MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MASTER

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE MEDIANTE UNA CÁMARA LIDAR

Autor: Cubillo Llanes, Diego

Co-Directora: Sáenz Nuño, María Ana

Co-Director: Zamora Macho, Juan Luis

Co-Director: Boal Martín-Larrauri, Jaime

Madrid
Agosto de 2022

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

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE MEDIANTE UNA CÁMARA LIDAR

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2021-2022 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.: Diego Cubillo Llanes Fecha: 24/08/2022

Autorizada la entrega del proyecto

LA DIRECTORA DEL PROYECTO

SAENZ NUÑO MARIA ANA - 08993694G

Firmado digitalmente por SAENZ NUÑO MARIA ANA - 08993694G
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-08993694G, givenName=MARIA ANA, sn=SAENZ NUÑO, cn=SAENZ NUÑO MARIA ANA - 08993694G
Fecha: 2022.08.25 10:53:06 +02'00'

Fdo.: María Ana Sáenz Nuño Fecha: …… / …… / ……

EL DIRECTOR DEL PROYECTO

Fdo.: Juan Luis Zamora Macho Fecha: …… / …… / ……

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2022.08.25 10:08:39 +02'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha: …… / …… / ……

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MASTER

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE MEDIANTE UNA CÁMARA LIDAR

Autor: Cubillo Llanes, Diego

Co-Directora: Sáenz Nuño, María Ana

Co-Director: Zamora Macho, Juan Luis

Co-Director: Boal Martín-Larrauri, Jaime

Madrid
Agosto de 2022

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE MEDIANTE UNA CÁMARA LIDAR

Autor: Cubillo Llanes, Diego
Codirectora: Sáenz Nuño, María Ana
Codirector: Zamora Macho, Juan Luis
Codirector: Boal Martín-Larrauri, Jaime
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Introducción

Al discontinuar el fabricante Pololu la producción de su kit de robótica Balboa 32U4, surgió la necesidad de sustituir los vehículos usados en los laboratorios de las asignaturas de control, que estaban basados en este kit. Esta situación unida a la proliferación de ideas para TFG y TFM relacionadas con la robótica móvil y la navegación puso sobre la mesa la opción de crear un vehículo multipropósito que aúne todas estas posibilidades, añadiendo a los recursos de la universidad una plataforma robótica modular y de fabricación propia.

Objetivos

Este proyecto tiene por objetivo el diseño de un nuevo vehículo terrestre basado en impresión 3D que sirva tanto para su uso en los laboratorios de las asignaturas de control como para su empleo en trabajos fin de grado y fin de master. Como muestra de su utilidad en estos últimos, el proyecto también incluye la implementación de librerías públicas para realizar SLAM (Simultaneous Localization And Mapping) y una posterior verificación del sistema de localización con un sistema de cámaras Optitrack para valorar la eficacia de esta integración. Un último objetivo es el uso de los datos obtenidos para hacer navegación punto a punto.

Solución

El diseño del vehículo partió de la elección de componentes electrónicos mostrados en la figura 1. Incluye una IMU, motores con encoders y una Raspberry Pi para cargar los programas entre otros. Todos estos componentes eran ya conocidos por otros proyectos del laboratorio de control del ICAI, se sabía que su integración sería sencilla y su funcionamiento bueno para la aplicación pensada.

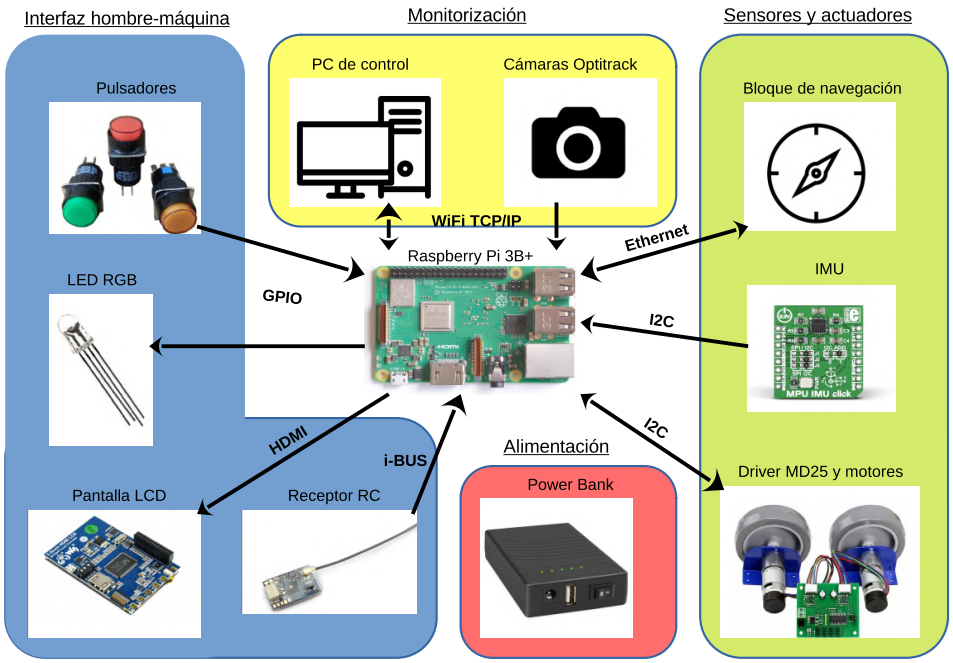


Figura 1: Esquema de componentes electrónicos

La solución consiste en un vehículo diseñado mediante el software CAD Solidworks y formado por módulos adaptables a las necesidades de fijación de los componentes electrónicos. Sobre este vehículo, un módulo independiente pero fijable al chasis se encarga de realizar tareas de más carga computacional, como es el SLAM y la lectura de datos del LiDAR.

Para el control de actuadores se conserva el código de Matlab/Simulink usado en los anteriores vehículos de laboratorio, ya que este contiene los drivers de todos los componentes electrónicos escogidos.

Para la ejecución del SLAM se opta por las librerías presentes en Navigation 2, un paquete ROS2 con herramientas orientadas a la navegación, entre ellas sistemas de localización y SLAM basados en filtros de partículas.

La verificación de posiciones se realiza con las cámaras Optitrack gestionadas por el *software* Motive. Sus plugins de integración con otras plataformas posibilitan el envío de posiciones de alta precisión en tiempo real a Matlab, facilitando el análisis de los datos obtenidos.

Resultados

El chasis se fabricó mediante impresión 3D, lo que permitió hacer iteraciones de diseño para refinar las piezas y corregir errores de diseño. El resultado fue un vehículo robusto y controlable con una autonomía que en la práctica superó los 40 minutos de funcionamiento ininterrumpido.

A continuación, se siguió el mismo procedimiento para diseñar el módulo de navegación. Su integración con el vehículo fue sencilla ya que se dejaron puntos de fijación con antelación en las placas modulares del mismo.

El diseño completo se muestra en la figura 2.

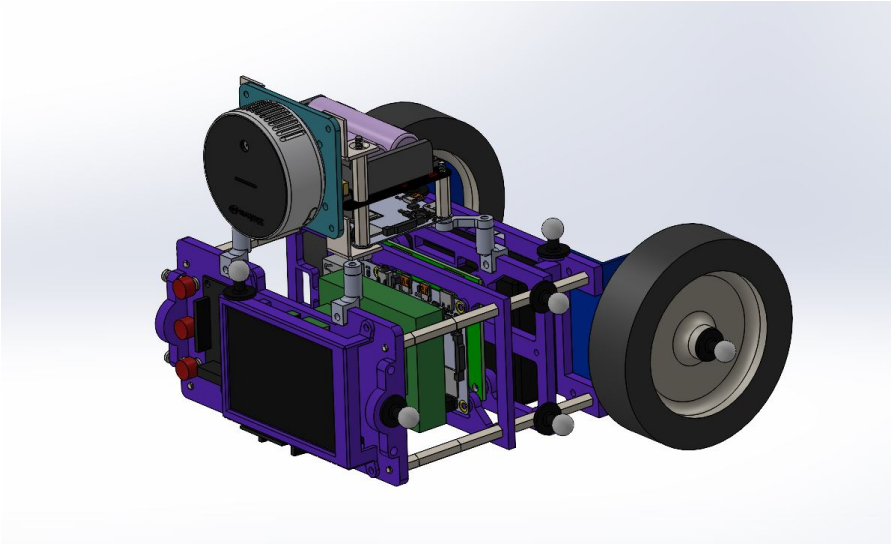


Figura 2: Diseño del vehículo con módulo de navegación

Mientras que el vehículo ejecuta sobre Raspberry Pi OS un código Matlab/Simulink de la universidad específico para la regulación automática de actuadores, el módulo de navegación ejecuta nodos ROS2 (rama Foxy) sobre Ubuntu server 20.04, por lo que se creó una conexión Ethernet entre ambos sistemas y un nodo ROS2 escrito en Python para gestionar la comunicación entre ambos, llevada a cabo usando el protocolo TCP.

El vehículo con su módulo de navegación fue modelado y se ajustaron sendos controles de tipo PI (Proporcional Integral) para controlar las velocidades de avance y giro. Tras esto se corroboró el funcionamiento del control a distancia con la emisora radiocontrol, usada para mover el vehículo mientras este escanea el entorno y compone el mapa en el que se localiza.

Se realizaron ajustes en los nombres de los mensajes ROS2, en las relaciones entre sistemas de coordenadas y en los parámetros de las diferentes librerías hasta lograr una comunicación correcta de la posición y orientación del vehículo relativas a su colocación inicial entre el control de sensores/actuadores y el algoritmo SLAM. Con este flujo de información se construyó un primer mapa, y tras ajustar los parámetros de la librería *slam-toolbox* se obtuvo un mapa de mayor calidad.

Conclusión

A pesar de que no se lograron todos los objetivos del proyecto (no se realizó la navegación punto a punto ni la verificación de las medidas obtenidas mediante SLAM), el vehículo desarrollado sirve como muestra de las posibilidades futuras, ya que integra un control de actuadores de bajo nivel con un sistema operativo de alto nivel y un paquete de librerías orientadas a la robótica en constante desarrollo como es ROS2.

El hecho de que al momento de presentar este documento el vehículo se haya usado en las asignaturas de control del ICAI y en al menos dos trabajos fin de grado muestra el comienzo de un gran abanico de posibilidades que realizar con este versátil robot móvil.

AUTONOMOUS NAVIGATION OF A TERRESTRIAL VEHICLE USING A LIDAR CAMERA

Author: Cubillo Llanes, Diego
Codirector: Sáenz Nuño, María Ana
Codirector: Zamora Macho, Juan Luis
Codirector: Boal Martín-Larrauri, Jaime
Colaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

Introduction

When the manufacturer Pololu discontinued the production of its Balboa 32U4 robotics kit, the need arose to replace the vehicles used in the laboratories of control and automation subjects, which were based on this kit. This situation, together with the proliferation of ideas for bachellor and master thesis related to mobile robotics and navigation brought to the table the option of creating a multi-purpose vehicle that brings together all these possibilities, adding to the university's resources a modular and self-made robotic platform.

Objectives

The objective of this project is the design of a new terrestrial vehicle based on 3D printing parts that can be used both in the laboratories of control and automation subjects and for use in bachellor's and master's final projects. As an example of its usefulness in the latter case, the project also includes the implementation of public libraries to perform SLAM (Simultaneous Localization And Mapping) and a subsequent verification of the localization system with an Optitrack camera system to assess the effectiveness of this integration. A final objective is the use of the data obtained to perform point-to-point navigation.

solution

The design of the vehicle started from the choice of electronic components shown in figure 3. It includes an IMU, motors with encoders and a Raspberry Pi to load the programs among others. All these components were already known from other projects of the ICAI control laboratory, it was known that their integration would be simple and their performance would be good for the intended application.

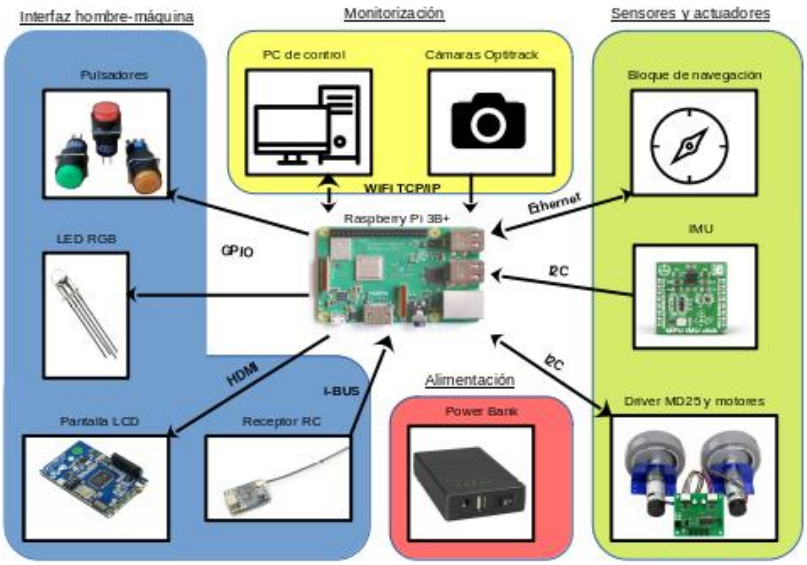


Figura 3: Electronic communications scheme

The solution consists in a vehicle designed using Solidworks CAD software and formed by modules adaptable to the fixing needs of the electronic components. On top of this vehicle, an independent module that can be fixed to the chassis is in charge of performing more computing-intensive tasks, such as SLAM and LiDAR data reading.

For the actuator control, the Matlab/Simulink code used in the previous laboratory vehicles is kept, since it contains the drivers of all the electronic components chosen.

For the SLAM implementation, the libraries present in Navigation 2, a ROS2 package with navigation-oriented tools, including localization systems and SLAM based on particle filters, are chosen.

Position verification is performed with Optitrack cameras managed by Motive software. Its integration plugins with other platforms make it possible to send high-precision positions in real time to Matlab, facilitating the analysis of the data obtained.

Results

The chassis was manufactured using 3D printing, which allowed design iterations to be made to refine parts and correct design errors. The result was a robust and controllable vehicle with an autonomy that in practice exceeded 40 minutes of uninterrupted operation.

The same procedure was then followed to design the navigation module. Its integration with the vehicle was straightforward as fixing points were left in advance on the vehicle's modular plates.

The complete design is shown in figure 4.

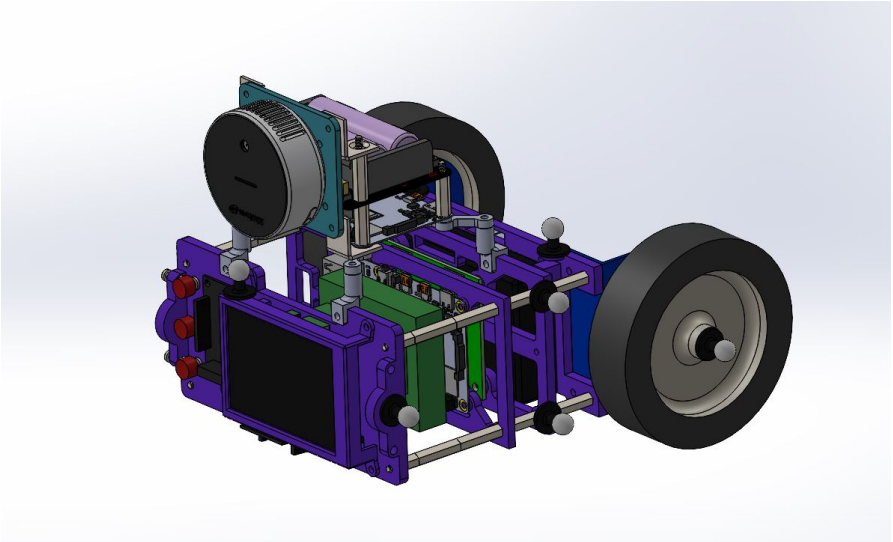


Figura 4: Vehicle assembly with its navigation module

While the vehicle runs on Raspberry Pi OS a Matlab/Simulink code from the university specific for the automatic regulation of actuators, the navigation module runs ROS2 nodes (Foxy branch) on Ubuntu server 20.04, so an Ethernet connection was created between both systems and a ROS2 node written in Python to manage the communication between them, carried out using the TCP protocol.

The vehicle with its navigation module was modeled and PI (Proportional Integral) controls were set to control the forward and yaw rates. After this, the operation of the remote control was corroborated with the radio control transmitter, used to move the vehicle while it scans the environment and composes the map on which it is located.

Adjustments were made to the names of the ROS2 messages, to the relationships between coordinate systems and to the parameters of the different li-

braries until correct communication of the position and orientation of the vehicle relative to its initial placement between the sensor/actuator control and the SLAM algorithm was achieved. With this flow of information a first map was built, and after adjusting the parameters of the slam-toolbox library a higher quality map was obtained.

Conclusion

Although not all the objectives of the project were achieved (point-to-point navigation and verification of the measurements obtained through SLAM were not performed), the developed vehicle serves as a sample of future possibilities, since it integrates a low-level actuator control with a high-level operating system and a robotics-oriented library package in constant development such as ROS2.

The fact that at the time of submitting this paper the vehicle has been used in ICAI's control subjects and in at least two final bachellor projects shows the beginning of a wide range of possibilities to be realized with this versatile mobile robot.

Agradecimientos

Quiero agradecer especialmente a Jaime Boal, María Ana Sáenz y Juan Luis Zamora, mis directores de TFM por darme su ayuda en todo momento, resolver mis dudas y haber sido claves en el recorrido de este trabajo. Han sabido darme lo necesario para avanzar y lo suficiente para aprender. También a mi padre, que en los momentos más complejos supo ayudarme a sacar lo mejor de mí y a avanzar este trabajo sin tener él ningún conocimiento de robótica, pero muchos sobre lograr objetivos.

Por supuesto, agradezco a mis amigos y compañeros de universidad que me han ayudado a conservar la cordura sacándome de vez en cuando del mundo de las librerías y las dependencias, a mi familia por animarme siempre a perseguir mis sueños y a los profesores del ICAI, que me han enseñado tantas cosas que ya no sé cómo razonaba antes de entrar a la universidad.

Un último agradecimiento a Fernando Oleo, por todo el trabajo que ha hecho desinteresadamente donando su tiempo y conocimientos en la enseñanza de software libre y de LaTeX a los alumnos de ICAI. Sin proponérselo no solo me ha ayudado en este trabajo, sino también en mi vida cotidiana.

Índice general

1. Descripción del proyecto **1**

1.1. Introducción 1

1.2. Estado de la cuestión 4

1.3. Motivación 7

1.4. Objetivos del proyecto 8

1.5. Metodología de trabajo 8

2. Diseño del vehículo **11**

2.1. Definición de requisitos 11

2.2. Componentes 15

2.3. Proceso de diseño 19

2.4. Primera versión del vehículo 21

2.4.1. Placa base 22

2.4.2. Placa 1 22

2.4.3. Placa 2 y rueda central 22

2.4.4. Placa 3 y marco de pantalla 22

2.5. Cambios realizados 23

2.6. Bloque de navegación 24

3. Implementación del software **29**

3.1. Software en el vehículo 29

3.1.1. Modelado de la planta 31

3.2. Software en el bloque de navegación 34

3.3. Creación del mapa 37

4. Conclusiones **45**

4.1. Revisión de objetivos 45

4.2. Trabajos futuros 46

Bibliografía **49**

Anexo A: Instalación del software **53**

.1. Raspberry Pi 3B+ 53

.2. Raspberry Pi 4B 4GB 55

Anexo B: Planos de los componentes del vehículo **59**

Anexo C: Objetivos de Desarrollo Sostenible **69**

Anexo D: Códigos Python **71**

Anexo E: Modelo dinámico del coche **79**

Anexo F: Referencias de imágenes utilizadas **87**

Índice de figuras

1. Esquema de componentes electrónicos VI
2. Diseño del vehículo con módulo de navegación VII
3. Electronic communications scheme X
4. Vehicle assembly with its navigation module XI

1.1. Vehículos utilizados en los laboratorios de control de ICAI 6

2.1. Fotografía del circuito del laboratorio con rampa y curvas 13
2.3. Comunicación entre componentes 18
2.4. Alimentación de los componentes 19
2.5. 4 diseños conceptuales 20
2.6. Primera versión del vehículo 21
2.7. Primer prototipo de la Placa 1 23
2.8. Placa auxiliar para sujetar la rueda de bola 24
2.9. Segunda versión del vehículo 25
2.10. Diseño conceptual del bloque de navegación 26
2.11. Anclaje estándar del bloque de navegación 27
2.12. Diseño CAD del bloque de navegación 28

3.1. Diagrama de bloques CAR_CONTROL_SYSTEM 30
3.2. Diagrama de bloques PC_CONTROL_STATION 31
3.3. Resultados del ensayo de avance 32
3.4. Resultados del ensayo de giro 33
3.5. Controles diseñados con la herramienta GUI 34
3.6. Ensayo de seguimiento de referencias de avance y giro 35

3.7. Sistemas coordenados usados para la generación del mapa 38
3.8. Nodos usados para crear el mapa 40
3.9. Primer mapa realizado por el vehículo 42
3.10. Mapa con diferentes parámetros 43
3.11. Mapa con SLAM asíncrono 44

1. Ventana del gestor de Add-Ons 53
2. Bloque de envío de posiciones por TCP 54
3. Raspberry Pi Imager 55
4. Ventana del software Realsense Viewer 57

5. ODS a los que contribuye este proyecto 69

Índice de códigos

3.1. Archivo dhcpcd.conf para configuración de comunicación Ethernet . 36
3.2. Archivo 50-cloud-init.yaml para configuración de comunicación Ethernet 36
3.3. Archivo .yaml con parámetros del mapa 41
1. Publicador de posición del vehículo en el cartografiado 71
2. Transformación estática del tf2 del LiDAR respecto al vehículo . . . 74
3. Archivo de ejecución de los nodos ros2 para cartografiar 76
4. Cálculo de posición del vehículo mediante integración de velocidades 77

Capítulo 1

Descripción del proyecto

1.1. Introducción

La robótica móvil es un nuevo paso en la evolución tecnológica que lleva ocurriendo desde el inicio de la producción de bienes. Si bien no es tan radical como la revolución industrial ocurrida en el siglo XVIII, que alteró tanto el concepto de producir como a la sociedad de la época y consolidó la concentración de las poblaciones en las ciudades, no hay duda de que supondrá un cambio de paradigma en los tipos de trabajos que se pueden automatizar y los que sigan siendo más idóneos realizados por una persona.

El concepto mismo de fábrica se basa en la eficiencia para producir más y/o mejor un producto aprovechando ventajas de organización del trabajo, distribución de tareas y empleo de maquinaria que genere fuerzas sobrehumanas (como las prensas hidráulicas), velocidad realizando el proceso (como el llenado de botellas) u homogeneidad en el producto y trazabilidad de fallos. Como vemos, hay muchos motivos para incluir maquinaria y automatizar la producción, lo que en un inicio desencadenó movimientos como el ludismo[1], caracterizado por el miedo al desempleo causado por la sustitución de puestos de trabajo por maquinaria. Mirando al pasado vemos que las labores simplemente han cambiado, y las capacidades de fabricación actuales son impensables para la sociedad de entonces.

