAR 2D
%-------------------------------------------------------------------------
if (CONTROL.STATE.CONTROL_MODE==4 || CONTROL.STATE.CONTROL_MODE==5) && ...
    CONTROL.STATE.WALL_FOLLOWER_MODE==1
    % The Lidar 2D 0 deg axis coincides with the vehicle negative X axis
    % The Lidar 2D rotates clockwise
    LIDAR_2D_ANG = CONTROL.EKF_WFL.PARAM.LIDAR_2D_WFL_ANG_SIGN*CONTROL.INPUT.LIDAR_2D(:,1) + ...
        CONTROL.EKF_WFL.PARAM.LIDAR_2D_WFL_ANG_OFFS;
    LIDAR_2D_DIST = CONTROL.INPUT.LIDAR_2D(:,2);
    ind = LIDAR_2D_ANG<=60 & LIDAR_2D_ANG>=-30 & ...
          LIDAR_2D_DIST>=0.1 & LIDAR_2D_DIST<=0.4;
    N = sum(ind);
    % d = dm*cos(alfa-theta) = dm*cos(alfa)*cos(theta) + dm*sin(alfa)*sin(theta)
    % Least squares to estimate cos(theta)/d and sin(theta)/d
    % 1 = [dm*cos(alfa) dm*sin(alfa)]*[cos(theta)/d sin(theta)/d]'
    if N>=5 && abs(max(LIDAR_2D_ANG(ind))-min(LIDAR_2D_ANG(ind)))>=22.5
        % B = A*param
        B = ones(N,1);
        A = [LIDAR_2D_DIST(ind).*cosd(LIDAR_2D_ANG(ind)) LIDAR_2D_DIST(ind).*sind(LIDAR_2D_ANG(ind))];
        param = A\B;
        CONTROL.INPUT.LIDAR_2D_WALL_DIST = 1/sqrt(param(1)^2+param(2)^2);
        if abs(CONTROL.INPUT.LIDAR_2D_WALL_DIST*param(2))<1
            CONTROL.INPUT.LIDAR_2D_WALL_ANG = pi/180*asind(CONTROL.INPUT.LIDAR_2D_WALL_DIST*param(2));
        end
    end
end
```

Figura 31. Código en MATLAB de cálculo de mínimos cuadrados.

Puede verse en el código como solo se seleccionan las medidas entre 60º y -30º, además de umbrales mínimos y máximos de distancia de entre 0.1 a 0.4m. Todas las demás medidas son despreciadas para realizar la estimación.

5.4.2 Filtro Extendido de Kalman

Como se ha comentado en el apartado del Estado del Arte de esta memoria, las señales provenientes de los sensores en muchas aplicaciones robóticas están alteradas por ruido. Esto hace que los valores medidos estén distorsionados llegando a influir en la respuesta del sistema.

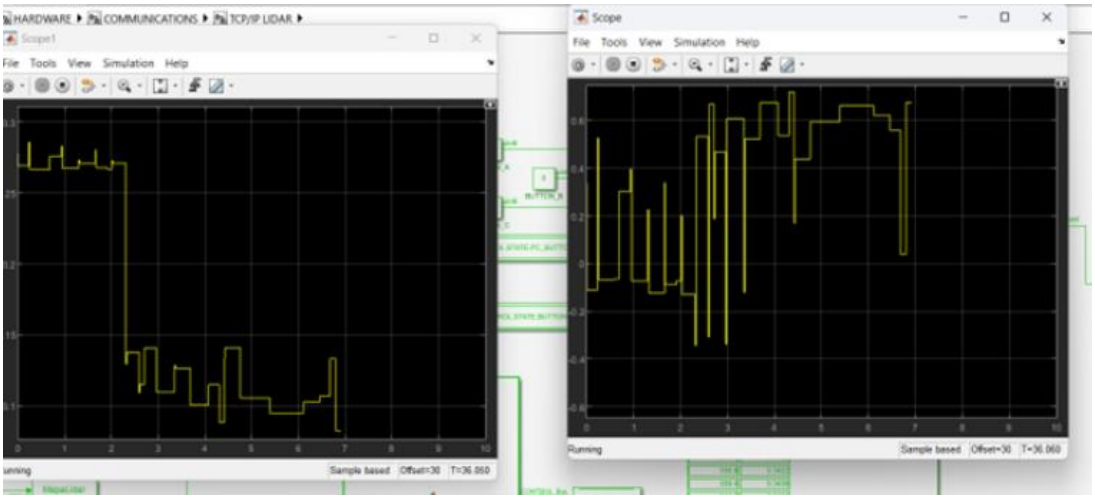


Figura 32. Señales distorsionadas por ruido previo a la aplicación del filtro de Kalman.

Como se explicó anteriormente, la versión común del Filtro de Kalman está restringida a sistemas de naturaleza lineal. En este caso, nuestro sistema tiene componentes no lineales.

MODELO NO LINEAL

$$\frac{d\omega}{dt} = -\frac{1}{T_m}\omega + \frac{K_m}{T_m}u_d$$

$$\frac{d\alpha}{dt} = \omega - \omega_{pared}$$

$$\frac{dd_n}{dt} = -(v - \omega y_A)\text{sen}(\alpha) - \omega x_A \cos(\alpha)$$

La 'velocidad de rotación de la pared' es una perturbación consistente en un pulso:

- Duración del pulso: $T_p = \frac{\pi r_{curva}}{v}$
- Amplitud del pulso: $\omega_{pared} = \frac{\pi}{T_p} = \frac{v}{r_{curva}}$ en rad/s

Figura 33. Modelo del sistema del vehículo. Fuente: ICAI.

La tercera ecuación es la relación no lineal que presenta nuestro modelo, por el hecho de contener relaciones trigonométricas. Para poder aplicar el filtrado en esta situación es necesario extender el filtro llegando a lo que se conoce como Filtro Extendido de Kalman.

Después de un proceso de linealización de la planta del vehículo, podemos obtener la siguiente función de transferencia, producto de dos funciones P1(s) y P2(s):

$$P(s) = \overbrace{\frac{K_m}{(1+T_m s)s}}^{P_1(s)} \times \overbrace{\frac{-v_o\left(1+\frac{x_A}{v_o}s\right)}{s}}^{P_2(s)} = \frac{-v_o K_m\left(1+\frac{x_A}{v_o}s\right)}{(1+T_m s)s^2}$$

Figura 34. Función de transferencia de la planta del vehículo. Fuente: ICAI.

El algoritmo del filtro funciona en dos etapas: predicción y actualización. En la etapa de predicción se estima el estado del sistema y su incertidumbre o covarianza tomando en cuenta el modelo del sistema y el estado anterior, ya que el algoritmo es iterativo.

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k$$

- $\hat{x}_{k|k-1}$ es la predicción del estado en el momento k.
- A es la matriz de transición del estado.
- $\hat{x}_{k-1|k-1}$ es la estimación del estado en el momento $k-1$.
- B es la matriz de control.
- u_k es el control del sistema en el momento k.

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q$$

donde:

- $P_{k|k-1}$ es la covarianza de la predicción del estado.