La evolución tecnológica llegó poco a poco a la inmensa mayoría de los procesos involucrados en una industria cualquiera: El transporte dentro de la fábrica mediante elevadores de palets y cintas transportadoras, fuera de la fábrica con la mejora del transporte naval, ferroviario y en carretera (e incluso su colaboración mediante la estandarización de contenedores), en la organización de stock y pedidos con el uso de bases de datos, la producción con la alta automatización de

los procesos repetitivos e incluso la venta del producto con los nuevos sistemas de comunicación, pagos digitales y tiendas online.

Centrándonos más en la robótica, que es el tema central de este trabajo, podemos decir que en la industria los robots son usados para realizar tareas con una o varias de las siguientes características:

- Que precisen fuerzas difícilmente alcanzables por una persona. Puede ser el atornillado de piezas de gran tamaño.
- Que ocurran en entornos peligrosos. Puede ser el manejo de piezas a alta temperatura.
- Que impliquen un ambiente controlado o limpio. Un ejemplo es la cirujía, aunque en la industria propiamente dicha se puede ver en los sectores farmacéutico y aeroespacial.
- Que por su condición repetitiva sea más eficiente automatizarse. Ocurre en las cadenas de producción en serie.
- Que implican alta precisión. La miniaturización de los componentes electrónicos ha llevado a hacer prácticamente imposible su montaje a mano.
- Que puedan cambiar. La fabricación de lotes de productos diferentes en una misma fábrica hace fundamental el carácter reprogramable y flexible de un robot.

Es por todas estas ventajas que la incorporación de robots en la industria está muy presente y en auge.

Algunos de los avances en robótica más destacables a día de hoy son: la fábrica modular, los cobots, la fabricación aditiva y la robótica móvil.

La fábrica modular consiste en el empleo de maquinaria de propósito general en forma de módulos que se puedan reconfigurar en diferentes distribuciones para producir de manera más flexible. Enfrenta el problema del alto coste para fabricar productos con alta personalización y el bajo coste solo para productos sin posibilidad de personalización, buscando un punto medio con un producto personalizable y una producción eficiente.

Los cobots, o robots cooperativos, son robots diseñados para trabajar conjuntamente con personas. De esta manera, se aprovechan las capacidades de análisis visual, comprensión de las situaciones imprevistas o capacidad de coger un objeto venga en la posición que venga que tiene una persona junto con las ventajas anteriormente mostradas de un robot. El reto principal de estos sistemas es garantizar

que el entorno de trabajo sea seguro para el trabajador, para lo que se sensoriza el contorno del robot y se reduce la velocidad de trabajo hasta niveles menos peligrosos.

La fabricación controlada por ordenador como es el corte CNC, y más concretamente la impresión 3D, están suponiendo una revolución por prescindir de crear moldes y utillaje, reducir el proceso entre el diseño de una pieza y la obtención de un prototipo, y permitir crear geometrías imposibles de fabricar por métodos convencionales. Estos robots reciben de manera digital la geometría del objeto dividida en capas o "cortes", y mediante métodos tan diversos como el sinterizado por láser, la deposición de material fundido o el curado de resina activado por luz crean objetos físicos en materiales plásticos o metálicos. La baja velocidad de producción y la escasa eficiencia energética del proceso hacen que actualmente solo sea viable en objetos de geometrías complejas [2] o prototipado rápido, aunque aparecen nuevas aplicaciones en otros sectores como el de la construcción [3], y aún hay mucho margen de mejora en calidad y velocidad.

La robótica móvil consiste en el uso de robots capaces de desplazar completamente su ubicación. Un ejemplo en el sector aeroespacial sería el rover Perseverance de la NASA, un vehículo dotado de sensores y herramientas para el análisis geológico del suelo de Marte y la búsqueda de marcas biológicas que indiquen presencia de seres vivos en el pasado o presente del planeta [4]. Los usos de un robot móvil van desde la vigilancia de zonas remotas como es el caso del rover y de los drones y demás UAVs (*Unmanned Aerial Vehicle*), o peligrosas como puede ser la inspección de conductos estrechos; la extinción de incendios o la desactivación de explosivos; hasta el transporte automatizado, como el desarrollado por Amazon [5]. Estos últimos robots de transporte se van volviendo más asequibles con el abaratamiento de los sensores necesarios para su funcionamiento, y permiten sustituir los sistemas tradicionales de cintas transportadoras u otros métodos dependientes de infraestructura fija, evitando errores humanos en el almacenaje de objetos.

Un enfoque aún más radical es el de Arrival [6], una empresa fabricante de automoviles que desarrolla el concepto de microfábrica. Situando máquinas del proceso productivo sobre plataformas móviles, hacen la fábrica adaptable a distintos productos según los procesos que precise, rompiendo con el modelo de producción a gran escala en fábricas altamente especializadas. Esto no solo relaciona la robótica móvil con la fabricación modular, sino que crea un precedente en la producción industrial, pudiendo transformarse las instalaciones rápidamente para fabricar de manera eficiente un producto totalmente distinto.

1.2. Estado de la cuestión

En el conjunto de los robots móviles, una de las clasificaciones más habituales e importantes es la del sistema de navegación del robot, pudiendo ser AGV o AMR [7].

En un principio se desarrollaron los AGV (*Automated Guided Vehicles*), plataformas móviles que se guían mediante referencias que pueden ser marcas o líneas pintadas en el suelo, marcas magnéticas, códigos QR situados en las paredes o incluso sensores y emisores laser (estos últimos vehículos son conocidos como LGV)[8]. Los AGV siguen rutas predefinidas localizándose mediante estas referencias, y cuentan con sensores de proximidad para detenerse en caso de encontrar un obstáculo en su camino. Sus principales inconvenientes son su sensibilidad a los cambios en su entorno, a la perdida de estas referencias y a encontrarse obstáculos, ya que no están programados para crear una ruta alternativa que los esquive. Al depender de referencias externas también necesitan la instalación de una infraestructura de localización que debe ser mantenida y puede condicionar otros trabajos realizados en la misma superficie usada por los AGVs.

Al mejorar la tecnología, fue posible alcanzar ciertos niveles de navegación autónoma, dando lugar a los AMR (*Autonomous Mobile Robot*). Estos se caracterizan por escanear su entorno y seguir rutas que en caso de imprevistos son modificadas. Al no depender de balizas, pueden funcionar en exteriores y en lugares en que no se puedan poner referencias o estas puedan entrar en conflicto con otras labores. El precio de renunciar a la infraestructura externa de localización es un incremento en el coste de los vehículos debido a la sensorización más avanzada del robot y a la electrónica de control de navegación, que requiere una mayor potencia de cálculo.

En los últimos años está habiendo un gran auge en el uso tanto de AGV y AMR debido a la digitalización de los procesos y a la búsqueda de automatizar la industria. Mientras que los AGV siguen siendo bastante utilizados por su funcionamiento simple, su precio más asequible y su capacidad de soportar grandes cargas, los AMR son una clara evolución de los AGV que soluciona algunos problemas que puedan parar el proceso de transporte, y permiten incluso extender su rango de operación a exteriores y entornos cambiantes.

Los AMR actualmente suelen estar basados en sensores LiDAR, sonar, cámaras 2D y cámaras 3D. Estos sensores les permiten localizarse en el espacio y calcular las trayectorias para cumplir sus tareas de manera autónoma, segura y gestionando obstáculos, imprevistos y situaciones no planificadas.

Una manera de realizar esta localización para llevar a cabo la navegación autónoma es mediante SLAM (Simultaneous Localization And Mapping), una técnica

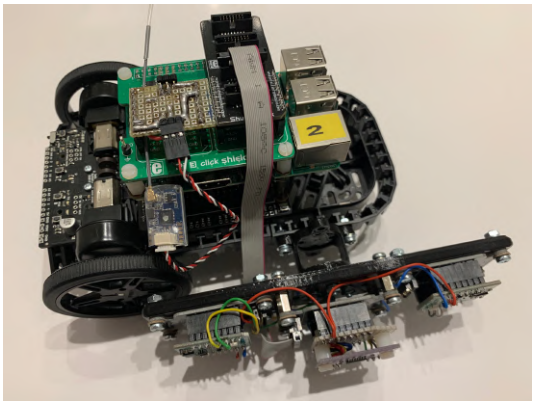
de localización en un mapa a partir de las medidas de sus sensores y actualización simultanea del mapa que el vehículo está usando en ese mismo momento para localizarse. Esto no solo supone independizarse de una instalación externa, también implica un ajuste automático del sistema sin intervención de un operario cualificado, y en caso de compartirse información entre los robots puede mejorar el funcionamiento global del proceso completo de transporte evitando zonas concurridas o temporalmente obstruidas.

SLAM se puede realizar con diferentes algoritmos que analizan de manera diferente las medidas de sus sensores, aunque todos ellos se basan en aproximaciones probabilisticas de la posición del robot respecto a las coordenadas detectadas a su alrededor. En su origen en los años 80, las fuentes de datos eran creadas en dos dimensiones, y los primeros algoritmos empleados usaban filtros extendidos de Kalman (EKF), una manera de contrastar la estimación de variables mediante un modelo matemático con medidas reales del entorno físico que la corroboren o actualicen. Otro enfoque muy usado es el FastSLAM, una forma de análisis topológico que utiliza filtros de partículas y el método Montecarlo junto con la aplicación del teorema Rao-Blackwell para reducir el número de partículas, dando una aproximación totalmente distinta a la resolución del problema SLAM y reduciendo la complejidad del problema de crecer cuadráticamente al aumentar el tamaño a crecer linealmente[9].

El desarrollo de nuevos sensores LiDAR más baratos, como son los SSL (*Solid State LiDAR*, sensores sin partes móviles que abarcan un cierto ángulo de detección[10]) hacen más atractivo el uso de robots móviles con SLAM en la industria e impulsa el desarrollo de nuevos algoritmos orientados a medidas tridimensionales, y la posibilidad de usar nuevos sensores aún más baratos como son las cámaras estéreo ha impulsado el VSLAM o Visual SLAM, una actualización de los algoritmos SLAM a las peculiaridades de estos sensores[11]. Además, el desarrollo de técnicas de aprendizaje profundo y segmentación dinámica podría reforzar la robustez de la solución al problema SLAM, descartando referencias transitorias y reforzando la unicidad de referencias fiables[12].

Aunque SLAM tiene una gran utilidad y un uso natural en la navegación de robots móviles, no es el único campo en que puede ser usado. Por ejemplo, se ha estudiado su uso en la creación de mapas 3D de ambientes que compliquen su cartografiado por las características del lugar, como es el caso de esta aplicación en una mina en Finlandia[13].

(b) Vehículo equilibrista



(a) Coche seguidor de pared

Figura 1.1: Vehículos utilizados en los laboratorios de control de ICAI

1.3. Motivación

Al momento de comenzar este trabajo, existía la necesidad de sustituir el vehículo usado en las prácticas del laboratorio de control basado en el kit *Balboa 32U4*, dado que el fabricante Pololu descontinuó la producción de este kit. Además, existía una demanda de proyectos fin de grado y fin de master relacionados con la navegación, la visión artificial y la robótica móvil.

Por estos motivos, en el momento de definir este proyecto, se decidió crear una plataforma robótica polivalente que permita por un lado usar recursos abundantemente disponibles, como son los múltiples vehículos necesarios en los laboratorios, para más usos que los laboratorios de ciertas asignaturas, y por otro lado centrar estos proyectos ambiciosos y con gran peso del software en su complejidad propia y no tanto en el medio en el que implementarlos.

Actualmente hay otras dos alternativas:

- Comprar una plataforma robótica: La existencia de kits como el Turtlebot3 resuelve la necesidad de construir plataformas propias orientadas a la robótica móvil, y tienen mucho soporte en la comunidad y foros. No obstante, no cumplen los requisitos necesarios para su uso en los laboratorios al no tener sensores de distancia a la pared, no realizar otros modos de funcionamiento especiales como puede ser el del vehículo equilibrista, y hacer más compleja la adquisición de repuestos que en un diseño ad hoc impreso en 3D.

- Usar entornos simulados. Esta opción ya se usa en la universidad, y aporta una flexibilidad absoluta a la hora de probar diferentes entornos y sensores. Sin embargo, no debería sustituir el empleo de hardware real, sino complementarlo y ayudar a su elección de componentes y puesta a prueba del sistema en las primeras etapas de desarrollo.

Por último, existe abundante software en Matlab y Simulink desarrollado en ICAI en el ámbito de control que de otra manera no se podría aprovechar para estos proyectos, o sería más difícil la adaptación de dichas plataformas conservando en el proceso sus ventajas inherentes de proyectos de código abierto, como son las librerías ROS para Turtlebot. Con el enfoque usado en este proyecto se combinan las ventajas de ambas fuentes de software, pudiendo comunicar ambos entornos entre sí y emplearlos a voluntad del usuario para adecuarse mejor a los requisitos de la aplicación que quiera llevar a cabo.

1.4. Objetivos del proyecto

Este vehículo pretende ser una demostración de la integración de códigos de control basados en Matlab y actualmente utilizados en los laboratorios por todos los alumnos de la rama electrónica con aplicaciones de navegación autónoma (y en general librerías públicas accesibles mediante ROS), y un primer paso para poder aplicar algoritmos de navegación y SLAM a otros vehículos en el futuro, pudiendo alcanzar los futuros alumnos objetivos más ambiciosos basándose en este trabajo ya hecho. El alcance de los objetivos correspondientes a este trabajo es el siguiente:

1. Diseño de un vehículo que permita su control de manera estable, soporte programación en MATLAB y ROS y contenga los sensores necesarios para realizar SLAM (Obj. 1).
2. Capacidad de realizar SLAM (localizarse en su mapa y actualizar el mismo) con los sensores presentes en el vehículo (Obj. 2).
3. Verificar la precisión de la posición calculada con SLAM contrastándola con la obtenida mediante el sistema de cámaras Optitrack (Obj. 3).
4. Navegación autónoma punto a punto utilizando las coordenadas halladas mediante SLAM (Obj. 4).

Mientras que el primer objetivo y parte del segundo son la creación de una solución a largo plazo, los demás objetivos son una muestra práctica de las posibilidades que puede ofrecer esta plataforma y que se detallan en más profundidad en la sección 4.2

1.5. Metodología de trabajo

Tras la selección de componentes hardware necesarios para que funcione el vehículo cumpliendo los requisitos de diseño, se realiza el diseño CAD de dicho vehículo mediante el software Solidworks[14]. La estructura de diseño consiste en la elección de un boceto seguida de un ciclo iterativo de diseño de prototipos usando la capacidad de prototipado rápido aportada por el uso de impresión 3D. En las distintas revisiones se consideran los fallos y correcciones necesarias para depurar el diseño hasta obtener una versión funcional y optimizada. La impresora 3D utilizada es una Ultimaker 2+[15] con filamento PLA, un material sencillo de imprimir y con buenas cualidades mecánicas para una aplicación como esta, en la que no es fundamental ni el peso ni la exposición al exterior.

Una vez obtenido el vehículo, se programa empleando Matlab/Simulink y ROS2. ROS es un conjunto de librerías muy usado en el ámbito de la robótica, y ROS2 es su segunda versión, actualmente en desarrollo. ROS organiza los procesos ejecutados en Nodos, bloques independientes que se comunican mediante mensajes llamados Topics. También incluye un servidor de parámetros propios de cada nodo y la posibilidad de solicitar servicios. Estos servicios pueden emplearse para solicitar acciones, recibir información sobre el nodo o sustituir valores internos.

En el proyecto se usan un conjunto de librerías propias y externas, muchas de estas provenientes de nav2. Nav2 es un conjunto de utilidades para ROS2 orientadas a la navegación, entre las que se incluyen la gestión de mapas y los algoritmos SLAM que se pretende usar.

Por último, se verifican las medidas calculadas con el sistema de cámaras Optitrack y el Software Motive. Este sistema con una calibración buena es capaz de ofrecer una precisión inferior al milímetro, y fue integrado con Matlab y los códigos del laboratorio de control en el trabajo fin de grado de Adriana Marroquín Rodríguez [16].

El presente documento está redactado en LaTeX[17] empleando el editor online Overleaf[18].

Capítulo 2

Diseño del vehículo

2.1. Definición de requisitos

El vehículo a diseñar pretende satisfacer una serie de motivaciones, tanto pertenecientes a la finalidad de este proyecto como para su uso en otros proyectos y laboratorios. Estas motivaciones de diseño son:

- Actualizar el *hardware* del vehículo equilibrista existente utilizado en el laboratorio de control digital (figura 1.1b), manteniendo las posibilidades que ofrecía.
- Sustituir el *hardware* que el fabricante ha descontinuado en su catálogo en el que se basaba el anterior vehículo usado en los laboratorios de regulación automática, control digital y control avanzado manteniendo sus posibilidades (control diferencial de avance y seguimiento de pared).
- Obtener mejores medidas de posición de las ruedas, dado que el encoder magnético anteriormente usado introducía mucho ruido y dificultaba su finalidad pedagógica.
- Añadir marcadores para definir su posición y orientación mediante el sistema de cámaras Optitrack.
- Añadir posibles expansiones de sensores para realizar nuevas operaciones (por ejemplo comunicación por Bluetooth/radio, seguir lineas, ubicación GPS…).
- Crear un módulo adicional con procesamiento independiente que permita realizar tareas más complejas. Para este trabajo dicha tarea serán algorit-

mos SLAM, pero puede ser lectura de marcadores fiduciales[1], identificación de objetos, aplicaciones en general más intensivas en procesamiento que el control de velocidad de los motores del vehículo.

Dado que este vehículo debe sustituir a dos vehículos ya utilizados en los laboratorios (coche y equilibrista, figura 1.1), los requisitos están muy influenciados por las actividades que se llevaban a cabo con ellos. Cumpliendo además con las motivaciones anteriormente mostradas resultan los siguientes requisitos de diseño:

- El ancho del vehículo debe rondar los 20 cm para realizar el circuito con curvas de los laboratorios de control y potencia.
- La interfaz de usuario, heredada de los anteriores vehículos y presente en otros dispositivos del laboratorio, consistirá en 3 pulsadores y un diodo LED RGB.
- El centro de gravedad debe estar lo más cerca posible de las ruedas, ya que esto facilita el control de giro y reduce la inestabilidad del vehículo cuando es usado como equilibrista.
- La IMU (siglas de *Inertial Measurement Unit*) debe estar orientada de manera que tanto en la posición de equilibrio inestable del equilibrista como en la posición horizontal del coche estén alineados los ejes de la IMU con las direcciones longitudinal, transversal y vertical del vehículo.
- Debe integrar sensores de distancia a la pared (preferiblemente los usados en los anteriores vehículos, como el de la figura 1.1a) que funcionen tanto en su disposición de coche como de equilibrista.
- Contará con marcadores reflectantes pasivos para determinar la localización y orientación del vehículo con el sistema de cámaras Optitrack.
- Los componentes estarán protegidos frente a caídas por pérdida de estabilidad o choques por fallos en el control, ya que es previsible que usándose como herramienta educativa esto suceda muy a menudo.
- Debe incluir un módulo de mayor potencia computacional que sea compatible con otros vehículos de la universidad como el utilizado en los siguientes trabajos de fin de grado [20][21]. Este modulo debería ser compatible con diversos sensores complejos y drivers externos para ahorrar trabajo no relevante (la programación propia de los drivers de una cámara en un proyecto

[1] Alejandro Ucelay Jiménez. "Navegación autónoma de un vehículo terrestre usando marcadores fiduciales". Trabajo fin de grado. ICAI - Universidad pontificia Comillas, jul. de 2022.

de identificación de objetos por aprendizaje profundo puede no ser relevante, aunque usar un driver sí es necesario para este fin).

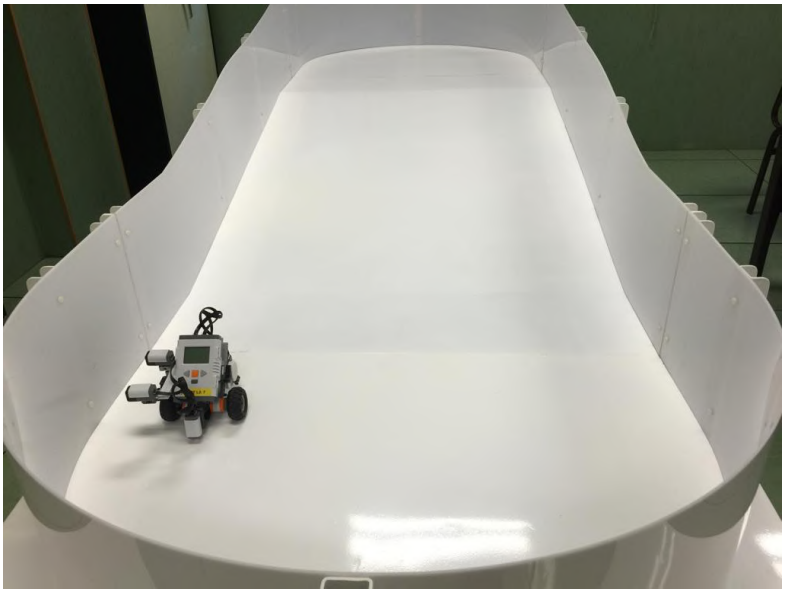
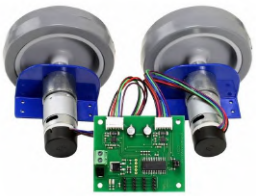


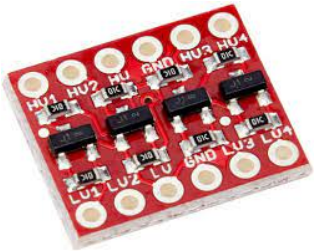
Figura 2.1: Fotografía del circuito del laboratorio con rampa y curvas

(a) Driver MD25

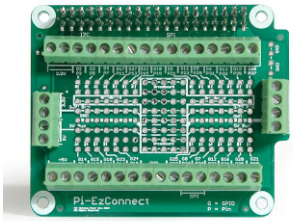


(b) Motores EMG30

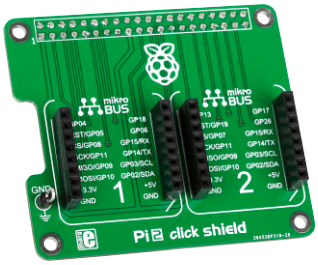
(c) Raspberry Pi 3B+



(d) Conversor 3,3-5V



(e) Pi-EZConnect



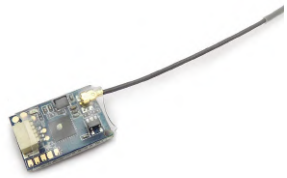
(f) Pi E Click Shield

(g) Shuttle Click

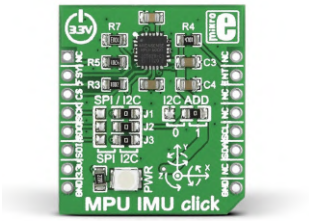
(h) MikroBUS Shuttle

(i) Power Bank

(j) Pantalla LCD



(k) Receptor RC



(l) IMU

2.2. Componentes

DRIVER MD25 . Este driver para motores DC controla los dos motores que impulsan y dirigen el vehículo. También da acceso a las lecturas de ambos encoders, de la corriente en los motores o del voltaje de la batería, permite automatizar la limitación de aceleración, escribir las velocidades de giro de manera individual, combinadas en avance y giro o incluso PWM[22]. Se comunica mediante protocolo I2C.