- $P_{k-1|k-1}$ es la covarianza del estado en el momento $k-1$.
- Q es la covarianza del ruido del proceso.

Una vez acabada la fase de predicción se procede a actualizar las estimaciones de estado e incertidumbre tomando en cuenta las medidas que contienen ruido. Así se obtiene una mejor estimación del verdadero estado del sistema

- **Cálculo de la ganancia de Kalman**:

$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1}$$

donde:

- K_k es la ganancia de Kalman.
- H es la matriz de observación.
- R es la covarianza del ruido de la medición.

- **Actualización del estado**:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1})$$

donde:

- $\hat{x}_{k|k}$ es la estimación del estado corregido.
- z_k es la medición en el momento k.

- **Actualización de la covarianza**:

$$P_{k|k} = (I - K_kH)P_{k|k-1}$$

donde:

- $P_{k|k}$ es la covarianza del estado corregido.

Una vez aplicada esta etapa de filtrado, tenemos listas nuestras variables de distancia y ángulo relativos a la pared para proceder a usarlas en los algoritmos de control.

5.5 Diseño del control

Una vez obtenidos los valores estimados por el método de mínimos cuadrados y filtrados con el Filtro Extendido de Kalman, pueden ejecutarse los algoritmos de control. El objetivo es que la variable estimada de distancia a la pared siga el valor constante de referencia introducido por el usuario.

El esquema de diseño que se utilizó es el mismo que usaba el coche original con los sensores de pared como fuente de medidas. Está estrategia es la implantada en los

laboratorios de las asignaturas de control del grado y se eprocede a explicar a continuación.

La idea inicial sería usar un diseño clásico de controlador con un lazo de realimentación como el siguiente diagrama:

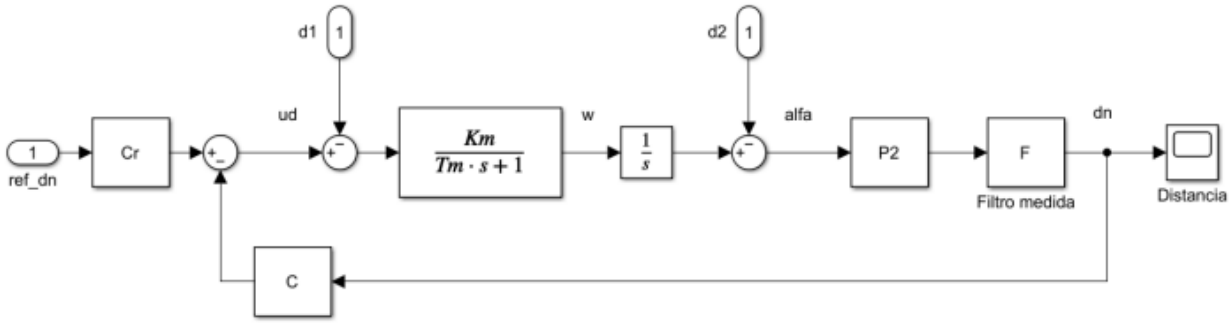


Figura 35. Propuesta de diseño clásico en lazo de realimentación. Fuente: ICAI.

En el diagrama la perturbación d_1 representa la asimetría de los motores mientras que la perturbación d_2 representa el cambio angular ante las secciones curvas del circuito. En este caso, la curva es semicircular por lo que el cambio en curvatura es constante y, por tanto, la perturbación de ángulo correspondería a una rampa. La salida del bloque controlador corresponde a la tensión diferencial de los motores.

Sin embargo, implementar este modelo de control resulta complicado. Esto se debe a que si nos fijamos en la función de transferencia de la planta P(s), esta presenta un doble integrador. Esto hace que el sistema sea bastante inestable y además tenga una respuesta lenta. La solución es adaptar al diseño a un control en cascada, con dos lazos diferentes, uno interno y otro externo. Con esta solución, se controla una variable con una dinámica más rápida con el lazo interno y así se estabiliza la parte inestable del sistema. El lazo externo controlará una variable relacionada a la anterior, pero de dinámica más lenta.

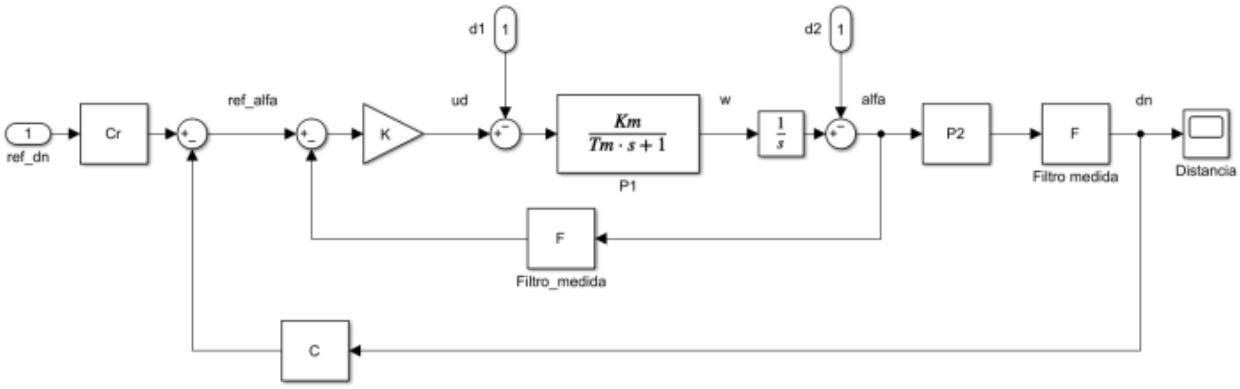


Figura 36. Diagrama de control en cascada con dos lazos. Fuente: ICAI.

En nuestro caso particular, el lazo externo controla la variable de distancia. En este segundo diseño con respecto al PID tradicional, la diferencia es que la salida del boque controlador ya no es la tensión diferencial de los motores, sino que pasa a ser la referencia de ángulo relativo del rumbo del vehículo a la pared. Esto es útil aprovechando que ese ángulo es una variable clave del sistema que puede medirse.

El lazo interno tiene la tarea de regular este ángulo relativo, que es clave en el gobierno del rumbo del vehículo. La idea es que el control sea capaz de rechazar las perturbaciones de cambio en la curvatura de la pared para tratar de mantener el rumbo del vehículo paralelo a la dirección de la pared en todo momento. En este lazo, el bloque controlador es un simple control P, una constante, y la salida de este bloque sí corresponde a la tensión diferencial de los motores.

5.6 Ensayos finales y análisis de resultados