MOTORES EMG30 . Estos motores ya habían sido usados en otros proyectos de la universidad con buenos resultados. Constan de encoders magnéticos integrados y al comprarlos incluyen unas placas metálicas en ángulo para facilitar su instalación. Además, incluyen ruedas con superficie de rodadura de goma de 100mm de diámetro que permiten subir la rampa del circuito de la figura 2.1. Son compatibles con el driver MD25.

RASPBERRY PI 3B+ . Un miniordenador muy versátil usado a menudo en aplicaciones caseras, DIY y prototipado por contar con un sistema operativo y un procesador ARM potente sin prescindir de pines digitales accesibles con lógica de 3.3 V para hacer controles todo/nada o comunicarse con protocolos UART, SPI o I2C. Es la tercera generación del miniordenador y cuenta con conectividad WiFi y Bluetooth, puertos USB, HDMI y Ethernet.

CONVERSOR DE NIVEL LÓGICO . Dado que el driver MD25 usa lógica de 5 V y las Raspberry Pi de 3.3 V, es necesario usar entre medias un circuito que adapte la lógica y permita la comunicación bidireccional por I2C entre ambos componentes. También había una amplia experiencia en el uso de este tipo de componente en otros proyectos de la universidad empleando el mismo protocolo.

PI-EZCONNECT . Este hat de expansión permite la conexión de cables desnudos mediante terminales de tornillo para acceder a los pines GPIO de la

Raspberry Pi. Se utilizará para conectar los botones (pulsadores genéricos con un contacto común, un contacto NO y otro NC), el diodo LED RGB (también genérico, con cátodo común) y los 4 cables de comunicación I2C para el driver MD25.

PI E CLICK SHIELD . Este hat permite conectar dos sensores de la marca MikroE, ya utilizada en los sensores de distancia a pared del anterior coche del laboratorio. La principal ventaja de esta marca es el extendido catálogo de sensores, todos ellos con un mismo conector, mikroBUS, y en general un formato de tamaño parecido. Esto aporta flexibilidad a la hora de elegir qué sensores conectar para la aplicación que se desee hacer con el vehículo.

SHUTTLE CLICK . Permite aumentar el número de sensores que se pueden conectar simultaneamente y posicionarlos en sitios separados de la Raspberry Pi mediante cables planos. Este componente es necesario en particular para comunicarse con los sensores de distancia a pared, que van ambos conectados mediante un único cable a una de las 4 bahías de este componente.

MIKROBUS SHUTTLE . Este componente convierte de vuelta el cable plano que sale del shuttle click al socket mikroBUS al que se conectan los sensores. Con estos, el número mínimo de sensores que se pueden conectar al vehículo es de 4.

POWER BANK . El modelo YB1203000-USB ofrece simultaneamente alimentación a 5V de hasta 2A y a 12V de hasta 3A y tiene una capacidad de 33.3 W h que puede alimentar al vehículo hasta 4 horas parado y hasta 1 hora con todos los sistemas funcionando continuamente a máxima potencia. Esta estimación se ha realizado considerando un consumo medio de la Raspberry Pi de 0,55A[2] y sabiendo que la Power Bank suministra un máximo de

[2]Datos sobre Raspberry Pi: `https://www.raspberrypi.com/documentation/computers/raspberry-pi.html`

3A y el driver MD25 consume potencia a 12V:

$$\frac{33,3Wh}{(5*0,55)+(12*2,45)} = 1,036h \tag{2.1}$$

PANTALLA LCD 3,5" . Además de indicar la dirección IP de cada vehículo al encenderse, permite saber si el sistema operativo se ha iniciado correctamente antes de comenzar la comunicación externa. En un futuro podría mostrar interfaces gráficas con más posibilidades que el LED RGB.

RECEPTOR FLYSKY FS-A8S . Mediante este receptor de radiofrecuencia habitual en aplicaciones de aeromodelismo se pueden enviar referencias al vehículo con un mando radiocontrol, ya sean constantes usando los interruptores o variables mediante las lecturas de joystick. La comunicación con la Raspberry Pi se realiza por i-BUS. En el laboratorio de control digital ya se usaba un mando similar para elegir entre 3 referencias para el vehículo equilibrista, y en este proyecto será de ayuda para generar los primeros mapas del entorno sin pre-establecer el movimiento y recorrido del vehículo.

MPU IMU CLICK . Este sensor de MikroE integra una IMU MPU-6000 con giroscopio y acelerómetro de 3 ejes. Este componente puede ser usado tanto para la monitorización como para el control del vehículo, y es fundamental en el funcionamiento del vehículo equilibrista para lograr su estabilidad.

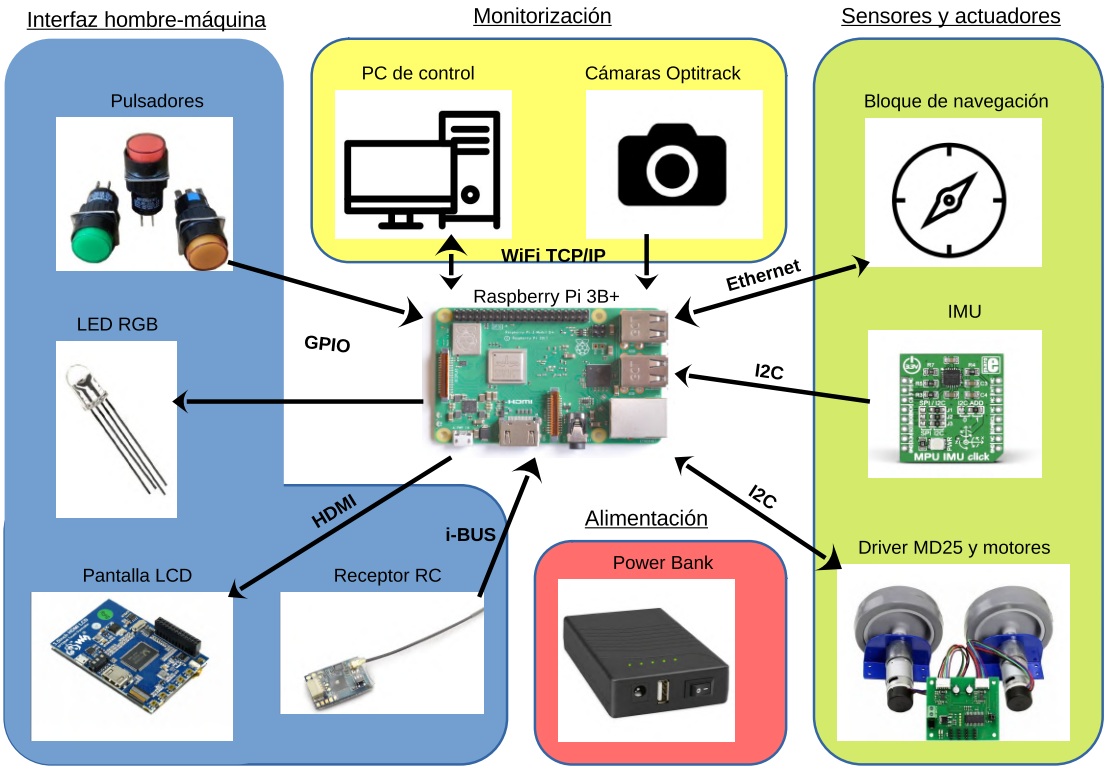


Figura 2.3: Comunicación entre componentes

En las figuras 2.3 y 2.4 se describe la comunicación entre componentes y la alimentación de los mismos respectivamente. No se muestran los componentes de conexionado.

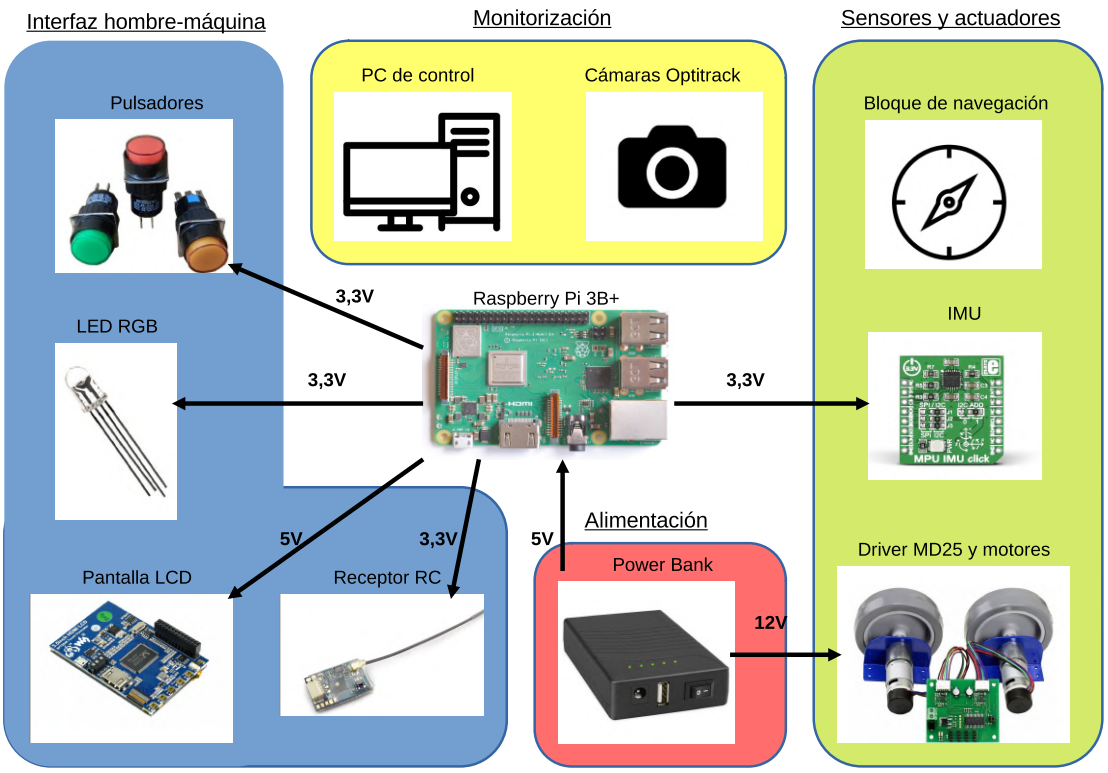
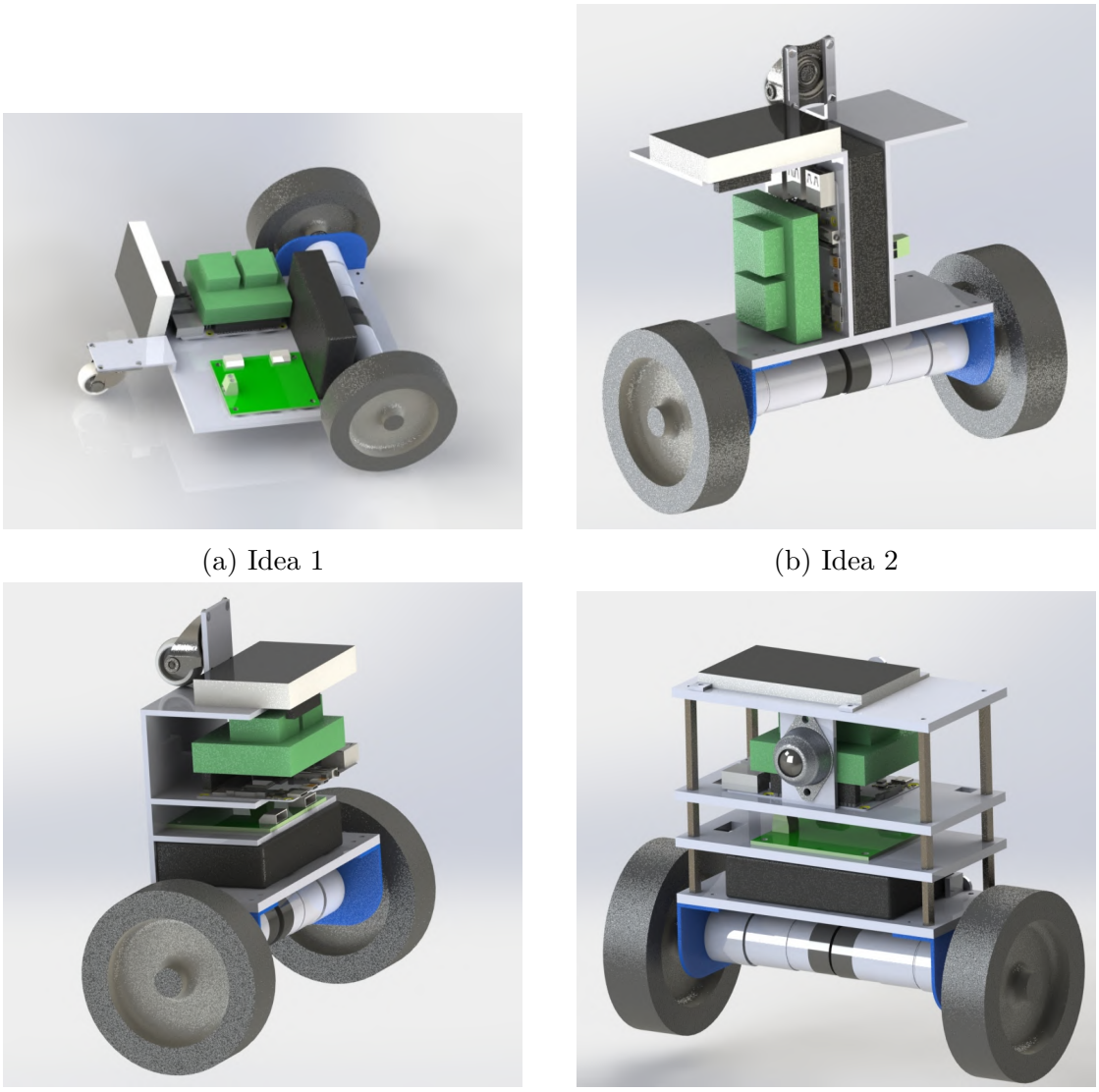


Figura 2.4: Alimentación de los componentes

2.3. Proceso de diseño

Contando con los requisitos y los componentes necesarios se barajaron 4 posibles configuraciones en una fase de diseño conceptual. Las 4, mostradas en la figura 2.5, están ordenadas por orden cronológico y muestran la importancia de que la batería este cerca de las ruedas para la estabilidad en modo balancín. Solo se reflejan los componentes de mayor tamaño y cuya fijación al chasis es más compleja.



(a) Idea 1

(b) Idea 2

(c) Idea 3

(d) Idea 4

Figura 2.5: 4 diseños conceptuales

El diseño seleccionado es el 4, ya que ofrece una serie de ventajas. Es el diseño más simple geométricamente, al estar formado por piezas rectangulares planas de tamaño y geometría similar. Esto hace más predecible su comportamiento y distribución de masa. Al contar con separadores metálicos estándar y secciones anchas en las placas se consigue un chasis robusto pero ligero, ya que las placas pueden ser vaciadas dejando únicamente nervios y el contorno para proteger los componentes.

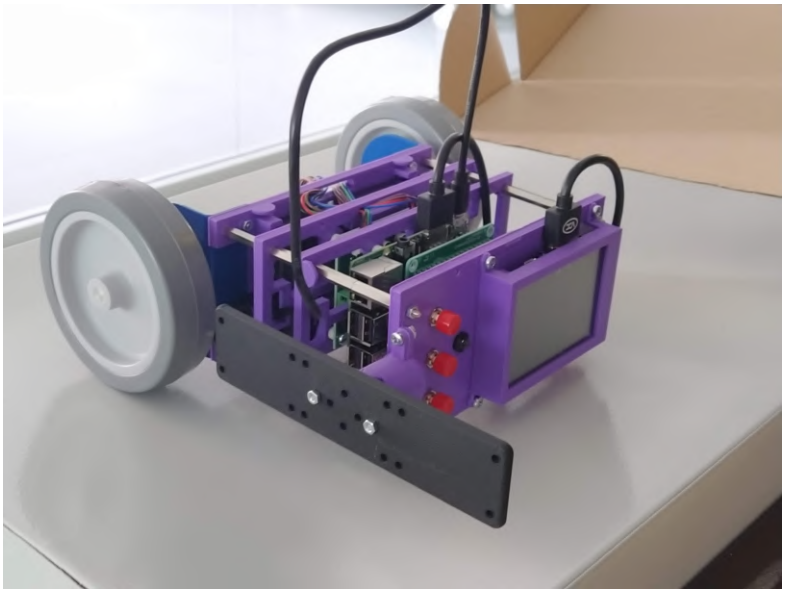


Figura 2.6: Primera versión del vehículo

Otra gran ventaja de la distribución en placas es la modularidad. Cada piso es independiente, por lo que el montaje ordenado es más sencillo y libre de recovecos o zonas inaccesibles. La modificación del diseño o de un componente hardware afectará solo a parte de las placas pero no obligará a desechar todas ellas, lo que ayuda a la sostenibilidad del diseño.

Cada placa está diseñada para ser fabricada con impresión 3D, con lo que se facilita conseguir repuestos en caso de que se rompa una placa contando con los recursos que tiene la universidad. Esto en combinación con la modularidad del diseño permite actualizar componentes hardware cuando sea preciso con medios tan básicos como un software CAD y acceso a una impresora 3D.

2.4. Primera versión del vehículo

Se puede apreciar en la figura 2.5d que este es un diseño preliminar, ya que integra los componentes considerando el conexionado entre ellos y determina la canalización de cables entre placas para conectarlos. También sustituye la rueda genérica por un modelo concreto de rueda esférica, más compacta y que aporta robustez al diseño.

A continuación se muestra el diseño dimensional de la primera iteración en la figura 2.6, desglosado en sus componentes:

2.4.1. Placa base

Dadas las dimensiones de los motores, esta placa condiciona el ancho del resto de placas. Su función es sujetar las placas metálicas en las que se fijan los motores sin que estos choquen entre sí. Se dejó una separación de 2.5 mm entre motores, lo que mantiene el ancho máximo del vehículo en 23 cm. También mantiene fija la Power Bank en una colocación centrada, permitiendo su extracción por un lateral quitando un tornillo pero sin la necesidad de desmontar el vehículo entero. Finalmente, canaliza los cables de los motores desde el driver MD25 y sujeta dos marcadores ópticos.

La distancia con la siguiente placa debe ser ligeramente superior al grosor de la Power Bank, que afortunadamente es de casi 20 mm, una distancia disponible en separadores estándar.

2.4.2. Placa 1

Este nivel contiene el driver MD25 y el conversor de nivel lógico, así como la canalización de los cables de los motores y toda la alimentación procedente de la Power Bank. También tiene dos agujeros para sujetar el bloque de navegación y dos salientes para colocar marcadores ópticos a un lado del vehículo. La rotura de simetría en la posición de los marcadores impide que el software motive capte su orientación de manera incorrecta, esto es, en espejo.

Dado que su superficie inferior sujeta la Power Bank, no puede haber ningún tornillo o pieza sobresaliendo en la parte central, por lo que no se usaron tuercas en la fijación del driver.

2.4.3. Placa 2 y rueda central

Sujeta tanto la Raspberry Pi como todos sus hats. Canaliza los cables de alimentación de la Raspberry y de comunicación I2C con el driver MD25. También presenta dos salientes para marcadores ópticos y una lámina vertical que sujeta la rueda de bola a la altura adecuada para que el vehículo esté en posición horizontal mientras rueda.

2.4.4. Placa 3 y marco de pantalla

Es la última placa y contiene toda la comunicación física con el usuario: los tres pulsadores para actuar sobre la máquina de estados, el LED RGB para indicar

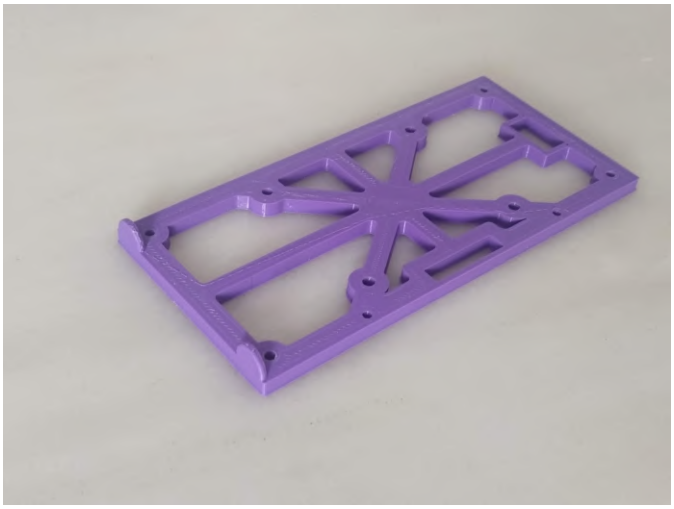


Figura 2.7: Primer prototipo de la Placa 1

el estado actual y la pantalla LCD mostrando el escritorio del sistema operativo. La pantalla se sujeta mediante la combinación de tres apoyos en zonas de su PCB sin componentes electrónicos con un marco que envuelve todo el contorno de la pantalla salvo los laterales que contienen los conectores microUSB y HDMI. Los demás componentes van enganchados en agujeros pasantes.

En un lateral tiene un enganche destinado al sensor de distancia a la pared, en otros dos hay salientes para marcadores ópticos y en el último hay una ranura para sujetar la lámina de la rueda de bola.

Finalmente, tiene dos agujeros adicionales para fijar el bloque de navegación.

2.5. Cambios realizados

Tras comprobar algunos fallos en la primera versión se hizo una primera iteración de correcciones a todas las placas menos a la placa base.

En la placa 1 se añadió altura en los agujeros que fijan el driver simulando unas arandelas, ya que los tornillos de menor longitud presentes en el taller atravesaban la placa totalmente y las patillas sobrantes de los componentes soldados del driver MD25 que no están soldados en superficie colisionaban con la placa 1.

Se hizo algo similar en la placa 2 por el motivo de las patillas sobrantes, y además se movió la placa completa hacia un lateral para evitar que el Shuttle Click colisionara con la lámina que sujeta la rueda.

Figura 2.8: Placa auxiliar para sujetar la rueda de bola

Dado que por la posición de impresión de esta placa y la anisotropía debida a esta técnica de fabricación esta lámina era frágil, se optó por dividir el diseño y hacer de esta lámina una pieza independiente (figura 2.8) que encaja entre las placas 2 y 3. Además se añadieron dos taladros para fijar un mikroBUS Shuttle que permita al vehículo navegar como un siguelineas con un conjunto de sensores de color.

En la placa 3 se quitó material del lateral del que vienen los cables de la pantalla, ya que estos tienen el conector a 90 grados de la dirección en la que viene el cable. El saliente para marcador óptico de ese lado fue movido al marco de la pantalla. Además, se movió la posición del LED RGB para acomodar otros mikroBUS Shuttle a ambos lados de la placa entre los botones y la pantalla.

En la figura 2.9 se muestra esta segunda iteración, ya totalmente funcional.

2.6. Bloque de navegación

Los requisitos para el bloque de navegación son:

- Debe ser compatible con otro vehículo del laboratorio (un dron en desarrollo).
- Debe integrar la posibilidad de montar diversos sensores: Cámaras Intel Realsense L515, D435 y T265, cámaras simple y estéreo para Raspberry Pi y LiDAR RPLIDAR A2M8.

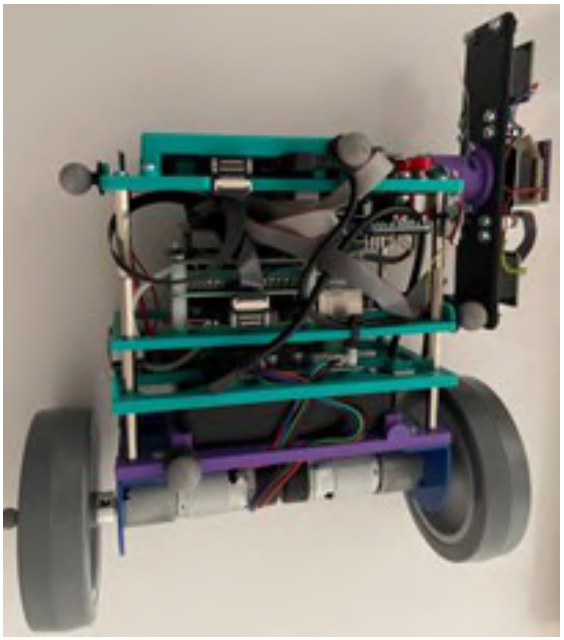


Figura 2.9: Segunda versión del vehículo

- Su ordenador SBC (*Single Board Computer*) debe poder integrar núcleos gráficos y ser compatible con sistemas operativos extendidos entre la comunidad software libre, como son Debian o Ubuntu.

Para la aplicación de este proyecto solo será necesaria la cámara Realsense L515, que consta tanto de un sensor óptico a color como de un sensor LiDAR de estado sólido, pudiendo formar conjuntamente mapas de puntos a color. Su consumo es de 3.5 W, su rango de medida entre los 25 cm y los 9 m con una apertura de 70 grados en el plano horizontal y 55 en el vertical y una resolución de 1024 X 768[3].

Como ordenador se utilizó la Raspberry Pi 4 B por la abundancia de experiencia y soporte de la comunidad que tiene, además de ser un proyecto activo y con futuro. Tiene la posibilidad de conectarse al Google Coral USB Accelerator, que le permite integrar Machine Learning. El modelo empleado tiene 4GB de memoria RAM, Integra USB de tercera generación, necesarios para comunicarse con las cámaras Realsense, y WiFi de 5GHz para comunicarse con un mayor ancho de banda, dando monitorizaciones con menor latencia.

Para alimentar ambos componentes se emplea un *UPS shield* (siglas de *Unin-*

[3]Intel Realsense. *LiDAR Camera L515*. 2021. URL: `https://www.intelrealsense.com/lidar-camera-l515/`.

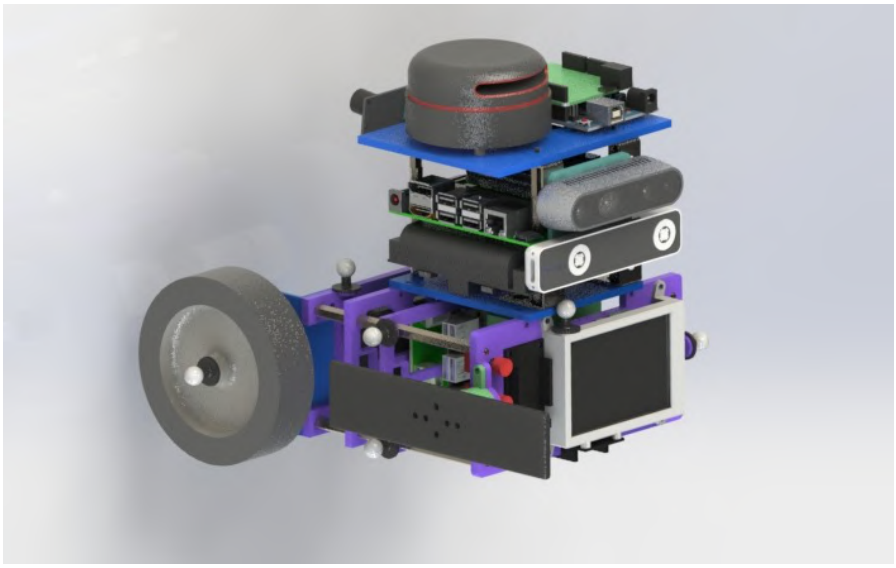


Figura 2.10: Diseño conceptual del bloque de navegación

terruptible Power Supply diseñado especificamente para Raspberry Pi y que suministra una carga de hasta 8 A a través de los pines GPIO. Tiene dos ranuras para baterías 18650, que con las baterías utilizadas da una capacidad de 19Wh, suficiente para alimentar ambas electrónicas durante 2 horas.

Para garantizar que pueda montarse tanto en el coche como en el dron, se diseñó una pieza intermedia entre el coche y el bloque de navegación. Esta se sujeta a los taladros presentes en las placas 1 y 3 para formar 4 agujeros en el mismo plano y a 79 mm el uno del otro, Adaptando el anclaje al formato que se muestra en la figura 2.11.

El diseño necesario para este proyecto es más simple que el de la figura 2.10. Consiste en una estructura que sujeta la Raspberry Pi 4 y en su lateral una placa con las distancias de fijación de las cámaras Realsense nombradas y cuatro fijaciones en sus esquinas que se atornillan a cuatro pequeñas escuadras metálicas. El resultado montado sobre el vehículo se muestra en la figura 2.12.

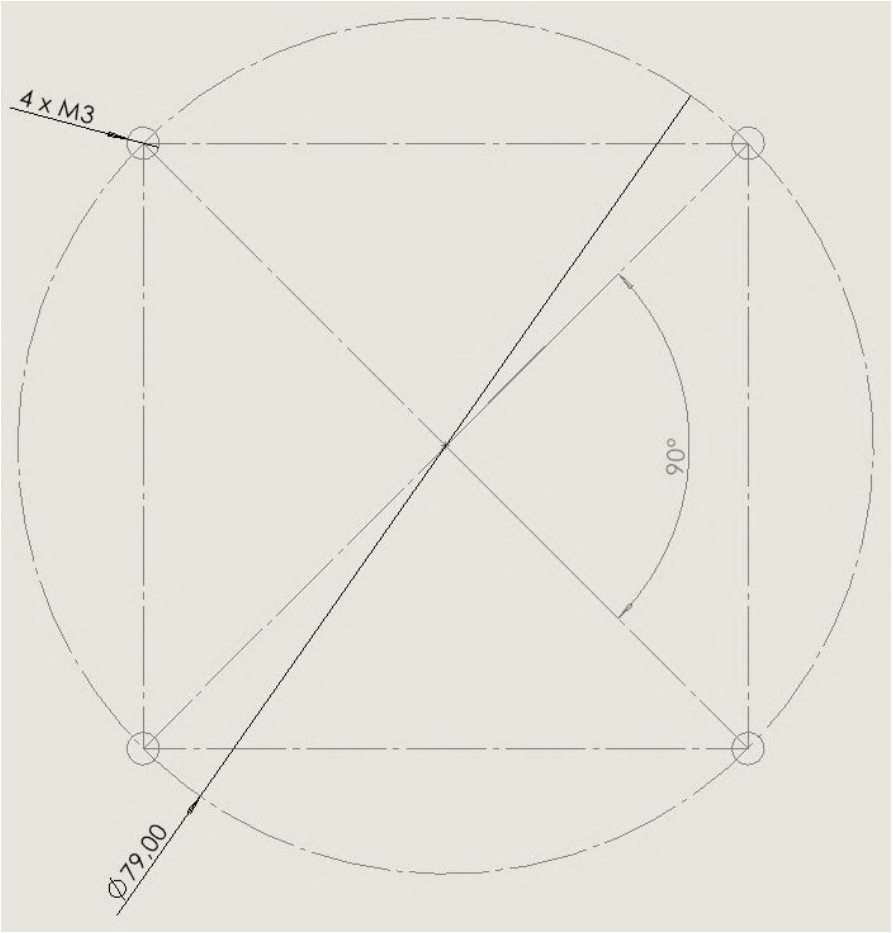


Figura 2.11: Anclaje estándar del bloque de navegación

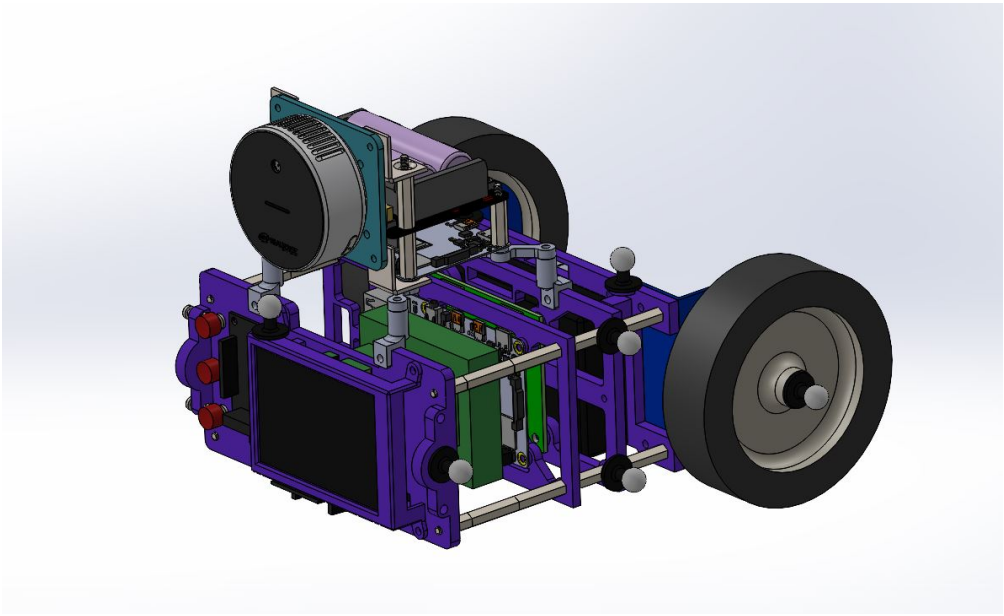


Figura 2.12: Diseño CAD del bloque de navegación

Capítulo 3

Implementación del software

3.1. Software en el vehículo

El vehículo es gobernado por la Raspberry Pi 3B+ ejecutando la versión publicada por MATLAB de Raspberry Pi OS (ver anexo .1), de la misma manera que lo hacía el vehículo de laboratorio anterior. Esto permite programar el sistema de control a través de MATLAB y Simulink, ambos lenguajes muy utilizados por los estudiantes de la universidad. Es muy destacable que Simulink contiene una librería con la que acceder a los pines de su GPIO, realizar comunicaciones por protocolos como son SPI, I2C, UART o TCP, utilizar los LEDs presentes en su placa base, adquirir imágenes desde su conector de cintas para cámara...Es por este motivo que esta combinación de hardware y software ha sido muy usada en los laboratorios de ICAI para controlar diversos robots (entre ellos todos los usados en asignaturas).

Además de las librerías hechas exclusivamente para Raspberry Pi, Simulink también consta de otras librerías de control, comunicación con ROS/ROS2, matemáticas y muchas más que son compatibles con su compilación en C/C++ para su ejecución en este y otros SBC de manera independiente, lo que es muy conveniente para aplicaciones de robótica móvil.

Fruto de sucesivos años de experiencia programando en los laboratorios de control distintos robots con MATLAB/Simulink, el profesor Juan Luis Zamora Macho ha desarrollado diferentes versiones de un código de control adaptadas a los distintos robots. Este código permite la regulación automática de actuadores con diferentes estrategias de control, como son PID (*Proporcional Integral Derivativo*) diseñado por margen de fase, margen de ganancia o definido por sus parámetros internos, realimentación de estado pudiendo aplicar Filtro de Kalman, o control

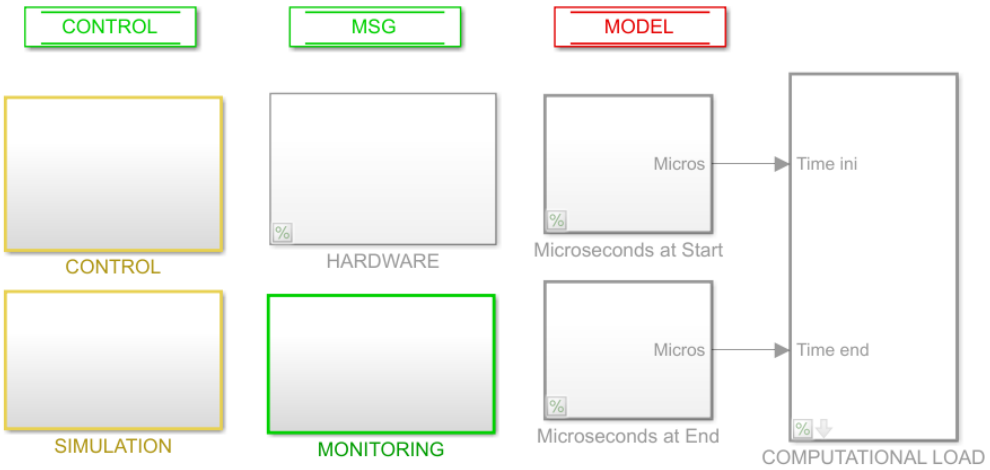


Figura 3.1: Diagrama de bloques CAR_CONTROL_SYSTEM

adaptativo, todo ello permitiendo adaptar el control a las características particulares de cada planta, posibilitando su simulación antes de implementarse y dando acceso a su posterior monitorización durante la ejecución.

El código de control está formado por dos diagramas de bloques de Simulink que trabajan de manera conjunta:

- CAR_CONTROL_SYSTEM (figura 3.1): Puede cargarse en el robot y comunicarse con los sensores y actuadores mediante los drivers presentes en el bloque HARDWARE o simular el comportamiento del robot y su hardware con el bloque SIMULATION. El bloque CONTROL se encarga de ejecutar el cálculo de órdenes a los actuadores según el tipo de control indicado y las referencias deseadas. El bloque COMMUNICATIONS gestiona la codificación y descodificación de los mensajes intercambiados con PC_CONTROL_STATION.
- PC_CONTROL_STATION (figura 3.2): Su bloque hardware se encarga exclusivamente de la comunicación TCP con CAR_CONTROL_SYSTEM, aunque la codificación y decodificación se lleva igualmente en COMMUNICATIONS. El bloque STATE MACHINE gestiona una máquina de estados con los siguientes estados:

  1. Espera a que el robot esté preparado tras recibir tipo, parámetros y referencias de control.

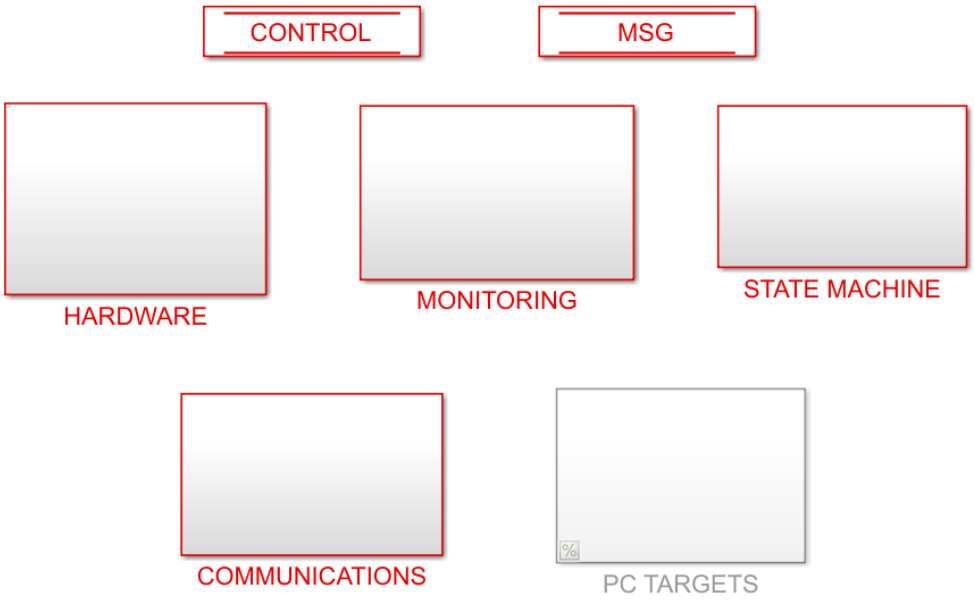


Figura 3.2: Diagrama de bloques PC_CONTROL_STATION

2. Calibra sus sensores hasta indicar que está listo para ejecutar el control.
3. Ejecuta el control hasta que se da la orden de parar.
4. Espera a recibir la orden de cargar nuevos datos de control.

Estos estados son cambiados mediante los tres pulsadores presentes en el robot. Por último, el bloque MONITORING muestra todos los datos de referencias, actuación y medidas, el estado actual en la máquina de estados y el tipo de control y referencia en uso.

Una descripción más detallada del funcionamiento de estos códigos se puede encontrar en el trabajo fin de grado de Adriana Marroquín Rodríguez [16].

3.1.1. Modelado de la planta

Para poder gobernar el vehículo mediante sus motores lo primero que se necesita hacer es modelar matemáticamente el vehículo. Esto se hace en forma de funciones de transferencia, unas funciones que usando la variable compleja s de Laplace relacionan la respuesta física de un sistema respecto a la variación de una variable de entrada. Para controlar el vehículo se usan la función que relaciona la

tensión común de los motores u_c (valor medio de ambas tensiones) con la velocidad de avance v y la función que relaciona la tensión diferencial u_d (tensión del motor izquierdo menos la del derecho) con la velocidad angular ω. El modelo matemático que relaciona las masas, inercias, resistencias e inductancias del motor, intensidades y pares para llegar a dichas funciones de transferencia se ha tomado del documento presente en el anexo .2.

Para obtener los parámetros específicos para este coche se actualizó la masa total a 2 kg en el fichero que define los parámetros del vehículo y se realizaron dos ensayos. El primero fue una referencia de velocidad de onda cuadrada, cuyos resultados (figura 3.3) luego fueron ajustados por mínimos cuadrados a un modelo simulado mediante el modo de ajuste 6 del código de obtención de parámetros del modelo, obteniendo la función de transferencia de avance 3.1.

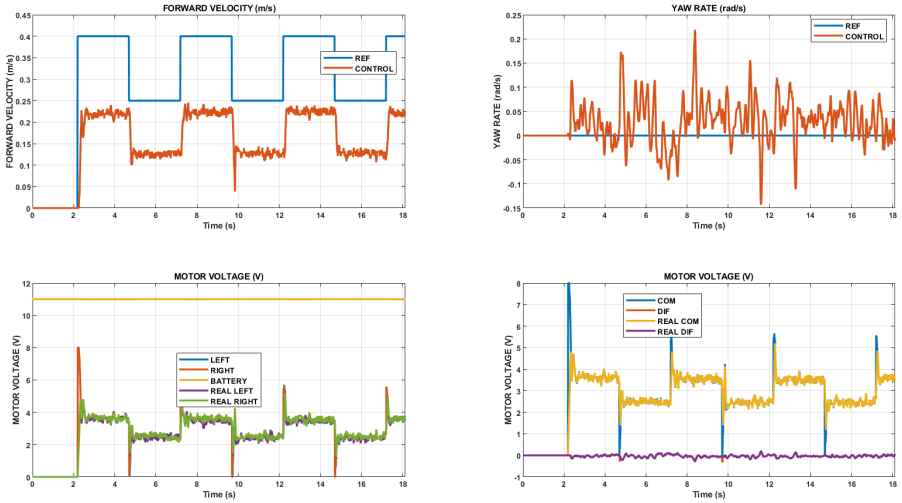


Figura 3.3: Resultados del ensayo de avance

$$\begin{aligned} \text{FORWARD\_VEL} &= \frac{53{,}97}{s^2 + 51{,}99s + 674{,}7} \\ &= \frac{0{,}079991}{(1 + 0{,}03705s)(1 + 0{,}04s)} \end{aligned} \tag{3.1}$$

El segundo ensayo fue un valor constante de velocidad de avance y una onda cuadrada simétrica, y sus resultados (figura 3.4) se usaron de la misma manera para

obtener los parámetros relacionados con la función de transferencia de velocidad de giro 3.2, y fueron guardados en el mismo fichero de configuración.

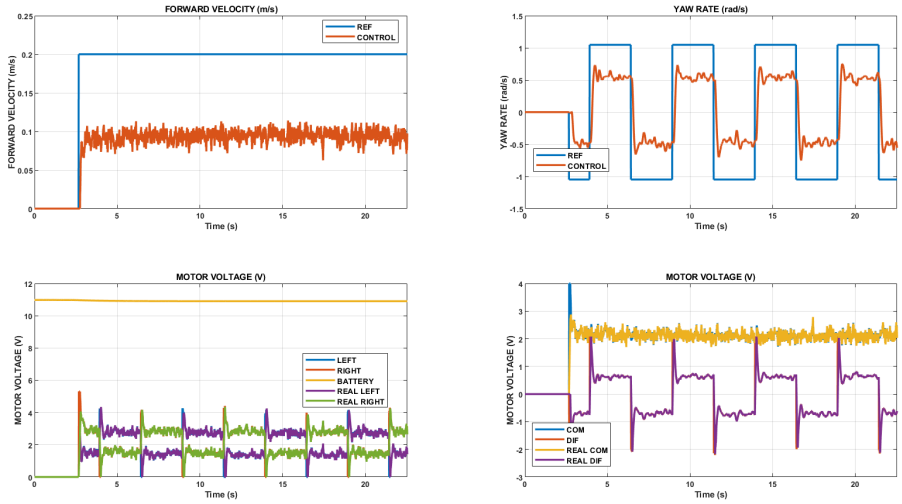
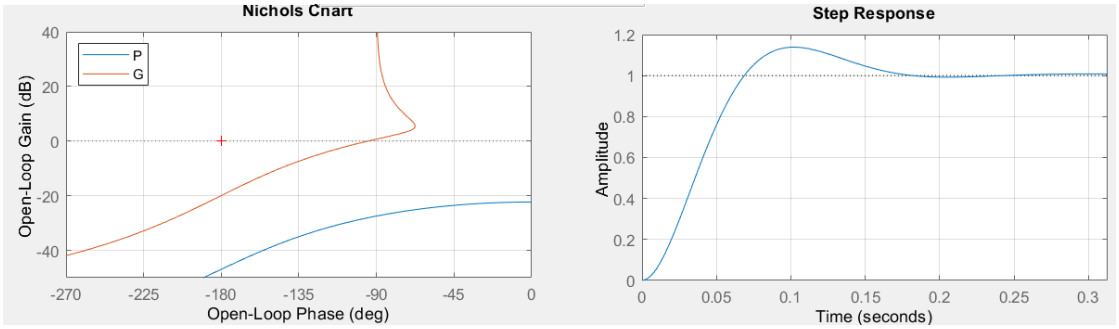


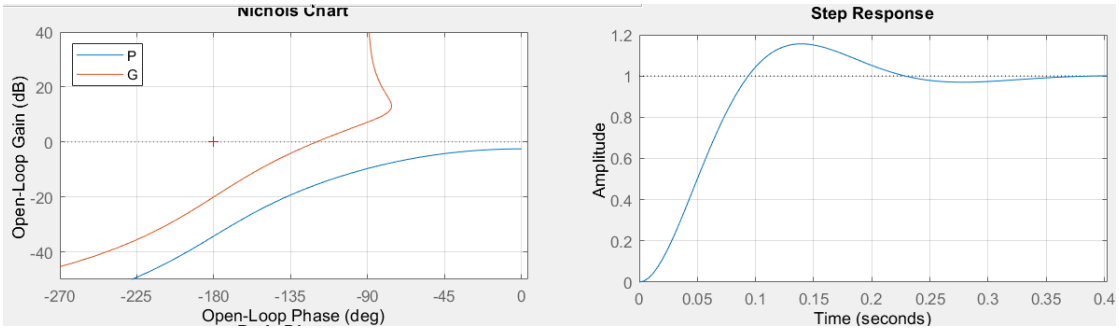
Figura 3.4: Resultados del ensayo de giro

$$\begin{aligned} \text{YAW\_RATE} &= \frac{122{,}6}{s^2 + 31{,}88s + 172} \\ &= \frac{0{,}71301}{(1 + 0{,}04s)(1 + 0{,}1454s)} \end{aligned} \tag{3.2}$$

A partir de estas dos funciones de transferencia se diseñaron por respuesta en frecuencia los controles de velocidad de avance y giro. Dado que en esta aplicación la rapidez no es crucial (incluso se intentará mantener una baja velocidad) y el error en el seguimiento de referencia tampoco tiene gran importancia mientras se estabilice en el entorno de las decenas de segundo que se intercambian la posición y orientación entre los ordenadores de control y navegación, se diseñaron dos controles PI (*Proporcional Integral*) con un sobrepaso de en torno al 15 %. El control PI tiene un diseño simple y un ajuste intuitivo, y cubre perfectamente las necesidades particulares de este proyecto.