Finalmente, el último paso antes de la finalización del proyecto es realizar ensayos en el circuito para el correcto ajuste de los parámetros de control. Se realizaron diversas pruebas para comprobar que la rapidez en la respuesta del vehículo era suficiente para girar y no colisionar con la pared del circuito y que el vehículo era capaz de recorrer todo el perímetro adaptando su velocidad y ángulo tanto a las paredes curvas como a los tramos inclinados. Al mismo tiempo, se recogían datos de las señales más relevantes del sistema para monitorizar su evolución.

En primer lugar, se realizó un ensayo con una referencia de distancia en escalón para comprobar que el vehículo era capaz de seguir los cambios ante referencia y poder analizar los transitorios de la respuesta para comprobar la rapidez y precisión del control:

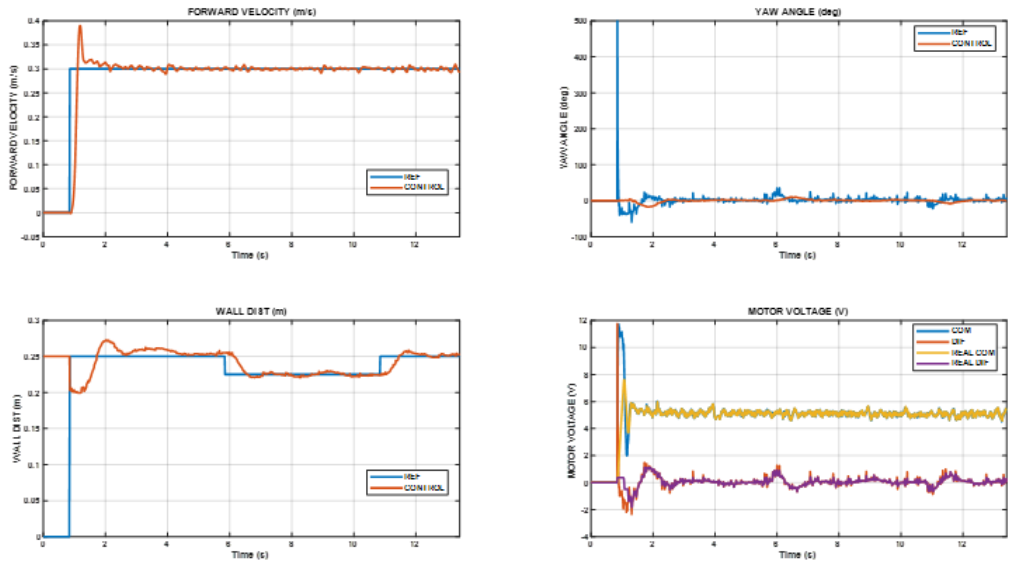


Figura 37. Ensayo ante referencia de distancia variable.

El ensayo fue exitoso. Podemos ver en el gráfico inferior izquierdo que la distancia a la pared sigue correctamente los cambios en la referencia. Esto prueba que el vehículo adapta su distancia según el valor objetivo especificado.

Una vez se había comprobado esto, se procedió a los ensayos en el circuito. Se decidió monitorizar las señales usando primero los sensores originales de pared como fuente de medidas y posteriormente el sensor LiDAR. El objetivo era comparar el comportamiento del vehículo en ambos escenarios, ya que la motivación original del proyecto era sustituir los sensores de pared originales por el nuevo sensor LiDAR. De esta forma quedaría demostrado que el cambio de sensor había sido exitoso.

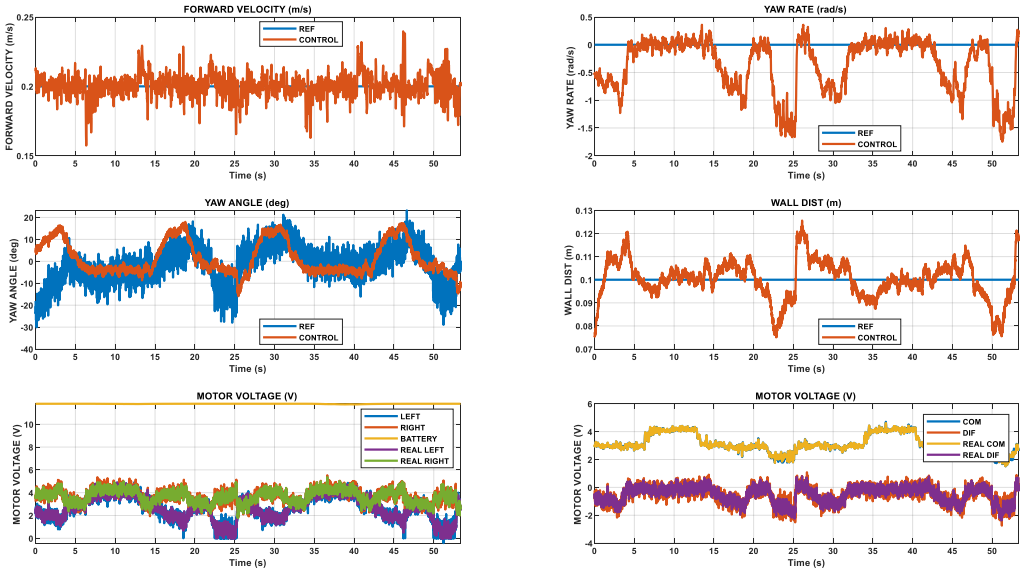


Figura 38. Ensayo en el circuito usando los sensores de pared.

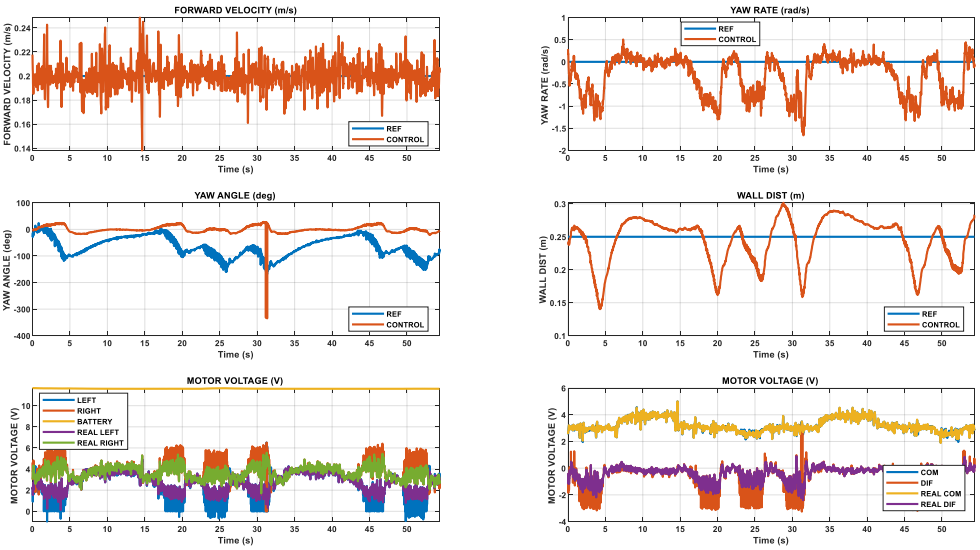


Figura 39. Ensayo en el circuito con el sensor LiDAR.

El vehículo fue capaz de recorrer todo el circuito sin colisiones, siendo capaz de desplazarse por los tramos curvos e inclinados sin problema usando como fuente de medidas el nuevo sensor LiDAR.

Además, la variable de velocidad también es regulada a la referencia establecida para los ensayos. Por tanto, puede decirse que ambos controles, distancia y velocidad, funcionan correctamente.

Capítulo 6. Conclusiones

Se puede concluir la memoria diciendo que el proyecto ha sido exitoso al haber cumplido con los objetivos establecidos en el inicio, siendo estos:

- **Generación de un software basado en ROS2 capaz de gestionar el funcionamiento del sensor y la comunicación de este con el ordenador de control del vehículo.**
  Ambos nodos ROS cumplen la función para la que fueron diseñados. Esto hace posible el control del sensor y la recepción de datos de medidas de distancia hasta llegar a la Raspberry Pi 3B+ de control.

- **Obtención de un sistema funcional en tiempo real que informe de la distancia y ángulo relativo a la pared mediante las medidas del sensor LiDAR.**
  Según se comprobó mediante la monitorización de estas señales, las medidas de ángulo y distancia relativos a la pared son coherentes con la posición real del robot, proporcionando por tanto información fiable a los algoritmos de control.

- **Correcto funcionamiento del robot en el seguimiento autónomo de pared mediante la comprobación por ensayos en el laboratorio.**
  En la parte final del proyecto se llevaron a cabo ensayos dinámicos en los que el vehículo consiguió recorrer la totalidad del circuito evitando cualquier colisión con la pared, adaptando su velocidad y rumbo.

En primer lugar, se ha conseguido desarrollar un sistema basado en dos nodos ROS2, que se ha explicado con detenimiento en la sección de Desarrollo Experimental de la memoria, que es capaz de extraer las medidas del sensor, darles un formato adecuado y finalmente, enviarlas al ordenador Raspberry Pi 3B+ de control. Por esto se puede concluir que el primer objetivo ha sido implementado de manera exitosa.

Segundamente, mediante los ensayos de monitorización de las señales de ángulo y distancia, Figura 26, se ha comprobado que el sistema responde en tiempo real a los cambios en la orientación y posición del vehículo respecto a la pared. Por tanto, nuestro

sistema proporciona datos coherentes que son la clave de funcionamiento de los algoritmos de control, cumpliendo así el segundo objetivo.