(a) Control de avance



(b) Control de giro

Figura 3.5: Controles diseñados con la herramienta GUI

Los controles, mostrados en la figura 3.5 fueron diseñados con la herramienta del laboratorio de control *GUI_control_digital*. Tras comprobar que los controles no eran ideales se modificaron ligeramente los parámetros del control. El de velocidad tenía un componente integral muy lento, por lo que se subió su ganancia integral T_i, se bajó un poco la ganancia proporcional K y se ajustó la ponderación de referencia b. El de giro era inestable y también tenía un tiempo de alcance largo, por lo que se bajó la ganancia proporcional K y se aumentó la ganancia integral T_i. Los parámetros de control finales utilizados son $K = 18, T_i = 6, b = 1{,}6$ para el control de avance y $K = 2, Ti = 0{,}7, b = 1{,}4$ para el control de giro, obteniendo la respuesta mostrada en la figura 3.6.

3.2. Software en el bloque de navegación

La Raspberry Pi 4 utiliza el sistema operativo Ubuntu Server 20.04 LTS, ya que en el momento de dar forma al proyecto era la última versión del sistema operativo con soporte a largo plazo (con actualizaciones hasta 2025). Además de

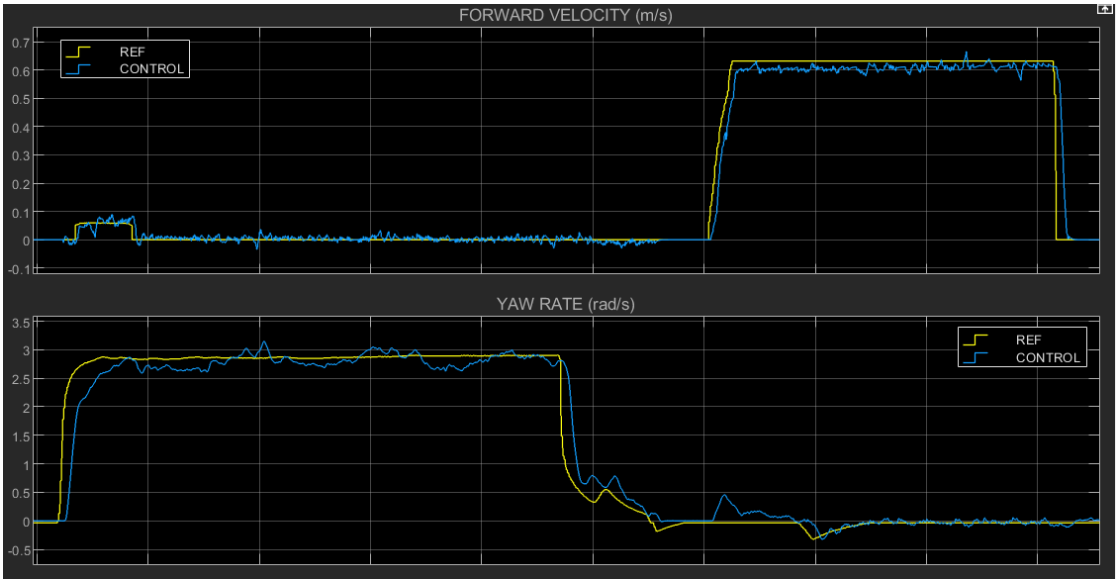


Figura 3.6: Ensayo de seguimiento de referencias de avance y giro

ser uno de los sistemas GNU Linux más extendidos, tiene una gran cantidad de soporte en ROS y la comunidad de software libre, por lo que muchas librerías y programas están precompilados para Ubuntu 20.04 y hacen más simple y rápida su instalación. Existe además una versión para Raspberry Pi (arm64) en su la herramienta Raspberry Pi imager, lo que facilita aún más la configuración inicial del sistema habilitando la comunicación ssh y WiFi sin siquiera necesitar conectarla a periféricos en el primer encendido.

Para comunicar los diferentes procesos llevados a cabo para la navegación y mapeado y utilizar drivers y librerías de SLAM de código abierto se instaló ROS2. La versión de ROS2 compatible con Ubuntu 20.04 es Foxy Fitzroy. La librería para adquirir los datos de distancias de la cámara LiDAR, llamada ROS Wrapper for Intel Realsense Devices (4.0.4), es compatible con Foxy y con el Intel Realsense SDK 2.0 (v2.50). Este último es compatible solo hasta Ubuntu 20.04, lo que refuerza la decisión de usar esta versión del sistema operativo.

Además del nodo para la cámara, se usa un nodo ros adicional llamado depthimage_to_laserscan. Este nodo extrae una fila de la matriz de medidas generada por la cámara LiDAR L515 y la publica en un `sensor_msgs/msg/LaserScan`. Este paso es importante ya que los algoritmos de SLAM escogidos no utilizan información tridimensional, sino bidimensional.

Se utilizó la librería nav2, que contiene todos los nodos necesarios para realizar navegación con un robot en el entorno ROS2. Concretamente se usan las librerías map_server (carga, guardado y publicación de mapas), amcl (localización en un

mapa mediante filtro de partículas), slam_toolbox (creación de mapas y SLAM) y lifecycle_manager (activación de los nodos), aunque en el futuro se podrían usar también el planificador de trayectorias, el seguidor de marcadores o la capacidad de recalcular trayectorias si encuentra obstáculos.

Para facilitar la puesta en marcha del sistema de navegación, se instaló adicionalmente Xfce, un entorno de escritorio ligero que permite visualizar las herramientas de depuración rqt_graph (muestra los nodos ros y sus conexiones), realsense viewer (muestra las medidas de sensores generadas por las cámaras Intel Realsense) y rviz2 (muestra el mapa y las posiciones de objetos, partículas y puntos detectados por el LiDAR en ese mapa).

La instalación de todo este software está descrita en el anexo .2.

Por último, para comunicar este SBC con el de control del vehículo se utiliza python 3.8, que viene de serie con Ubuntu 20.04, y su librería socket, que permite la comunicación con clientes y servidores TCP. Para realizar una comunicación más rápida entre ambas Raspberrys se creó una subred en la que el ordenador de control tiene la IP 192.168.0.123 y el ordenador de navegación tiene la 192.168.0.124. De esta manera se pueden comunicar para monitorización y modificación de archivos mediante WiFi, pero se asegura una transmisión por Ethernet entre ambas sin intermediarios (los envíos no llega al router).

La configuración de Ethernet se hizo escribiendo las líneas mostradas en el código 3.1 en la Raspberry Pi 3 (Raspberry Pi OS) en el fichero /etc/dhcpcd.conf, mientras en la Raspberry Pi 4 (Ubuntu) se usó el código 3.2 en el archivo /etc/netplan/50-cloud-init.yaml. Este procedimiento se tomó del TFG de Alejandro Ucelay [19, pag. 41].

Código 3.1: Archivo dhcpcd.conf para configuración de comunicación Ethernet

```
interface eth0
static ip_address=192.168.0.123/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
```

Código 3.2: Archivo 50-cloud-init.yaml para configuración de comunicación Ethernet

```
network:
    version: 2
    renderer: NetworkManager
    ethernets:
        eth0:
            dhcp4: no
            addresses: [192.168.0.124/24]
```

```
gateway4: 192.168.0.1
nameservers:
    addresses: [192.168.0.1]
```

Se verificó con la función ping que la información se transmitía en ambas direcciones.

3.3. Creación del mapa

El mapa es un elemento fundamental del proceso SLAM, ya que para localizar cualquier cosa necesitas un sistema de referencia que te permita saber dónde estas respecto a los demás objetos, de forma que la posición calculada por el sistema detectando esos objetos coincida con la esperada en el mundo real. Por ello, su creación precisa de dos informaciones fundamentales para garantizar que se elabore correctamente: medidas de distancia a las paredes/objetos e información de desplazamiento y rotación del robot mientras se mueve escaneando el entorno. Con esta combinación se relacionan los objetos detectados con el punto desde el que se detectaron.

Un sensor láser no puede distinguir lo que está detectando, solo indica la distancia a la que lo percibe. En la práctica la detección de paredes y objetos se realiza con nubes de puntos, puntos medidos simultáneamente que se repiten de forma cercana en las medidas sucesivas. El algoritmo SLAM realmente decide qué medidas son fiables para formar parte de estas nubes de puntos y extiende el número de ellas cuando llegan medidas de nuevos lugares, definiendo sus posiciones respecto a las partículas antiguas. Este análisis se puede hacer de modo asíncrono, incluyendo solo las partículas más cercanas (muy útil en mapas grandes para que la complejidad de la solución no supere las capacidades del ordenador), o de modo síncrono, incluyendo todas las medidas . En un principio se usará el modo síncrono por su mayor precisión y por el pequeño tamaño del espacio a cartografiar.

La manera de gestionar los sistemas de referencia, ampliamente empleada en ROS y necesaria para SLAM_toolbox, es la librería tf2. Esta librería permite definir sistemas de coordenadas relacionados con los diferentes eslabones de uno o varios robots, así como de un sistema de coordenadas global o de un mapa. Todo sistema de coordenadas está relacionado con un sistema de coordenadas padre, dando lugar a una estructura en arbol como la que podemos ver en la figura 3.7. Las transformaciones entre un sistema y otro pueden ser estáticas, como es la posicion de un sensor respecto al centro de masas de un robot, o dinámicas actualizables desde un nodo, como es el desplazamiento del robot en el espacio. La mayor fortaleza de esta librería es que a partir de las relaciones individuales padre-

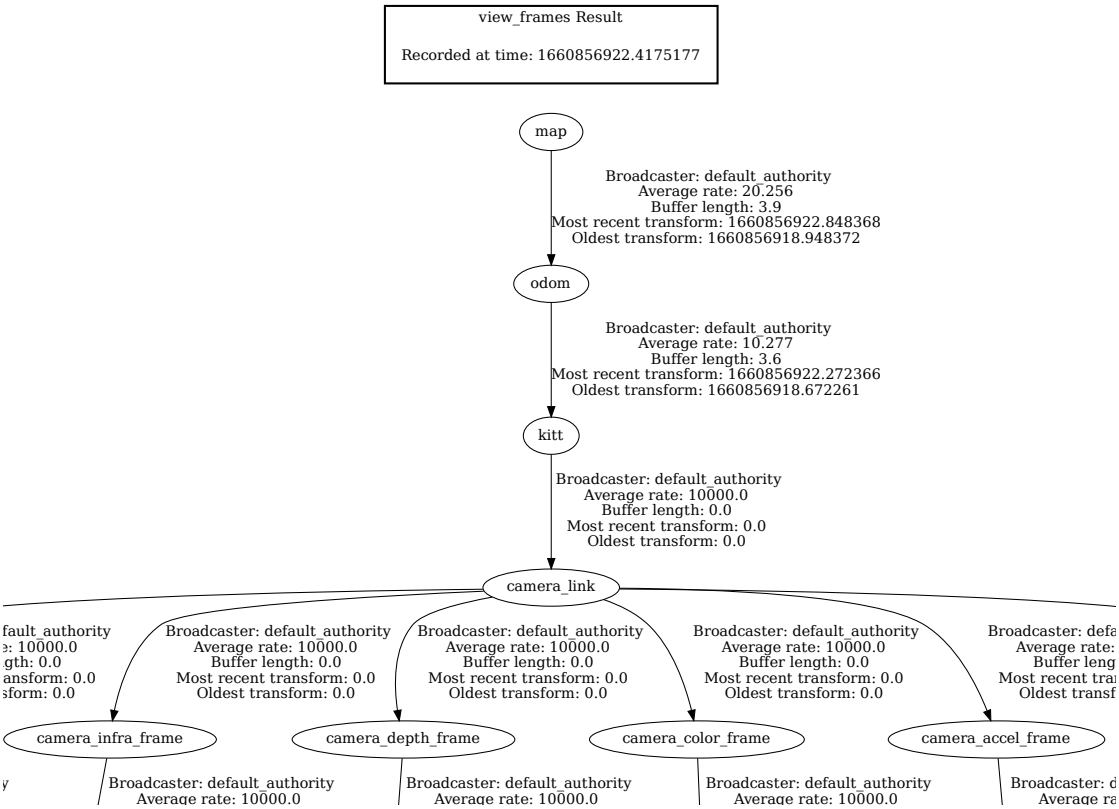


Figura 3.7: Sistemas coordenados usados para la generación del mapa

hijo calcula y publica bajo demanda la relación entre dos sistemas cualesquiera de entre los sistemas de coordenadas creados.

En este sistema robótico tenemos un sistema de referencia absoluto, llamado `map`, un sistema de referencia fijo que muestra dónde empezó el vehículo a escanear, llamado `odom`, un sistema situado en el eje teórico de giro de las ruedas en el punto medio entre las mismas, llamado `kitt`, y un sistema a una distancia fija de `kitt` correspondiente a la cámara LiDAR llamado `camera_frame`. Para aplicaciones de mayor precisión, el nodo `/camera` genera sistemas individuales para cada sensor, siendo el usado por el LiDAR el sistema `camera_depth_frame`.

La transformación estática entre `kitt` y `camera_depth_frame` se realiza con el código 2 del anexo .2.

La librería SLAM se encarga de crear el mapa con las medidas del LiDAR sabiendo que se han tomado desde la posición y orientación del sistema coordenado `camera_depth_frame` respecto al sistema de coordenadas `map`, pero cuando ve que la posición del vehículo no es coherente con las medidas recibidas modifica la posición del sistema coordenado `odom` respecto a `map`. Así se realiza simultaneamente la localización y el mapeo (*Simultaneous Localization And Mapping*).

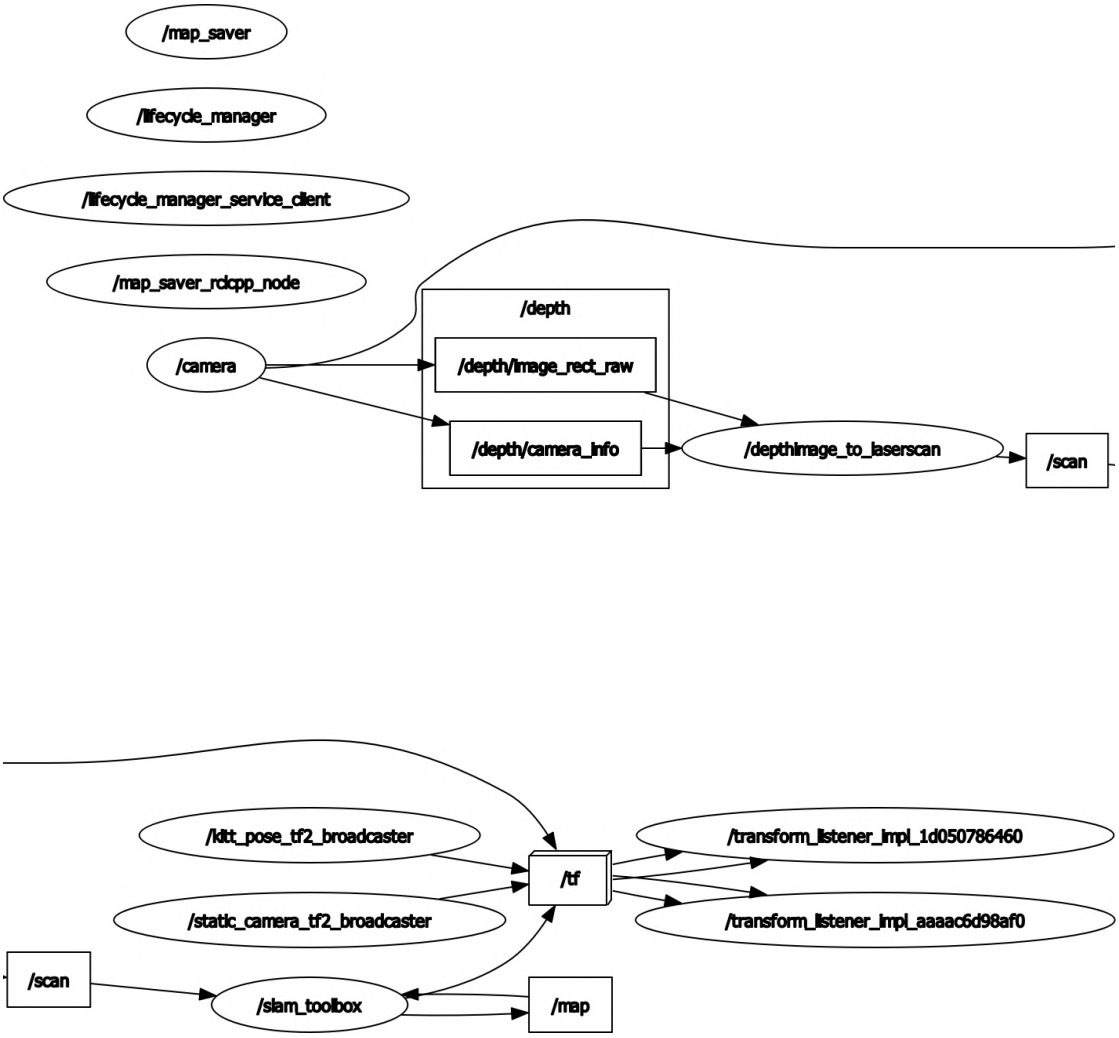


Figura 3.8: Nodos usados para crear el mapa

Como vemos en la figura 3.8, el nodo `/camera` publica los datos tridimensionales obtenidos con el LiDAR en el topic `/depth`. Este es transformado en un conjunto de puntos bidimensionales por el nodo `/depthimage_to_laserscan`, que lo envía a traves del topic `/scan` al nodo `/slam_toolbox` encargado de crear el mapa, que se publica en el nodo `/map`. Los nodos `/lifecycle_manager` y `/map_saver` trabajan conjuntamente. y este último se suscribe al topic `/map` para guardar el mapa cuando se ejecuta el servicio de guardado.

Las coordenadas y ángulos de Euler del vehículo son recibidas por TCP en tiempo real y decodificadas en el nodo `/kitt_pose_tf2_broadcaster` (código 1 del anexo .2) usando las librerías de Python *socket* y *struct* como se muestra en las siguientes lineas de código:

Recepción por TCP

```
realmsg = 0
while realmsg == 0:
    newdata = self.s.recv(13)
    if newdata and unpack('f', newdata[0:4])[0] != 0.0:
        data = newdata
        realmsg = 1

# Reshape TCP message
posx = unpack('f', data[0:4])[0]
posy = -1 * unpack('f', data[4:8])[0]
angz = -1 * unpack('f', data[8:12])[0]
```

Dado que el vehículo envía cada 100 ms su nueva posición y orientación y el resto del tiempo envía el mensaje 0.0 0.0 0.0, este código se encarga de filtrar los mensajes vacíos y solo publicar la información útil en forma de transformación tf entre `odom` y `kitt` y en el topic de depuración `/kitt_pose`. Además, en las últimas lineas se decodifica el mensaje recibido de 4 bytes para obtener de vuelta valores decimales y con signo (clase *float* en Python). Tanto esta transformación como la estática de la cámara y la del SLAM también se ven reflejadas en la figura 3.8 como publicación al topic `/tf`, que es accedido por nodos auxiliares creados por los nodos que leen esas poses. Ambos códigos se basan en los tutoriales oficiales de la librería tf2 [24] y de ROS2 Foxy [25].

Para crear el mapa se ejecutan los nodos mencionados, y mediante la emisora de radiocontrol se mueve el vehículo por la habitación a cartografiar. Una vez completado el mapa, se llama al servicio de guardado del mapa perteneciente al nodo `/map_saver` con el comando: `ros2 service call /map_saver/save_map nav2_msgs/srv/SaveMap "{map_topic: map, map_url: /home/pi/my_map, image_format: pgm, map_mode: trinary, free_thresh: 0.25, occupied_thresh: 0.65}"`, que guarda el mapa en un archivo de parámetros .yaml y un archivo de datos del mapa .pgm. Este último archivo es una imagen como la de la figura 3.9 con una relación $1\,pixel = 0,05\,metros$. Es preciso crear antes ambos archivos, y rellenar el archivo .yaml (con los valores deseados y que encajen con la llamada al servicio de guardado del mapa) como se muestra en el siguiente código :

Código 3.3: Archivo .yaml con parámetros del mapa

```
image: /home/pi/my_map.pgm
mode: trinary
resolution: 0.05
origin: [-0.148, -3.94, 0]
negate: 0
```

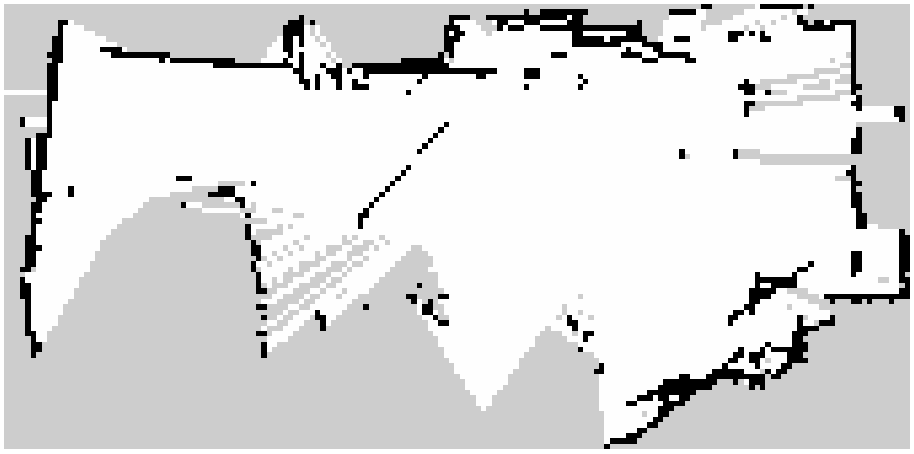


Figura 3.9: Primer mapa realizado por el vehículo

```
occupied_thresh: 0.65
free_thresh: 0.25
```

Como se puede ver, el mapa con los valores por defecto quedaba bastante sucio, con dobles paredes y puntos erroneos generados en medio del mapa. Durante la construcción del mapa se podía ver cómo la orientación del vehículo se iba distanciando de la orientación recibida, y una vez creadas las partículas erroneas el algoritmo SLAM recalculaba la posición del sistema coordenado odom erroneamente y acumulaba más fallos en el mapa.

Para mejorar la cálidad del mapa se cambiaron los siguientes parámetros:

- position_covariance_scale: Representa la confianza que tiene el sistema en las posiciones aportadas mediante el topic /odom. Dado que sabemos que acumulan error, se cambió el valor de 1.0 a 2.0.
- yaw_covariance_scale: Es similar al parámetro anterior, pero con la orientación del robot. También se ha cambiado de 1.0 a 2.0.
- tf_buffer_duration: Tiempo que se almacenan los sistemas de coordenadas para procesar los datos. Se aumentó de 30.0 a 60.0.
- minimum_travel_distance: Distancia mínima que avanza el robot antes de que se vuelvan a tomar nuevas medidas. se disminuyó de 0.5 a 0.25.

Con estos cambios se espera que se tomen más medidas y se use más el emplazamiento probabilístico de las medidas que sus posiciones geométricas respecto al sistema de coordenadas del robot. El nuevo mapa resultó ser mejor, tal y como se ve en la figura 3.10.

Figura 3.10: Mapa con diferentes parámetros

Figura 3.11: Mapa con SLAM asíncrono

Por último, se probó a usar un nodo SLAM asíncrono, y se verificó que la calidad del mapa (figura 3.11) para los mismos parámetros es peor que con el nodo síncrono. La localización del algoritmo SLAM no es tan buena, y los errores en la pose crean falsos puntos que contaminan y estropean el mapa.

Capítulo 4

Conclusiones

4.1. Revisión de objetivos

Como se ha visto, este trabajo ha dado como resultado un hardware, un vehículo, con bastantes capacidades, algunas de ellas demostradas en este mismo proyecto. Sin embargo, por contratiempos debidos a cambios de hardware y a la resolución de problemas y aprendizaje de recursos no esperados, la planificación se vio desplazada, por lo que se hace el siguiente análisis de la alineación con los objetivos iniciales:

1. Objetivo 1: Se ha diseñado ad hoc un vehículo compatible con todas las posibilidades de los vehículos anteriormente usados en los laboratorios y muchas más posibilidades dada su integración con otros muchos sensores. El vehículo ha demostrado ser controlable tal y como se ha hecho con el diseño de controles PI. Una muestra clara de ambas cosas es el uso actual del vehículo en la asignatura de regulación automática, control digital y control avanzado. Además, se ha logrado una integración entre MATLAB y ROS2 usando códigos externos de software libre, lo que ha hecho posible la creación del mapa mediante SLAM. Objetivo logrado.

2. Objetivo 2: Como se ha visto en la sección 3.3, el robot no solo es capaz de crear un mapa, sino que mediante la coordinación de los códigos en MATLAB y ROS se ha construido y guardado para su uso posterior un mapa en un formato habitual en esta y otras librerías ROS. Objetivo logrado.

3. Objetivo 3: Dado que cuando se logró la creación del mapa, necesario para localizarse y posteriormente enviar la localización al ordenador de monitorización, la universidad y sus laboratorios estaban cerrados por vacaciones,

no se pudo realizar la verificación de las medidas calculadas con algoritmos mediante el sistema de cámaras Optitrack como se planificó. No obstante, se realizó la calibración del sistema para obtener una precisión inferior al milímetro, se logró la adquisición de la posición y orientación del vehículo en el software y se comunicó exitosamente al vehículo estos datos mientras se conducía con el mando de radiocontrol, por lo que el simple acceso al laboratorio mostraría los resultados para su análisis tras un simple ajuste en los sistemas de coordenadas del software Motive. Objetivo avanzado.

4. Objetivo 4: Por falta de tiempo no se pudo configurar el planificador de trayectorias ni la comunicación de las posiciones al ordenador de control, con lo que no se realizó una navegación autónoma punto a punto. Objetivo no logrado.

4.2. Trabajos futuros

Dentro de lo que se ha logrado realizar en este trabajo, la principal mejora que se puede hacer es en la calidad de las posiciones y orientación del robot generadas en el SBC de control. Su cálculo, mostrado en el código 4 del anexo .2, se basa en la integración de los datos de velocidad creados con las medidas sin filtrar de los encoders. Al integrar estas medidas, se introduce un error permanente (offset) que perjudica mucho la adquisición de un mapa de calidad. Esta situación mejoraría usando un EKF en el cálculo de estas variables, que podría recibir tanto las medidas de los encoders como de la IMU presente en el vehículo.

La comunicación entre ambos SBC ha demostrado ser eficiente mediante Ethernet, y el hecho de comunicar el "motor" con el "cerebro" ofrece muchas posibilidades. Se podría informar de si se ha llegado a un puesto de carga inalámbrica, dar una realimentación en las posiciones calculadas en el bloque de navegación, incluso aplicar arboles de comportamiento y navegación con detección de obstáculos y enviar directamente las referencias desde la Raspberry Pi 4. Por ello, sería interesante crear una codificación de mensajes TCP similar a la realizada entre los códigos CAR_CONTROL_SYSTEM y PC_CONTROL_STATION que permita reutilizar el mismo firmware del vehículo para todas estas aplicaciones.

Aunque solo se hayan usado dos herramientas de la librería Navigation 2, podrían usarse otras también muy interesantes y prácticas, como nav2 smoother para depurar las trayectorias y hacerlas más suaves, o nav2 planner para crear trayectorias punto a punto pudiendo trazar rutas alternativas sobre la marcha para esquivar obstáculos.

La intención al dotar de más capacidad de cálculo al bloque de navegación

que al de control es extender los usos que se le puedan dar al vehículo, lo que incluye otras formas de localización, análisis de imagen, sincronización de robots colaborativos e incluso aplicación en otros vehículos o robots. para poder realizar tareas con Inteligencia Artificial o Aprendizaje Profundo sería preciso integrar en el diseño el Google Coral Accelerator, compatible con Raspberry Pi.

Finalmente, el vehículo en sí mismo sin el SBC de navegación incluye dos bloques seguidores de pared y nuevas bahías de sensores MikroE, que pueden ser usadas para resolver laberintos siguiendo paredes, programar robot siguelineas con detección de color, integrar en su posibilidad de cálculo de trayectorias datos de localización GPS, detección frontal de obstáculos mediante sensores de ultrasonidos o LiDAR... Las posibilidades están únicamente limitadas por la creatividad del usuario.

Bibliografía

[1] Francisco Coll Morales. "Ludismo". En: *Economipedia* (8 de sep. de 2020). URL: https://economipedia.com/definiciones/ludismo.html#:~:text=El%5C%20ludismo%5C%20fue%5C%20un%5C%20movimiento,los%5C%20a%5C%C3%5C%B1os%5C%201811%5C%20y%5C%201816..

[2] Oliver Hitchens. "3D-printed rocket engines: The technology driving the private sector space race". En: *Space* (29 de sep. de 2021). URL: https://www.space.com/3d-printed-rocket-engines-private-space-technology.

[3] COBOD International. *World leader in 3D construction printing*. URL: https://cobod.com/.

[4] NASA. *Mars 2020 Perseverance Rover*. 2020. URL: https://mars.nasa.gov/mars2020/ (visitado 21-08-2022).

[5] Brianna Wessling. "A decade after acquiring Kiva, Amazon unveils its first AMR". En: *The Robot Report* (22 de jun. de 2022). URL: https://www.therobotreport.com/a-decade-after-acquiring-kiva-amazon-unveils-its-first-amr/.

[6] Arrival. *Making the Microfactory*. 22 de mar. de 2021. URL: https://www.youtube.com/embed/_X9ge_W_ZPA.

[7] Mike Oitzman. "What's the difference between an AMR and an AGV?" En: *Mobile Robot Guide* (6 de ago. de 2021). URL: https://mobilerobotguide.com/2021/08/06/whats-the-difference-between-an-amr-and-an-agv/.

[8] Alfredo Pastor Tella. *AGV Navigation: Methods, Comparison, Pros and Cons - Illustrated Guide*. URL: https://www.agvnetwork.com/types-of-navigation-systems-automated-guided-vehicles.

[9] Hugh Durrant-Whyte y col. "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms". En: *IEEE Robotics & Automation Magazine* (2006).

[10] *Solid-State LiDAR based-SLAM: A Concise Review and Application*. International Conference on Big Data and Smart Computing (BigComp) (Jeju Island, Korea). IEEE, 2021, págs. 302-305.

[11] Alexey Merzlyakov y Steven Macenski. "A Comparison of Modern General-Purpose Visual SLAM Approaches". En: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

[12] C. Cadena y col. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". En: *IEEE Transactions on Robotics* (2016), págs. 1309-1332.

[13] *Using SLAM-based Handheld Laser Scanning to Gain Information on Difficult-to-access Areas for Use in Maintenance Model*. International Symposium on Automation and Robotics in Construction (Waterloo). IAARC Publications, 2017, págs. 1-6.

[14] Dassault Systemes. *Aplicaciones y servicios de la marca*. URL: `https://www.3ds.com/es/productos-y-servicios/` (visitado 01-07-2020).

[15] Ultimaker. *Ultimaker web page*. URL: `https://ultimaker.com/es/` (visitado 01-07-2020).

[16] Adriana Marroquín Rodríguez. "Navegación de Robots Móviles en Entornos Dotados de Sistemas de Localización Externos". Trabajo fin de grado. ICAI - Universidad pontificia Comillas, jul. de 2021.

[17] LaTeX. *LaTeX – A document preparation system*. URL: `https://www.latex-project.org/` (visitado 01-07-2020).

[18] Overleaf. *El editor de LaTeX fácil de usar, online y colaborativo*. URL: `https://es.overleaf.com/` (visitado 01-07-2020).

[19] Alejandro Ucelay Jiménez. "Navegación autónoma de un vehículo terrestre usando marcadores fiduciales". Trabajo fin de grado. ICAI - Universidad pontificia Comillas, jul. de 2022.

[20] Diego Cubillo Llanes. "Diseño mecánico de un dron para inventariado automático de almacenes". Trabajo fin de grado. ICAI - Universidad pontificia Comillas, jul. de 2020.

[21] Gonzalo Calvo Mosquera. "Navegación autónoma de un cuadricóptero basado en un sistema de cámara IR externas". Trabajo fin de grado. ICAI - Universidad pontificia Comillas, jul. de 2022.

[22] Robot electronics. *MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive. I2C mode documentation*. URL: `https://www.robot-electronics.co.uk/htm/md25tech.htm` (visitado 14-08-2022).

[23] Intel Realsense. *LiDAR Camera L515*. 2021. URL: https://www.intelrealsense.com/lidar-camera-l515/.

[24] Open Robotics. *Writing a broadcaster (Python)*. URL: https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Broadcaster-Py.html (visitado 19-08-2022).

[25] Open Robotics. *Writing a simple publisher and subscriber (Python)*. URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html (visitado 19-08-2022).

[26] Intel RealSense. *NVIDIA Jetson installation*. 2021. URL: https://dev.intelrealsense.com/docs/nvidia-jetson-tx2-installation (visitado 19-08-2022).

Anexo A: Instalación del software

.1. Raspberry Pi 3B+

Este ordenador va a ejecutar únicamente código proveniente de Matlab/Simulink, por lo que es fundamental que funcionen correctamente de manera conjunta. Para ello se usa la propia herramienta de Matlab para cargar el sistema operativo. En la pestaña HOME → ENVIRONMENT → Add-Ons → Manage Add-Ons se accede a una ventana como la mostrada en la figura 1. En el lateral derecho se hace clic en el símbolo con un engranaje gris.

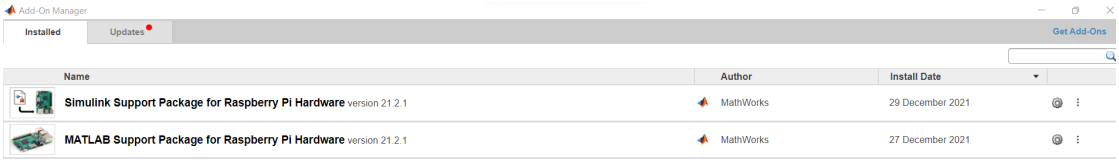


Figura 1: Ventana del gestor de Add-Ons

Se selecciona el modelo de Raspberry Pi a usar y se clica en Next. Se selecciona la opción *Setup hardware with MathWorks Raspbian image*, tras lo que se muestran las imágenes del sistema operativo disponibles. Se descarga la imagen siguiendo el enlace aportado la versión estándar (más simple) si hay otras y se valida el archivo descargado en la siguiente ventana. A continuación se puede configurar la conexión a internet o el método para que se comunique Matlab con el SBC durante la configuración del sistema operativo.

Se inserta la tarjeta MicroSD en el ordenador con el adaptador disponible, se selecciona el directorio asignado por el ordenador a esa memoria y se escribe el sistema en la tarjeta. Cuando termine el proceso, se conecta la Raspberry Pi a una fuente de alimentación. Matlab intentará comunicarse con ella, y haciendo clic en Test Connection verificará que puedan conectarse correctamente para poder cargar códigos y recibir realimentación de la ejecución de estos.

Tras esto, es necesario cargar el firmware de control. Para ello se ejecuta el códi-

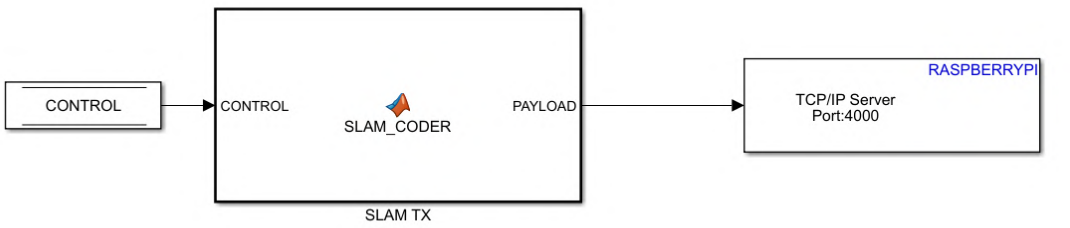


Figura 2: Bloque de envío de posiciones por TCP

go *CAR_PROJECT/ CONFIGURATION/ CONFIG_CAR.m* con $RUN\_MODE = 4$. Se abrirá un diagrama de bloques de Simulink llamado *CAR_CONTROL_SYSTEM*, en el que es imprescindible que el bloque HARDWARE → COMMUNICATIONS → TCP/IP SLAM esté descomentado. Este bloque (figura 2) está hecho para este proyecto en particular, aunque puede usarse para cualquiera que envíe coordenadas por TCP desde el vehículo, y ejecuta el código 4 del anexo .2. Tras verificar que en Hardware Settings → Hardware Implementation → Hardware board settings → Target hardware resources → Board parameters tanto la ip como el usuario y contraseña sean los correctos para esa placa, se hace clic en Build, Deploy & Start. El proceso de carga lleva varios minutos, y dependiendo de la calidad de la red puede llegar a prolongarse más de 15 minutos.

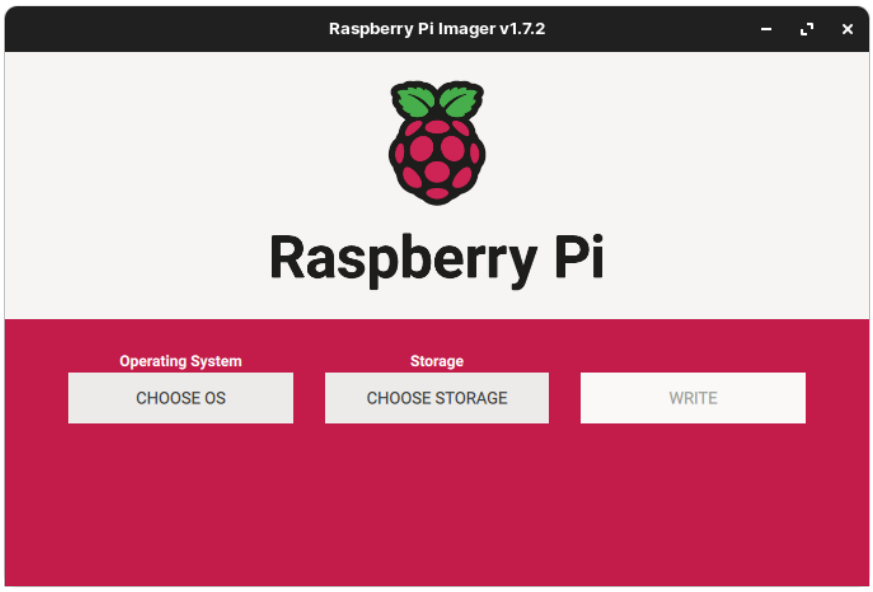


Figura 3: Raspberry Pi Imager

.2. Raspberry Pi 4B 4GB

Para cargar el sistema operativo en la tarjeta MicroSD se instaló el software Raspberry Pi Imager en el siguiente enlace:

`https://www.raspberrypi.com/software/`

Una vez instalado se ejecutó y se enchufó la tarjeta MicroSD al ordenador con un adaptador. En Operating system (ver figura 3) se seleccionó CHOOSE OS → Other general-purpose OS → Ubuntu → Ubuntu Server 20.04.4 LTS (versión de 64-bit) y se seleccionó en Storage la memoria MicroSD. Adicionalmente se clicó el símbolo del engranaje en la zona inferior derecha, y en la ventana que se abre se habilitó la comunicación ssh y se establecieron el nombre de usuario, la contraseña, el SSID del WiFi y su contraseña. Tras esto se pulsó en SAVE y en la ventana inicial en WRITE.

A continuación se instaló ROS2 Foxy Fitzroy siguiendo las instrucciones habituales mostradas en su tutorial oficial y escogiendo la versión Desktop Install, que incluye herramientas de depuración como rqt_graph o rviz2. Enlace al tutorial:

`https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html#install-ros-2-packages`

Para ejecutar comandos ros cada vez que se abre una terminal se ejecutó el comando:

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

Para comprobar la recepción de imagen desde la cámara Intel Realsense y facilitar otras tareas se instaló el entorno de escritorio XFCE Ejecutando los siguientes comandos:

```
sudo apt update && sudo apt upgrade -y
sudo apt install xubuntu-desktop
```

Tras esto se instaló el Intel Realsense SDK 2.0 (versión v2.50.0). Se descargó la librería de github mediante el comando:

```
wget https://github.com/IntelRealSense/librealsense
    /archive/master.zip
```

Tras esto se descomprimió el archivo, se accedió a su directorio *scripts* y se ejecutó el fichero *libuvc_installation.sh*. Este método de instalación, extraído de [26] permite compilar el SDK (siglas de *Software Development Kit*) con el backend RSUSB, ya que sin este se tuvieron problemas en la ejecución del software Realsense.

Para comprobar que la instalación se hubiese realizado correctamente se conectó la cámara Intel RealSense L515 con un cable USB 3.1 Gen1 a uno de los dos puertos USB 3 de la Raspberry Pi 4 y se ejecutó la herramienta de visualización de Realsense mediante el comando:

```
realsense-viewer
```

La ventana con la recepción de imagen a color y de profundidad se muestra en la figura 4.