Finalmente, los ensayos finales del proyecto permitieron ajustar los parámetros de control correctamente para conseguir que el vehículo fuese capaz de recorrer tanto los tramos curvos como rectos con inclinación de la pared. La monitorización de señales, por su parte, prueba que las variables son correctamente reguladas y siguen sus respectivos valores de referencia especificados. Así se concluye cumpliendo el último de los objetivos propuestos para el proyecto.

Capítulo 7. Futuros proyectos

Para proyectos futuros sería interesante explorar cómo se podrían implantar algunas técnicas de navegación autónoma expuestas en el Estado del Arte de esta memoria.

En este proyecto el resultado era seguir una pared. Sin embargo, esta implementación hace necesario que el robot tenga siempre una pared de referencia para poder marcar su trayectoria. En caso de no encontrarla, el sistema no podría generar una ruta y el desplazamiento sería impredecible. Mediante la implementación de técnicas SLAM, se podría conseguir que el vehículo, haciendo uso del sistema LiDAR desarrollado en este proyecto, mapease el espacio que le rodea. Con este mapa, podrían llevarse a cabo técnicas de Path Planning para crear trayectorias óptimas de navegación. Así conseguiríamos un vehículo capaz de desplazarse sin la necesidad de disponer de una referencia de pared continuamente.

Otro enfoque interesante sería explorar opciones de sensores diferentes para que el vehículo contase con más información de su entorno. Sería interesante realizar un proyecto donde mediante información de vídeo recogida por cámaras y mediante algoritmos de visión por ordenador y técnicas de clasificación de inteligencia artificial, el robot pudiese distinguir objetos de su entorno y adaptar su comportamiento a los mismos. Podrían simularse señales de tráfico, semáforos y otro tipo de elementos de señalización para testear el comportamiento del vehículo ante la presencia de diferentes objetos.

Últimamente, sería interesante explorar el campo de la coordinación entre robots. Podrían realizarse proyectos donde teniendo varios de estos vehículos, estos se coordinasen para realizar tareas conjuntas o incluso formas de interés, como era el caso de los espectáculos nocturnos con drones.

Las posibilidades dentro del campo de la robótica autónoma son inmensas y también lo es el número de combinaciones de proyectos que pueden llevarse a cabo teniendo este sistema de percepción de sensor LiDAR y vehículo como base.

Bibliografía

[1] H. He, J. Gray, A. Cangelosi, Q. Meng, T. M. McGinnity and J. Mehnen, *"The Challenges and Opportunities of Human-Centered AI for Trustworthy Robots and Autonomous Systems,"* in IEEE Transactions on Cognitive and Developmental Systems, vol. 14, no. 4, pp. 1398-1412, Dec. 2022, doi: 10.1109/TCDS.2021.3132282.

[2] Madrigal Moreno, S. A., & Muñoz Ceballos, N. D. (2019). *Vehículos de guiado autónomo (AGV) en aplicaciones industriales: una revisión.* Revista Politécnica, 15(28),117-137.[fecha de Consulta 25 de Febrero de 2024]. ISSN: 1900-2351. Recuperado de: https://www.redalyc.org/articulo.oa?id=607866567012
Último acceso: 04/2024

[3] M. O. TAŞ, H. Serhan YAVUZ and A. YAZICI, *"Updating HD-Maps for Autonomous Transfer Vehicles in Smart Factories,"* (25-27 October 2018) 2018 6th International Conference on Control Engineering & Information Technology (CEIT), Istanbul, Turkey, 2018, pp. 1-5, doi: 10.1109/CEIT.2018.8751934.

[4] A. N. Catapang and M. Ramos, *"Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle,"* (25-27 November 2016) 2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Penang, Malaysia, 2016, pp. 441-445, doi: 10.1109/ICCSCE.2016.7893614.

[5] *La Automatización: Un Avance Tecnológico Que Ha Transformado La Historia*. (2023, junio 28). Conceptos de la Historia. https://conceptosdelahistoria.com/innovaciones-tecnologicas/revolucion-de-la-informacion/la-automatizacion/
Último acceso: 23/04/2024

[6] Serrano, A. R., (1996). *Navegación autónoma de robots móviles mediante lógica difusa.* [Tesis de Maestría en Ciencias Computacionales, Tecnológico de Monterrey] Repositorio Institucional – Tecnológico de Monterrey.