Para instalar los nodos ros que transmiten los datos de la cámara LiDAR primero se instaló colcon, la utilidad oficial de ros2 para compilar librerías ros, y eProsima Fast DDS mediante los comandos:

```
sudo apt install python3-colcon-common-extensions
sudo apt install ros-foxy-rmw-fastrtps-cpp
export RMW_IMPLEMENTATION=rmw_fastrtps_cpp
```

A continuación, se siguieron las instrucciones de instalación mostradas en su repositorio github:

`https://github.com/IntelRealSense/realsense-ros/tree/4.0.2`

Se cambió $ROS_DISTRO por *foxy* en el comando anterior a ejecutar el comando *colcon build*. En el mismo workspace creado para la librería anterior se copió el contenido del repositorio de depthimage_to_laserscan y se ejecutó de nuevo *colcon build*. Para copiar el repositorio se ejecutaron los siguientes comandos:

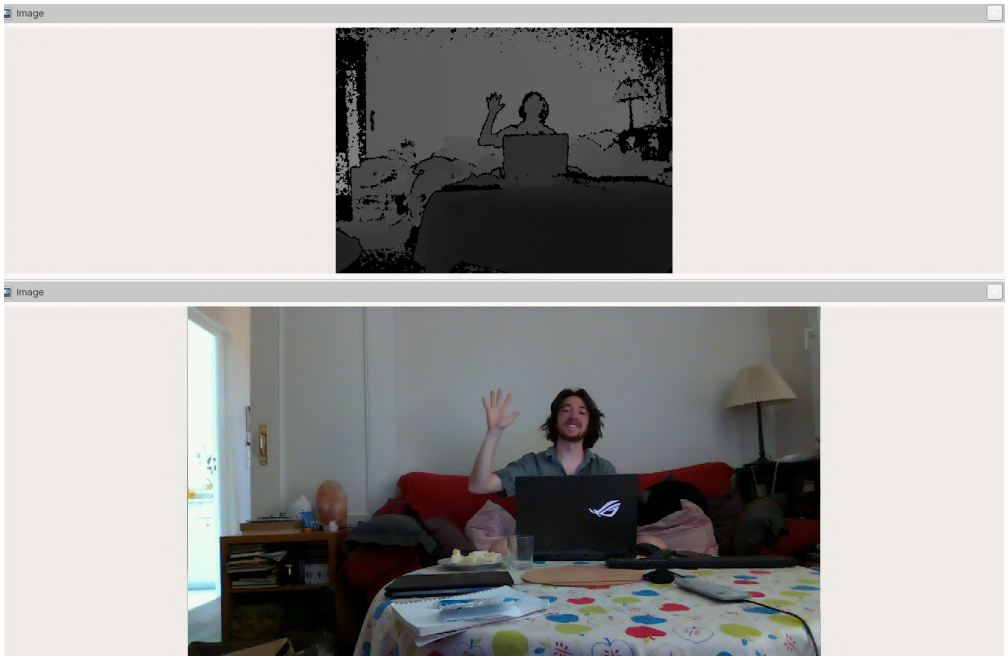


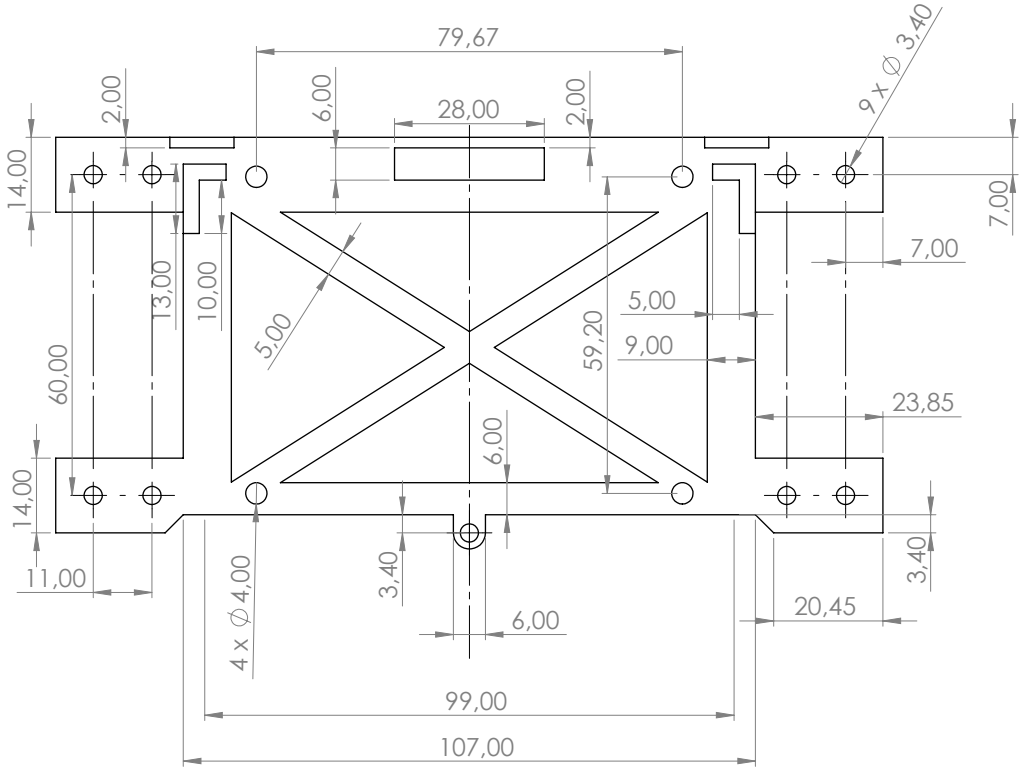
Figura 4: Ventana del software Realsense Viewer

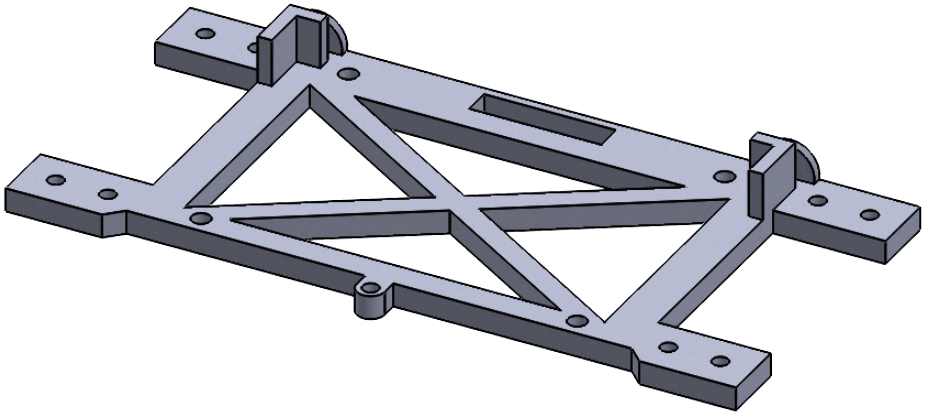
```
cd ~/ros2_ws/src
git clone -b ros2 https://github.com/ros-perception
    /depthimage_to_laserscan.git
```

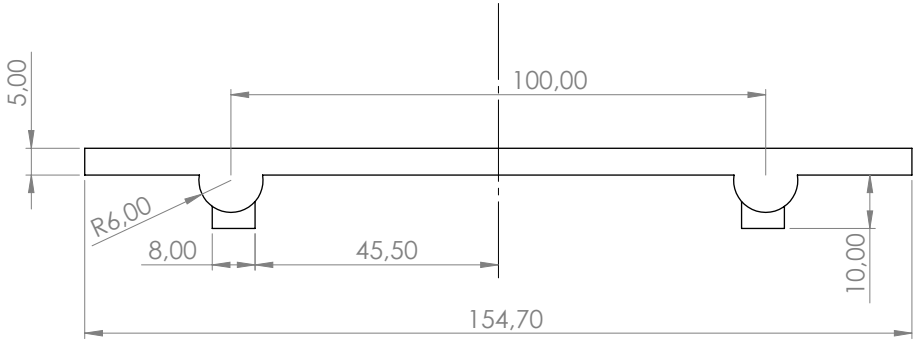
Finalmente, se instaló la librería ros Navigation 2 mediante los siguientes comandos:

```
sudo apt install ros-foxy-navigation2
sudo apt install ros-foxy-nav2-bringup
```

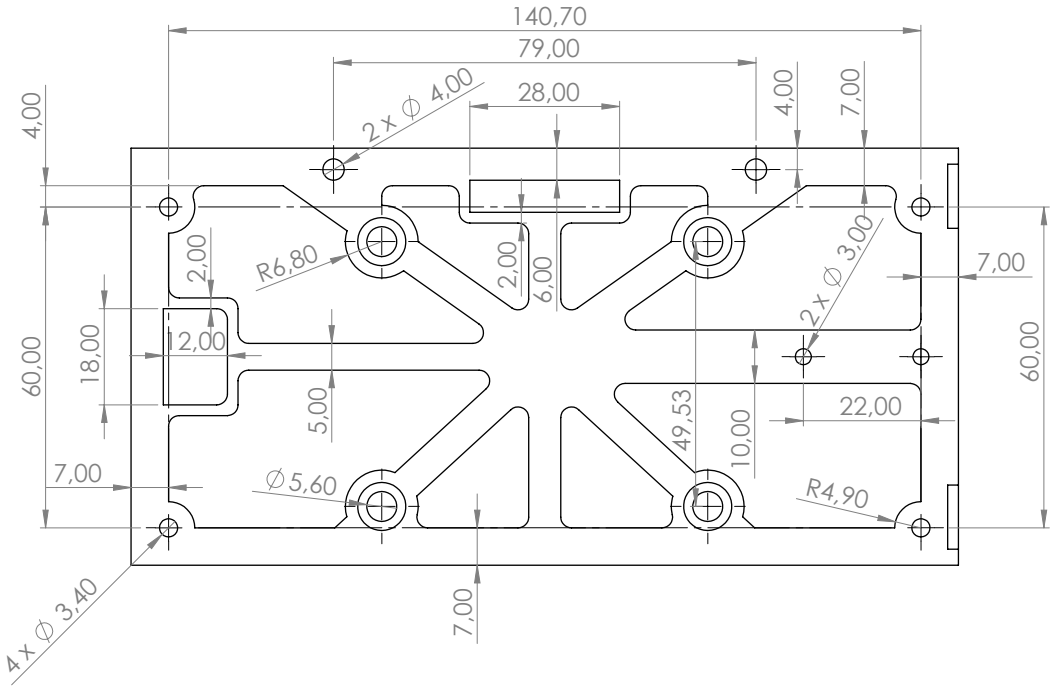
Anexo B: Planos de los componentes del vehículo

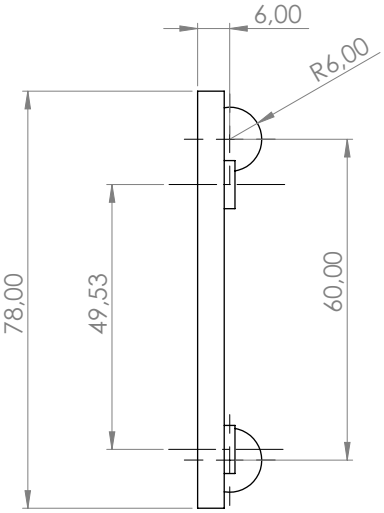


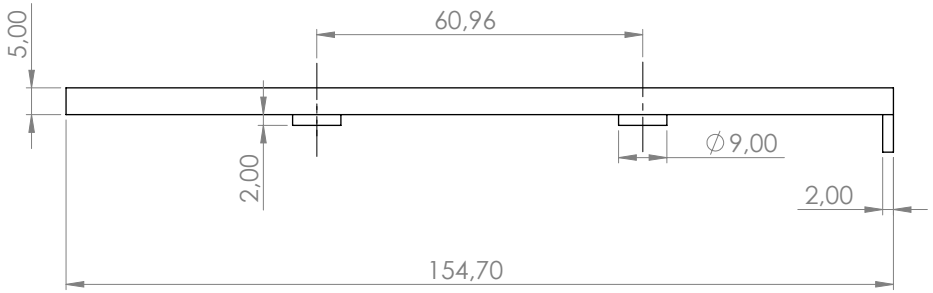


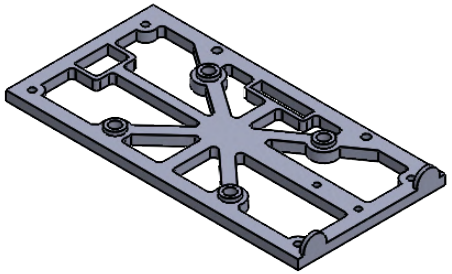


LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L						Diseño orientado a impresión 3D sin necesidad de soporte.	NO CAMBIE LA ESCALA	REVISIÓN 0
	NOMBRE	FIRMA	FECHA				TÍTULO: Placa Base	
DIBUJ.	Diego Cubillo		23/08/22					
VERIF.								
APROB.								
FABR.								
CALID.				MATERIAL: PLA			N.º DE DIBUJO Vehículo-01	A3
							ESCALA:1:1	HOJA 1 DE 1





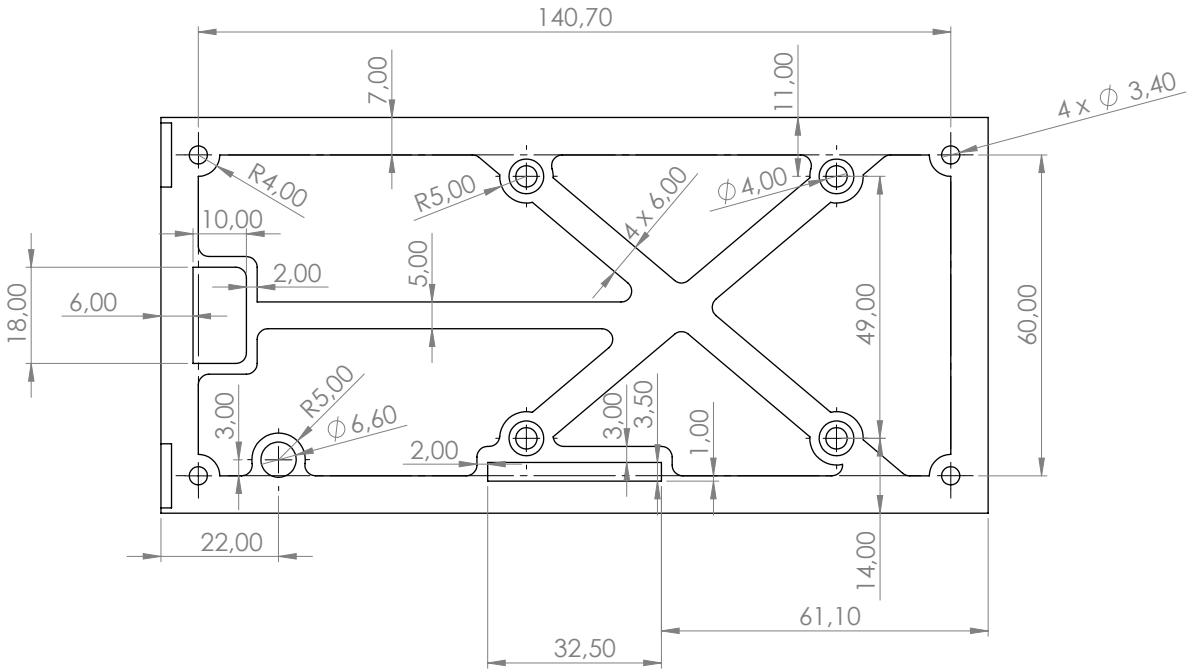


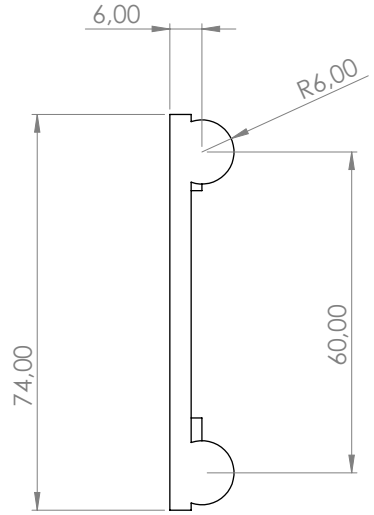


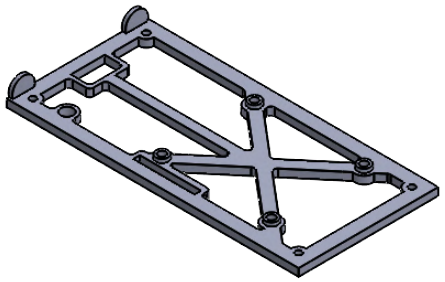
Nervios sin acotar 6,00

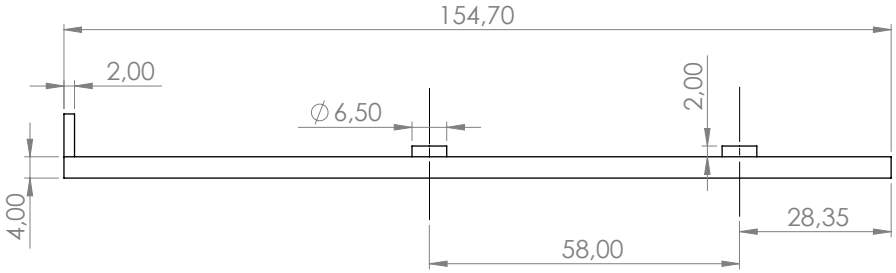
Redondeos sin acotar R2,00

LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L			Diseño orientado a impresión 3D sin necesidad de soporte.				REVISIÓN 1	
	NOMBRE	FIRMA	FECHA				TÍTULO: Placa 1	
DIBUJ.	Diego Cubillo		23/08/22					
VERIF.								
APROB.								
FABR.								
CALID.				MATERIAL: PLA			N.º DE DIBUJO Vehiculo_02	A3
							ESCALA:1:1	HOJA 1 DE 1



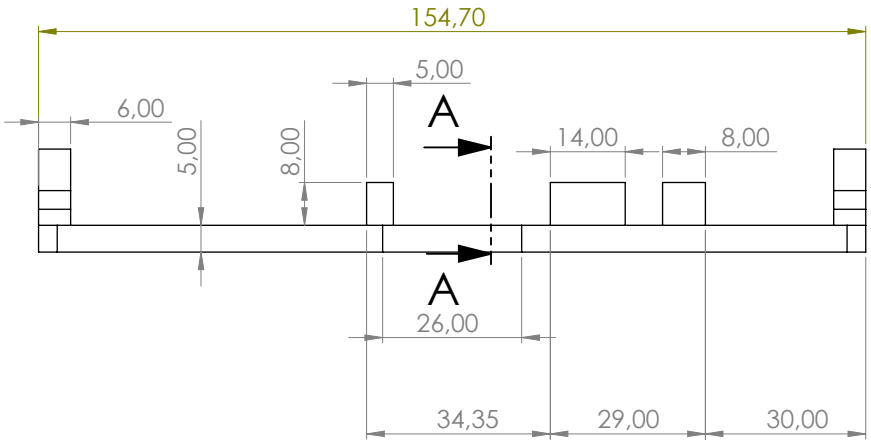


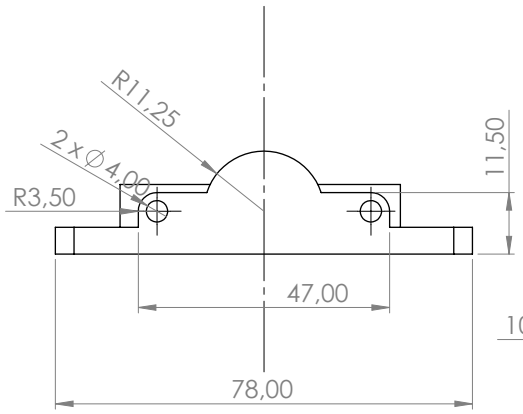


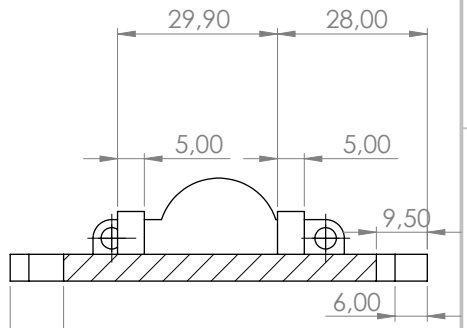


Redondeos de R 2 mm

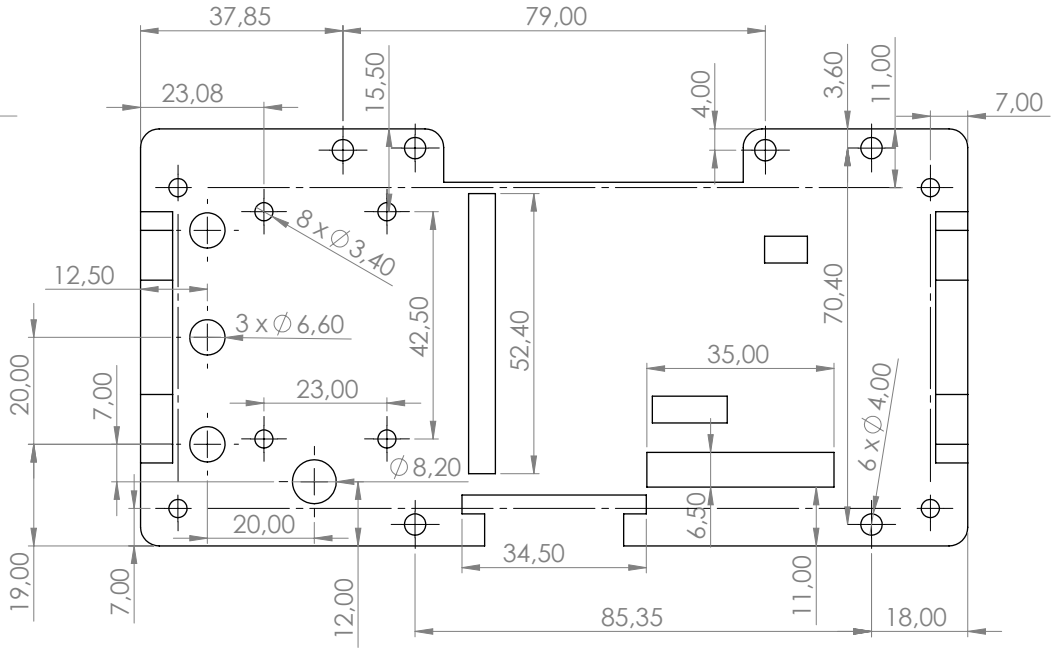
LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L			Diseño orientado a impresión 3D sin necesidad de soporte.				REVISIÓN 1	
	NOMBRE	FIRMA	FECHA				TÍTULO: Placa 2	
DIBUJ.	Diego Cubillo		23/08/22					
VERIF.								
APROB.								
FABR.								
CALID.				MATERIAL: PLA			N.º DE DIBUJO Vehículo_03	A3
							ESCALA:1:1	HOJA 1 DE 1

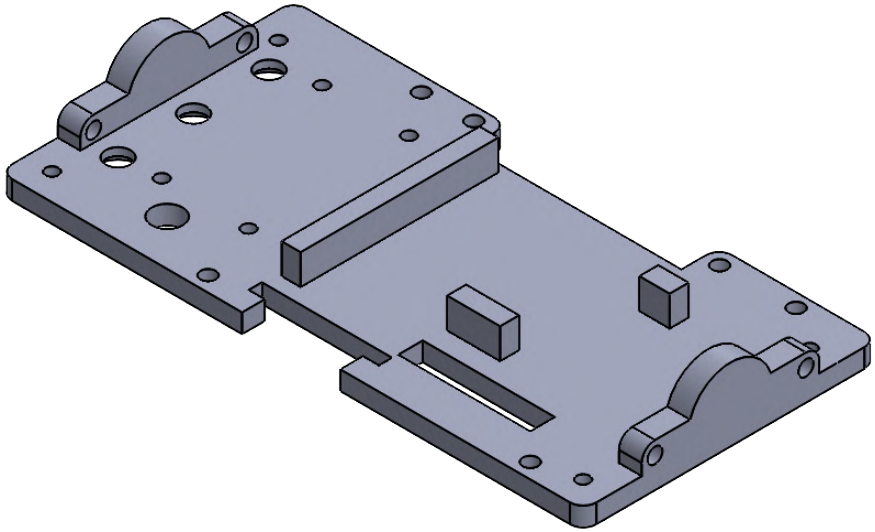






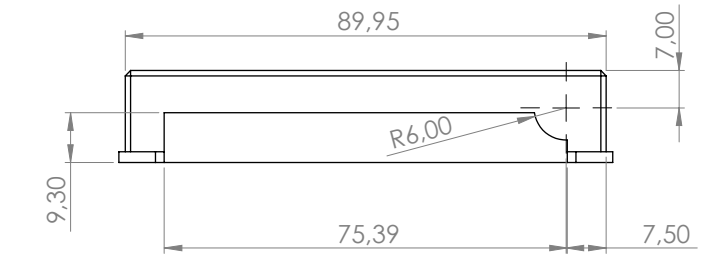
SECCIÓN A-A

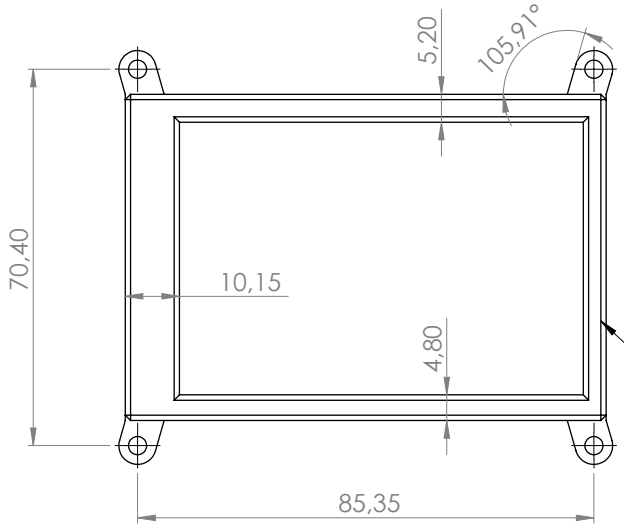


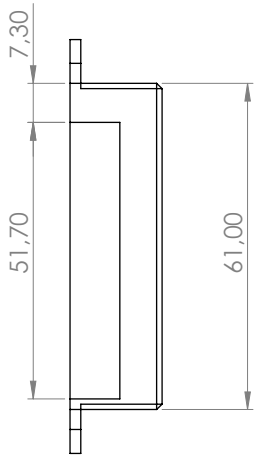


Redondeos sin acotar R3,50

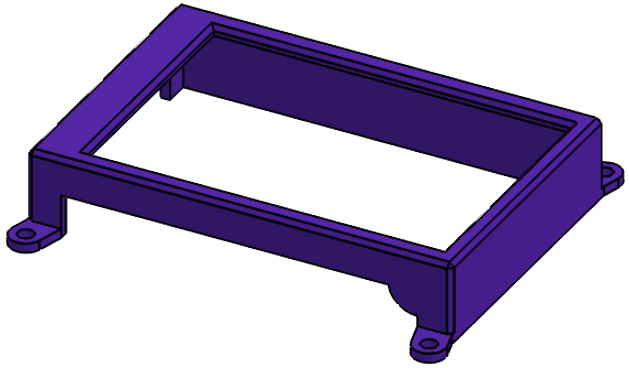
LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L			Diseño orientado a impresión 3D sin necesidad de soporte.				REVISIÓN 1	
	NOMBRE	FIRMA	FECHA				TÍTULO: Placa 3	
DIBUJ.	Diego Cubillo		23/08/22					
VERIF.								
APROB.								
FABR.								
CALID.				MATERIAL: PLA			N.º DE DIBUJO Vehículo-04	A3
							ESCALA:1:1	HOJA 1 DE 1