[7] Faisal, A., Kamruzzaman, M., Yigitcanlar, T., & Currie, G. (2019). *Understanding autonomous vehicles: A systematic literature review on capability, impact, planning and policy*. Journal of Transport and Land Use, 12(1), 45–72.

[8] Terol, M. (2021, enero 12). *Vehículos autónomos: un siglo de evolución marcado por la innovación tecnológica*. Blogthinkbig.com.

https://blogthinkbig.com/conoce-los-niveles-de-los-vehiculos-autonomos-y-sus-caracteristicas
Último acceso: 23/04/2024

[9] Matus, Daniel (2 de noviembre de 2017) *«La historia de los carros autónomos contada en unos pocos hitos»*. Digital Trends
Disponible en: https://goo.gl/rx7neJ
Último acceso: 23/04/2024

[10] Murias, D. (2018, July 26). *El primer coche autónomo fue esta furgoneta Mercedes-Benz, creada en 1986 por el ingeniero Ernst Dickmanns*. Motorpasión. https://www.motorpasion.com/tecnologia/primer-coche-autonomo-fue-esta-furgoneta-mercedes-benz-creada-1986-ingeniero-ernst-dickamnns
Último acceso: 23/04/2024

[11] *Índice 2018 de adaptación a vehículos autónomos*. (2018, January 18). KPMG. https://kpmg.com/cl/es/home/insights/2018/01/2018-01-kpmg-chile-advisory-autonomous-vehicles.html
Último acceso: 24/04/2024

[12] Corporativa, I. (n.d.). *¿Podría un "hacker" conducir mi coche autónomo? Iberdrola*. https://www.iberdrola.com/innovacion/coche-autonomo
Último acceso: 24/04/2024

[13] Otero. (2014, April 2). *Las cámaras de visión trasera serán obligatorias en los EE.UU. a partir de 2018*. Motorpasión. https://www.motorpasion.com/coches-hibridos-alternativos/las-camaras-de-vision-trasera-seran-obligatorias-en-los-ee-uu-a-partir-de-2018
Último acceso: 25/04/2024

[14] Dawkins, T. (2022, March 17). *Sensores usados en vehículos autónomos* | Level Five Supplies. Level Five Supplies. https://levelfivesupplies.com/automoviles-autonomos-101-que-sensores-se-utilizan-en-los-vehiculos-autonomos/
Último acceso: 25/04/2024

[15] Bejerano, P. G. (2021, November 15). *5 sensores: esto es lo que necesita un coche para conducir como un humano*. Blogthinkbig.com. https://blogthinkbig.com/sensores-esencia-coche-autonomo
Último acceso: 27/04/2024

[16] D. Jia, A. Hermans and B. Leibe, *"2D vs. 3D LiDAR-based Person Detection on Mobile Robots*," (23-27 October 2022) 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 2022, pp. 3604-3611, doi: 10.1109/IROS47612.2022.9981519.

[17] *What is GNSS?* (2023, December 15). EU Agency for the Space Programme. https://www.euspa.europa.eu/european-space/eu-space-programme/what-gnss
Último acceso: 27/04/2024

[18] Rivera, N. (2021, March 11). *¿Qué sensores tiene un coche autónomo y cómo funcionan?* Hipertextual. https://hipertextual.com/2017/06/sensores-coches-autonomos
Último acceso: 27/04/2024

[19] Alteryx. (2023, July 17). *Supervised vs. Unsupervised Learning; Which Is Best?* - Alteryx. https://www.alteryx.com/es/glossary/supervised-vs-unsupervised-learning#:~:text=Hay%20una%20diferencia%20clave%20entre,etiquetados%20con%20la%20respuesta%20correcta.
Último acceso: 28/04/2024

[20] Srivastava, S. (2023, December 1). *AI in Self-Driving Cars – How Autonomous Vehicles are Changing the Industry*. Appinventiv. https://appinventiv.com/blog/ai-in-self-driving-cars/#:~:text=AI%20in%20self%2Ddriving%20cars%20employs%20sensors%20and%20algorithms%20to,competent%20to%20handle%20complex%20roads.
Último acceso: 28/04/2024

[21] De La A Salinas, L. D., Carrera-Fernández, R. E., & Erazo-Velasco, I. E. (2023). *Aplicación de los Filtros de Kalman en los Sistemas de Navegación Autónoma.* Ibero-American Journal of Engineering & Technology Studies, 3(1), 198–204. https://doi.org/10.56183/iberotecs.v3i1.596
Último acceso: 28/04/2024

[22] Penizzotto, F. A. (2019). *Sistema de control basado en fusión para la navegación autónoma de vehiculos.* [Tesis Doctorado no publicada] Universidad Nacional de San Juan, Instituto de Automática. Argentina.

[23] Marquetti Gómez, Y. (2014) *Integración GPS/INS para la navegación de vehículos autónomos*. [Proyecto de Trabajo de Diploma , Universidad Central" Marta Abreu" de Las Villas]. Santa Clara, Cuba.

[24] *What is SLAM (Simultaneous Localization and Mapping)* – MATLAB & Simulink.
(n.d.). MATLAB & Simulink. https://www.mathworks.com/discovery/slam.html
Último acceso: 29/04/2024

[25] Z. Liu, *"Implementation of SLAM and path planning for mobile robots under ROS framework,"* (09-11 April 2021) 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 2021, pp. 1096-1100, doi: 10.1109/ICSP51882.2021.9408882.

[26] Sa, F. (n.d.). *What is Simultaneous Localization and Mapping (SLAM)?*
https://www.flyability.com/simultaneous-localization-and-mapping#:~:text=SLAM%20(simultaneous%20localization%20and%20mapping,map%20at%20the%20same%20time.
Último acceso: 29/04/2024

[27] *Path planning.* (n.d.). MATLAB & Simulink.
https://www.mathworks.com/discovery/path-planning.html#:~:text=Path%20planning%20lets%20an%20autonomous,and%20goal%20states%20as%20input.
Último acceso: 29/04/2024

[28] SLAMTEC Global Network. (2023, December 4). SlamTec RPLIDAR A2 - SLAMTEC Global Network. SLAMTEC Global Network - Robot Autonomous Localization and Navigation Solution (Lidar, SLAM, Robot Platform, Robotics) Supplier. https://www.slamtec.ai/product/slamtec-rplidar-a2/
Último acceso: 29/04/2024

[29] *Introducción a la robótica y la automatización. (n.d.). Introducción a La Robótica Y La Automatización.*
https://courses.minnalearn.com/es/courses/emerging-technologies/robotics-and-automation/introduction-to-robotics-and-automation/
Último acceso: 29/04/2024

[30] Paguayo. (2022, April 26). *¿Que es Raspberry Pi?* - Raspberry Pi. Raspberry Pi.
https://raspberrypi.cl/que-es-raspberry/
Último acceso: 29/04/2024

[31] Ltd, R. P. (n.d.). *Buy a Raspberry Pi 3 Model B+* – Raspberry Pi. Raspberry Pi.
https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

Último acceso: 29/04/2024

[32] *Raspberry pi 4 model B specifications* – raspberry pi. (n.d.). https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

Último acceso: 30/04/2024

[33] *ROS: Why ROS?* (n.d.). https://www.ros.org/blog/why-ros/

Último acceso: 30/04/2024

[34] *ROS/Introduction* - ROS Wiki. (n.d.). https://wiki.ros.org/ROS/Introduction

Último acceso: 30/04/2024

[35] *Managed nodes*. (n.d.-b). https://design.ros2.org/articles/node_lifecycle.html

Último acceso:30/04/2024

[36] SkoltechRobotics. (n.d.). GitHub - SkoltechRobotics/rplidar: Python module for RPLidar A1 and A2 rangefinder scanners. GitHub. https://github.com/SkoltechRobotics/rplidar

Último acceso: 30/04/2024

[37] *Welcome to RPLidar's documentation! — RPLidar 0.9.1 documentation*. (n.d.). https://rplidar.readthedocs.io/en/latest/#

Último acceso: 30/04/2024

[38] *MATLAB*. (n.d.). https://www.mathworks.com/products/matlab.html

Último acceso: 01/05/2024

[39] Chonacky, N.; Winch, D. (2005). *"Reviews of Maple, Mathematica, and Matlab: Coming Soon to a Publication Near You"*. Computing in Science & Engineering. 7 (2). Institute of Electrical and Electronics Engineers (IEEE): 9–10.

[40] T. Haigh, *"Cleve Moler: Mathematical Software Pioneer and Creator of Matlab"* in IEEE Annals of the History of Computing, vol. 30, no. 1, pp. 87-91, Jan.-March 2008, doi: 10.1109/MAHC.2008.2.

[41] Windows (binary) — *ROS 2 Documentation: Humble documentation*. (n.d.). https://docs.ros.org/en/humble/Installation/Windows-Install-Binary.html Último acceso: 01/05/2024

[42] UN. General Assembly (71st sess. : 2016-2017) ,(2017) *Resolution adopted by the General Assembly on 6 July 2017, Work of the Statistical Commission pertaining to the 2030 Agenda for Sustainable Development* *https://digitallibrary.un.org/record/1291226?v=pdf* Último acceso: 08/07/2024

[43] Moran, M. (2023, 13 septiembre). *La Agenda para el Desarrollo Sostenible - Desarrollo Sostenible*. Desarrollo Sostenible. https://www.un.org/sustainabledevelopment/es/development-agenda/ Último acceso: 01/05/2024

Anexo I: Nodo Publisher ROS 2

En este anexo se recoge el código completo del Nodo Publisher ROS 2 escrito en Python para el proyecto. La finalidad de este es gestionar la puesta en marcha del sensor LiDAR 2D y dar la orden de escaneo para la obtención de medidas. Además, funciona como un nodo Publisher por lo que debe publicar los datos en un canal de comunicación, topic, para que el segundo nodo pueda recibir esa información.

```python
import rclpy
import time

from rplidar import RPLidar
from rclpy.node import Node
from std_msgs.msg import Float32MultiArray


lidar = RPLidar("/dev/ttyUSB0")  # usar en Linux
# lidar = RPLidar('COM3')        #usar en Windows


class MinimalPublisher(Node):
    def __init__(self):
        super().__init__("minimal_publisher")
        self.publisher_ = self.create_publisher(
            Float32MultiArray, "lifecycle_chatter", 10)

    def timer_callback(self):
        try:
            # Creamos una matriz vacía que servirá como mensaje
            msg = Float32MultiArray()
            msg.data = []

            # Inicamos el escaneo y vamos rellenando la matriz del mensaje
            for mea in lidar.iter_measures(max_buf_meas=3000):
                msg.data.append(mea[2])  # angulo en grados
                msg.data.append(mea[3] / 1000)  # distancia en metros

                # comprobación de las medidas que el sensor está midiendo
                print(mea[2], mea[3] / 1000, sep=" || ")

                if len(msg.data) >= 40:
                    # tenemos 20 medidas --> mandamos el mensaje y reseteamos
                    self.publisher_.publish(msg)
                    msg.data = []

        except KeyboardInterrupt:
            # Detener el escaneo y apagar el motor en caso de interrupción
            print("Escaneo interrumpido por el usuario")


def main(args=None):
    try:
        rclpy.init(args=args)
        minimal_publisher = MinimalPublisher()

        # se ejecuta el programa de obtención de medidas indefinidamente
        minimal_publisher.timer_callback()
```

```
    except KeyboardInterrupt:
        print("Desconectado Lidar porr teclado")

    finally:
        # nos aseguramos de desconectar y parar el LiDAR cuando el programa finalice
        print("Desconectamos")
        lidar.stop()
        lidar.stop_motor()
        lidar.disconnect()

        time.sleep(1)
        print("Desconectado nodo")
        minimal_publisher.destroy_node()
        rclpy.shutdown()


if __name__ == "__main__":
    main()
```

Anexo II: Nodo Lifecycle Listener ROS 2

En este anexo queda recogido el nodo lifecycle Listener en su totalidad, explicado con detenimiento en el apartado de Desarrollo Experimental de la memoria. Su función es tomar los datos publicados por el nodo Publisher en el topic de comunicación, formatearlos y enviarlos por protocolo TCP/IP a la Raspberry `Pi 3B+. Además, cuenta con una estructura lifecycle que le permite adaptar su funcionalidad a varios casos posibles de uso.

```
from typing import Optional

import rclpy
import socket
import time
import struct
import numpy

from rclpy.lifecycle import Node
from rclpy.lifecycle import State
from rclpy.lifecycle import TransitionCallbackReturn
from rclpy.subscription import Subscription

from std_msgs.msg import Float32MultiArray

class LifecycleTalker(Node):
    modo = 'base'

    def __init__(self, node_name, **kwargs):
        self.subscription: Optional[Subscription] = None
        self.subscription  # prevent unused variable warning
        super().__init__(node_name, **kwargs)

    def listener_laser(self, msg):
        start = time.time()

        TIEMPO_MUESTRA = 0.01
        NUM_MEAS = len(msg.data)
        LEN_MSG = 20 * 2 #numero de medidas que queremos mandar por dos
        print('numero medidas: ', NUM_MEAS)

        #Creamos el mensaje de medidas
        message = numpy.full(LEN_MSG, -1.0)

        for i in range(LEN_MSG):
            if i < NUM_MEAS:
                message[i] = msg.data[i] #si no se rellena entero serán -1.0 que serán
elimindaos en matlab
```

```
        #Modificamos el mensaje
        a = bytearray(struct.pack('>ffffffffffffffffffffffffffffffffffffffff',
message[0],message[1],message[2],message[3],message[4],message[5],message[6],message[7],
message[8],message[9],message[10],message[11],message[12],message[13],message[14],messag
e[15],message[16],message[17],message[18],message[19],message[20],message[21],message[22
],message[23],message[24],message[25],message[26],message[27],message[28],message[29],me
ssage[30],message[31],message[32],message[33],message[34],message[35],message[36],messag
e[37],message[38],message[39])).hex()

        fin = bytes.fromhex(a) #transformamos a bytes y ya lo podemos mandar
        #mandamos mensaje por TCP

        if time.time()-start< TIEMPO_MUESTRA:
            time.sleep(TIEMPO_MUESTRA-(time.time()-start)) #esperamos hasta que toca
mandar el mensaje

        self.sock.send(fin)
        print('longitud mensaje: ', len(message))
        j = 0
        print('timepo: ', time.time()-start)
        for i in range(20):
            print(message[j])

            j+= 2


    def on_configure(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_configure() is called.')

        #Seleccionamos el modo de funcionamiento del nodo que queremos
        fin = 0
        while(not fin):
            self.modo = input('LIDAR: \n')

            if(self.modo == 'LIDAR'):
                print('LIDAR is saved')
                print('Vamos a establecer conexion')

                PORT = 7000
                SERVER =  '192.168.2.123' #direccion IP Rapsberry Pi 3B+
                ADDR = (SERVER,PORT)

                self.sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                connected = False

                while connected==False:
                    try:
                        print('Waiting...')
                        #Solicitamos unirse al servidor cada 5 segundos
                        time.sleep(5)
                        resp = self.sock.connect(ADDR)
                        if resp ==None:
```

```
                            connected = True

                    except TimeoutError:
                        print('Not connected yet...')
                    except ConnectionRefusedError:
                        print('Not connected yet...')
            else:
                fin = 0
                print('Again \n')

        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_activate() is called.')

        #iniciamos la subscripcion si asi lo hemos seleccionado en el estadod e
configuracion
        if (self.modo == 'LIDAR'):
            self.subscription = self.create_subscription(
                    Float32MultiArray,
                    'lifecycle_chatter',
                    self.listener_laser,
                    10)
            self.subscription

        return super().on_activate(state)

    def on_deactivate(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_deactivate() is called.')

        #Destruirmos la subscripción al pasar a inactivo
        self.destroy_subscription(self.subscription)

        return super().on_deactivate(state)

    def on_cleanup(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_cleanup() is called.')

        #Cerramos el socket si es que lo habiamos abierto
        if( self.modo == 'LIDAR'):
            print('closing socket')
            self.sock.close()

        return TransitionCallbackReturn.SUCCESS

    def on_shutdown(self, state: State) -> TransitionCallbackReturn:

        self.get_logger().info('on_shutdown() is called.')

        if(self.modo == 'LIDAR'):
```

```
            print('closing socket')
            self.sock.close()

        return TransitionCallbackReturn.SUCCESS

def main():
    rclpy.init()

    executor = rclpy.executors.SingleThreadedExecutor()
    lc_node = LifecycleTalker('lc_talker')
    executor.add_node(lc_node)
    try:
        executor.spin()

    except (KeyboardInterrupt, rclpy.executors.ExternalShutdownException):
        lc_node.destroy_node()


if __name__ == '__main__':
    main()
```

Anexo III: Instalación de ROS2 Humble

El primer paso antes de comenzar con el desarrollo de los nodos es la instalación de ROS2. Las primeras pruebas de funcionamiento de los nodos se llevaron a cabo en un ordenador con sistema operativo Windows. Esto es relevante ya que para el proyecto final estos nodos serán ejecutados en una Raspberry Pi 4, que funciona sobre un sistema operativo Linux, como se ha explicado en el apartado de Hardware.

La instalación de ROS2 puede realizarse sobre un sistema Linux o Windows, pero el proceso de instalación para cada caso es diferente. Para asegurar una instalación exitosa, se siguió el tutorial de instalación oficial de ROS2 para un sistema operativo Windows [41].

ROS2 está disponible en varias versiones. Al principio del trabajo se instaló la versión Foxy de ROS2. Sin embargo, debido a que se quería implementar un nodo con estructura managed life cycle, **se tuvo que cambiar a la versión Humble** de ROS2. Este dato es importante tenerlo en cuenta ya que, al menos en el tiempo de realización del proyecto, la versión Foxy no soportaba la generación de nodos life cycle

Anexo IV: Configuración de las IPs estáticas

En este anexo se explica cómo configurar la comunicación TCP/IP entre ambos ordenadores. Se usó un cable Ethernet para la conexión física de estos, pero hay un paso importante antes de que la comunicación sea funcional.

Para que la comunicación TCP/IP pueda funcionar debemos asignarle unas direcciones IP estáticas y conocidas a ambos ordenadores. Para esto, se deben modificar unos archivos de configuración indicando la dirección que se quiere asignar.

En el caso de la Raspberry Pi 3B+, encargada del control, se le decidió asignar la dirección 192.168.2.123. Para ello se tuvo que modificar el archivo /etc/dhcpcd.conf. Las líneas de código a añadir son las siguientes.

```
interface eth0
static ip_address =192.168.2.123/24
static routers =192.168.2.1
static domain_name_servers =192.168.2.1
```

Por otro lado, la Raspberry Pi 4 se identificó con la dirección 192.168.2.124. En este caso el archivo a modificar es el /etc/netplan/01-network-manager-all.yaml. Se deben añadir las siguientes líneas:

```
network:
        version: 2
        renderer: NetworkManager
        ethernets:
                eth0:
                        dhcp4: no
                        addresses: [192.168.2.124/24]
                gateway4: 192.168.2.1
                nameservers:
                        addresses: [192.168.2.1]
```

Una vez hecho esto, la conexión por cable Ethernet debería estar preparada para establecer la comunicación y poder enviar información de un ordenador a otro. Por tanto, podemos proceder a la parte del proyecto que se centra en la integración del sensor LiDAR con el resto de los elementos del vehículo.

Anexo A: Alineación del proyecto con los Objetivos de Desarrollo Sostenible de la ONU

Los Objetivos de Desarrollo Sostenible, también conocidos como Agenda 2030, son 17 objetivos globales interconectados que pretenden servir de guía para lograr un futuro mejor y más sostenible para todos. Los ODS fueron establecidos en 2015 por la Asamblea General de las Naciones Unidas (AG-ONU) y se pretende alcanzarlos para 2030 [42].

Constituyen un llamamiento universal a la acción para poner fin a la pobreza, proteger el planeta y mejorar las vidas y las perspectivas de las personas en todo el mundo. Actualmente, se está progresando en muchos lugares, pero el ritmo de implantación de nuevas medidas no es lo suficientemente rápido como para llegar a los objetivos marcados a tiempo [43]. Es por ello, que en esta década los esfuerzos deben acelerarse si se pretende cumplir con las metas establecidas y, por tanto, proyectos que potencien alguno de estos objetivos suponen un impulso positivo para apoyar esta iniciativa.

Se podrían identificar algunos objetivos contemplados en los ODS que están directamente relacionados con el proyecto expuesto en esta memoria:

- ## Industria Innovación e Infraestructura

La necesidad de crear robots con capacidad de movilidad autónoma surge de la búsqueda de una industria cada vez más eficiente. Al igual que la automatización de muchas otras tareas desde la Revolución Industrial, la navegación autónoma supone un gran avance en la eficiencia de operación de muchas de las industrias activas hoy en día. Esta eficiencia contribuye a un mejor funcionamiento en la actividad productiva y a su vez, a una gestión óptima de muchos recursos.

- ## Acción por el clima

Muchas de las aplicaciones de vehículos autónomos en el área del transporte han probado tener un mayor nivel de compromiso con el medio ambiente al estar muchas de ellas impulsadas con fuentes de energía alternativa y poco contaminante. Además, gracias al desarrollo de la Inteligencia Artificial, las rutas de estos vehículos autónomos se pueden optimizar mucho. Todo esto unido a otras disciplinas como la coordinación entre robots,

hace que disminuya el tráfico y los atascos en las ciudades. Así se sientan las bases de un transporte menos contaminante y más sostenible.

- Ciudades y comunidades sostenibles

Unido al punto anterior, el hecho de que estas nuevas máquinas estén impulsando la transformación del sector del transporte a un modelo más sostenible, juega un gran papel en la transformación de nuestras ciudades. Impulsando estas innovaciones tecnológicas se sientan las bases de un modelo de infraestructura mucho más sostenible.