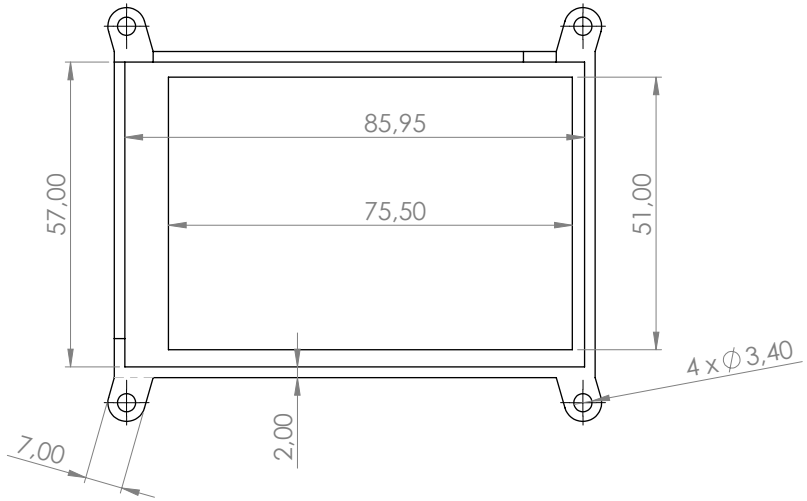




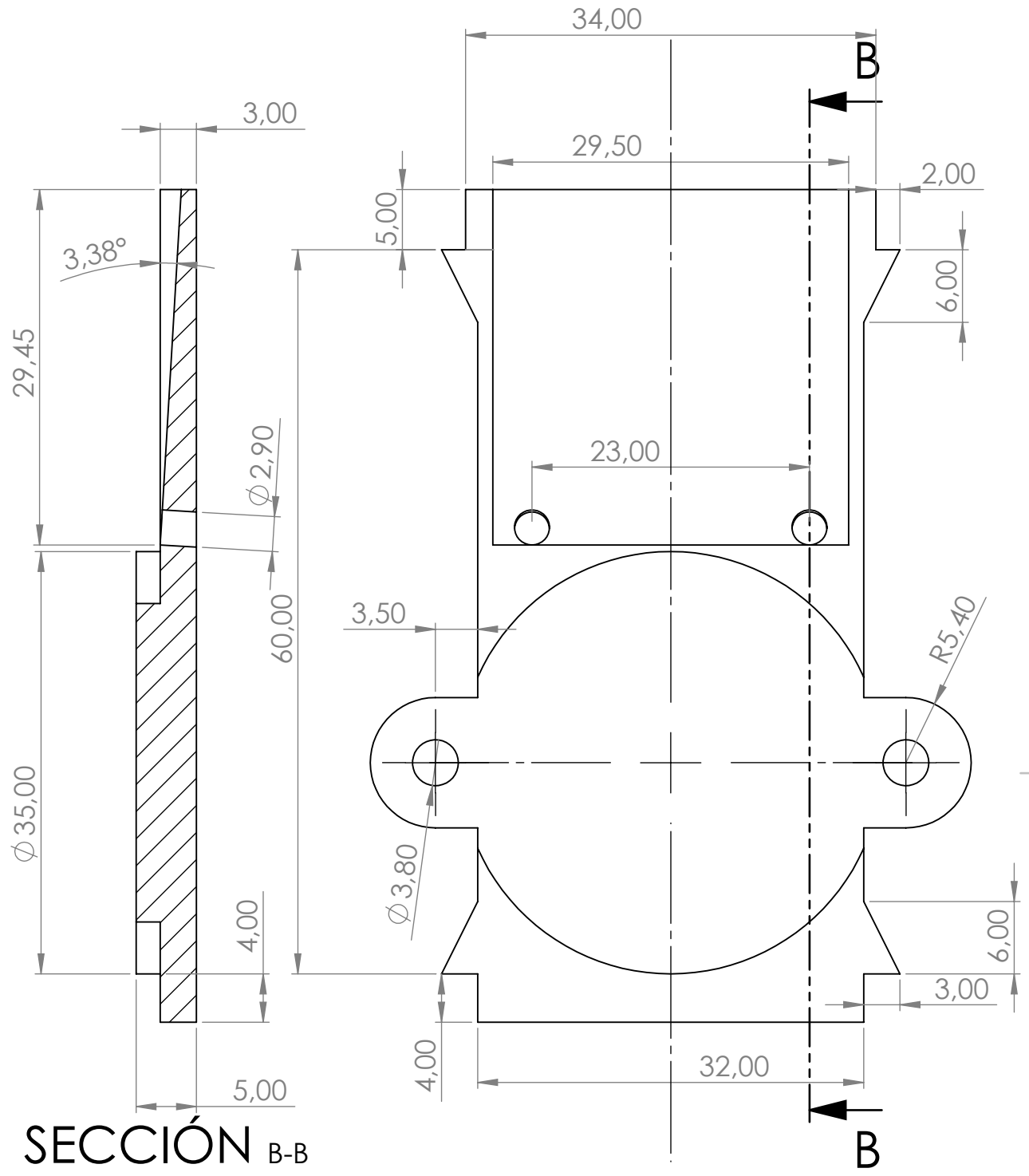


Chaflanes de 45° x 1mm



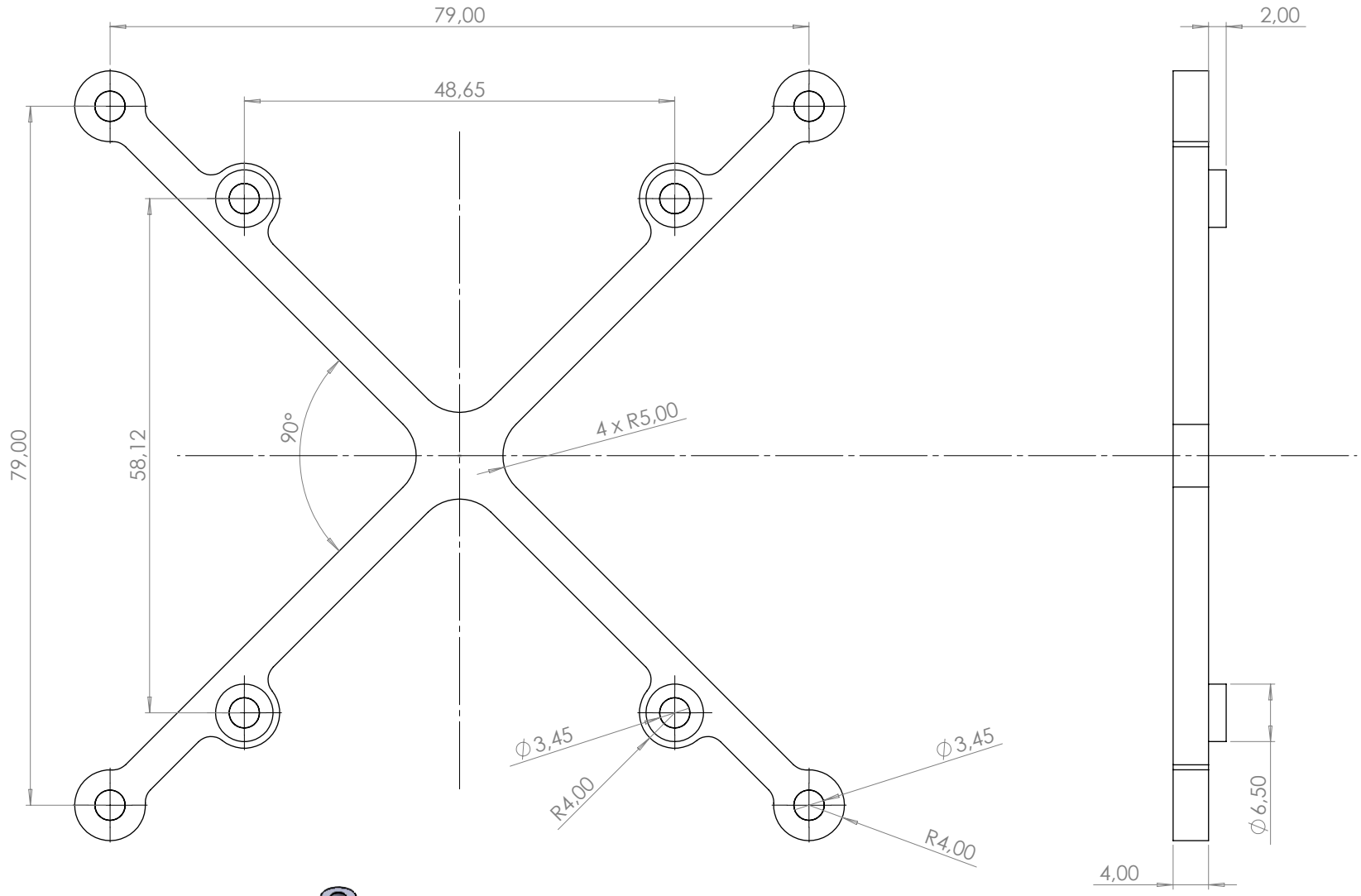


LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L			Diseño orientado a impresión 3D sin necesidad de soporte.			REVISIÓN 1	
	NOMBRE	FIRMA	FECHA			TÍTULO:	
DIBUJ.	Diego Cubillo		23/08/22			Marco pantalla	
VERIF.							
APROB.							
FABR.							
CALID.				MATERIAL:		N.º DE DIBUJO	A3
				PLA		Vehículo_05	
						ESCALA:1:1	HOJA 1 DE 1

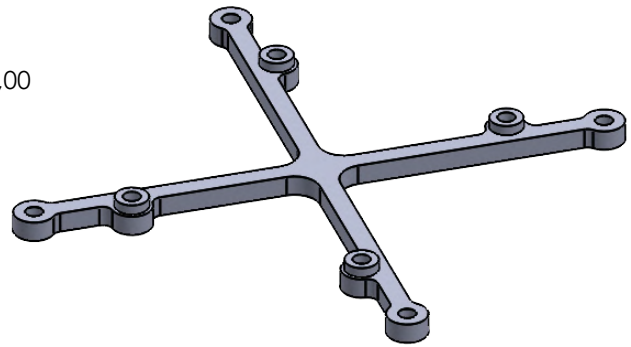


SECCIÓN B-B

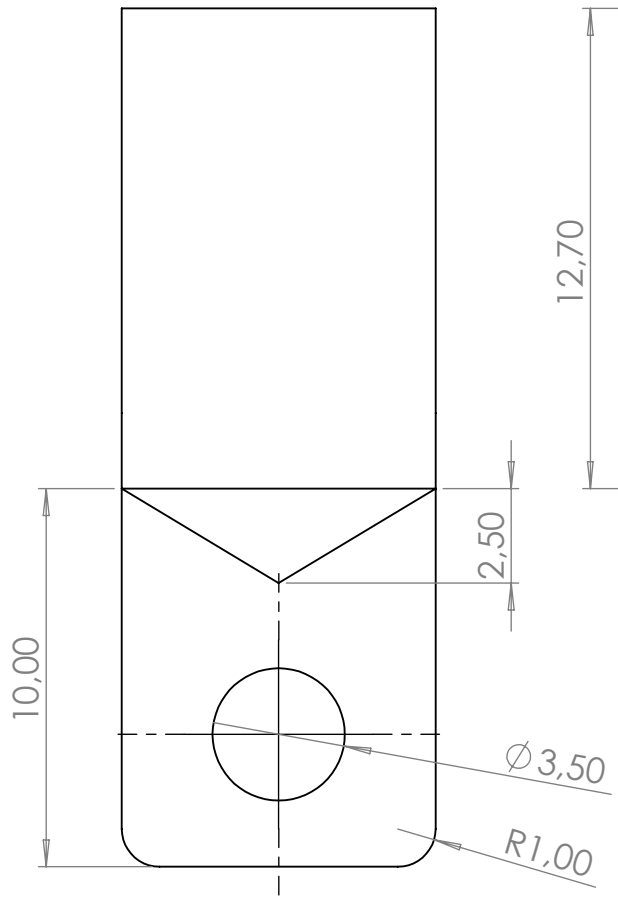
LAS COTAS SE EXPRESAN EN MM ISO 1768 - 1 c, L			Diseño orientado a impresión 3D sin necesidad de soporte.			REVISIÓN 1	
	NOMBRE	FIRMA	FECHA			TÍTULO: Placa de rueda	
DIBUJ.	Diego Cubillo		23/08/22				
VERIF.							
APROB.							
FABR.							
CALID.				MATERIAL: PLA		N.º DE DIBUJO Vehículo_06	A4
						ESCALA:2:1	HOJA 1 DE 1

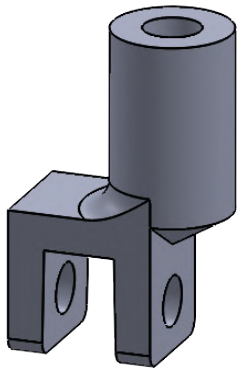


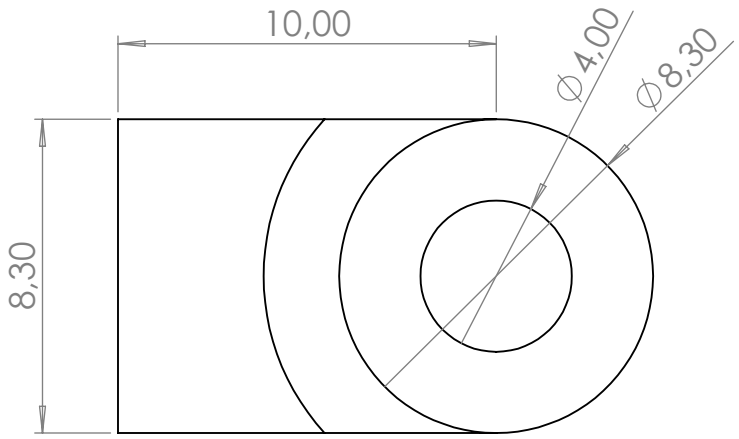
Redondeos no defiinidos R2,00



LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L						Diseño orientado a impresión 3D sin necesidad de soporte.	NO CAMBIE LA ESCALA	REVISIÓN 0
	NOMBRE	FIRMA	FECHA				TÍTULO: Soporte para Raspberry Pi	
DIBUJ.	Diego Cubillo		23/08/22					
VERIF.								
APROB.								
FABR.								
CALID.							N.º DE DIBUJO MóduloNav-01	A3
				MATERIAL: PLA				
							ESCALA:2:1	HOJA 1 DE 1







LAS COTAS SE EXPRESAN EN MM ISO 2768 - 1 c, L	Diseño orientado a impresión 3D sin necesidad de soporte.
NO CAMBIE LA ESCALA	REVISIÓN 2

	NOMBRE	FIRMA	FECHA
DIBUJ.	Diego Cubillo		23/08/22
VERIF.			
APROB.			
FABR.			
CALID.			

TÍTULO:

Adaptador Vehículo

MATERIAL: PLA

N.º DE DIBUJO: ModuloNav_02

A4

ESCALA:5:1

HOJA 1 DE 1

Anexo C: Objetivos de Desarrollo Sostenible

Este proyecto está alineado con el ODS 9, favoreciendo una mayor eficiencia en los procesos industriales e introduciendo innovación en las posibilidades de uso del espacio de fabricación. La aplicación de SLAM en la floreciente robótica móviles una de las innovaciones más importantes de la robótica industrial.

También se alinea con el ODS 4, ya que el resultado de este trabajo se puede usar intelectual y físicamente para la realización de otros trabajos de investigación, y de forma segura como medio docente para la educación en conceptos de regulación automática y robótica en el ICAI.

Por último, el diseño modular e impreso en 3D que garantiza la fácil adquisición de repuestos, la adaptabilidad de fragmentos del diseño a actualizaciones de hardware y el uso de materiales reciclables se alinean con el ODS 12, teniendo la sostenibilidad en cuenta en el proceso de diseño.

Figura 5: ODS a los que contribuye este proyecto

Anexo D: Códigos Python

Código 1: Publicador de posición del vehículo en el cartografiado

```
from geometry_msgs.msg import TransformStamped, Pose

import rclpy
from rclpy.node import Node

from tf2_ros import TransformBroadcaster

import tf_transformations

import socket
from struct import unpack

class FramePublisher(Node):

    def __init__(self):
        super().__init__('kitt_pose_tf2_broadcaster')

        # Initialize the transform broadcaster
        self.br = TransformBroadcaster(self)

        # Declare topic /kitt_pose
        self.publisher = self.create_publisher(Pose, 'kitt_pose', 1)

        # Init TCP server and communication
        self.s = socket.socket()
        self.s.connect(('192.168.0.123', 4000))
        print("connected")

        # Call on timer function twice per second
        self.timer = self.create_timer(0.1, self.on_timer)
```

```
    def on_timer(self):
        # Receive new TCP message
        realmsg = 0
        while realmsg == 0:
            newdata = self.s.recv(13)
            if newdata and unpack('f', newdata[0:4])[0] != 0.0:
                data = newdata
                realmsg = 1

        # Reshape TCP message
        posx = unpack('f', data[0:4])[0]
        posy = -1 * unpack('f', data[4:8])[0]
        angz = -1 * unpack('f', data[8:12])[0]

        t = TransformStamped()

        # Read message content and assign it to
        # corresponding tf variables
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'odom'
        t.child_frame_id = 'kitt'

        # Kitt only exists in 2D, thus we get x and y translation
        # coordinates from the message and set the z coordinate to 0
        t.transform.translation.x = posx
        t.transform.translation.y = posy
        t.transform.translation.z = 0.0

        # For the same reason, kitt can only rotate around one axis
        # and this why we set rotation in x and y to 0 and obtain
        # rotation in z axis from the message
        q = tf_transformations.quaternion_from_euler(0.0, 0.0, angz)
        t.transform.rotation.x = q[0]
        t.transform.rotation.y = q[1]
        t.transform.rotation.z = q[2]
        t.transform.rotation.w = q[3]

        # Send the transformation
        self.br.sendTransform(t)

        # Publish kitt_pose topic
        p = Pose()
        p.position.x = posx
```

```
        p.position.y = posy
        p.position.z = 0.0
        p.orientation.x = q[0]
        p.orientation.y = q[1]
        p.orientation.z = q[2]
        p.orientation.w = q[3]

        self.publisher.publish(p)



def main():
    rclpy.init()
    node = FramePublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

Código 2: Transformación estática del tf2 del LiDAR respecto al vehículo

```
import sys

from geometry_msgs.msg import TransformStamped

import rclpy
from rclpy.node import Node

from tf2_ros.static_transform_broadcaster import StaticTransformBroadcaster

import tf_transformations


class StaticFramePublisher(Node):
    """
    Broadcast transforms that never change.

    This example publishes transforms from `kitt` to a fixed LiDAR camera frame.
    The transforms are only published once at startup, and are constant for all
    time.
    """

    def __init__(self):
        super().__init__('static_camera_tf2_broadcaster')

        self._tf_publisher = StaticTransformBroadcaster(self)

        # Publish static transforms once at startup
        self.make_transforms()

    def make_transforms(self):
        static_transformStamped = TransformStamped()
        static_transformStamped.header.stamp = self.get_clock().now().to_msg()
        static_transformStamped.header.frame_id = 'kitt'
        static_transformStamped.child_frame_id = 'camera_link'
        static_transformStamped.transform.translation.x = 0.19
        static_transformStamped.transform.translation.y = 0.0
        static_transformStamped.transform.translation.z = 0.0
        static_transformStamped.transform.rotation.x = 0.0
        static_transformStamped.transform.rotation.y = 0.0
        static_transformStamped.transform.rotation.z = 0.0
        static_transformStamped.transform.rotation.w = 1.0

        self._tf_publisher.sendTransform(static_transformStamped)


def main():
    # initialize node
```

```
rclpy.init()
node = StaticFramePublisher()
try:
    rclpy.spin(node)
except KeyboardInterrupt:
    pass

rclpy.shutdown()
```

Código 3: Archivo de ejecución de los nodos ros2 para cartografiar

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([


        # Realsense
        Node(
            package='realsense2_camera',
            executable='realsense2_camera_node'
        ),
        Node(
            package='depthimage_to_laserscan',
            executable='depthimage_to_laserscan_node',
            remappings=[
                ('/depth', '/depth/image_rect_raw'),
                ('/depth_camera_info', '/depth/camera_info')
            ]
        ),
        Node(
            package='slam_toolbox',
            executable='sync_slam_toolbox_node',
            output='screen',
            parameters=[{'mode': 'mapping'},
                        {'base_frame': 'camera_depth_frame'}]
        ),
        Node(
            package='kitt_to_ros2',
            executable='kitt_pose_tf2_broadcaster1'
        ),
        Node(
            package='kitt_to_ros2',
            executable='static_camera_tf2_broadcaster'
        )
    ])
```

Código 4: Cálculo de posición del vehículo mediante integración de velocidades

```
function PAYLOAD  = SLAM_CODER(CONTROL)

%-------------------------------------------------------------------
% INITIALIZATION
%-------------------------------------------------------------------
persistent init TIMER x y shi
if isempty(init)
    TIMER = 0;
    x = 0;
    y = 0;
    shi = 0;
    init = 1;
end

%-------------------------------------------------------------------
% TIMER UPDATE
%-------------------------------------------------------------------
ts = CONTROL.PARAM.SAMPLING_TIME;
TIMER = TIMER + ts;

%-------------------------------------------------------------------
%FORWARD VELOCITY AND YAW RATE UPDATE
%-------------------------------------------------------------------
v = CONTROL.INPUT.FORWARD_VEL;
w = CONTROL.INPUT.YAW_RATE;

%-------------------------------------------------------------------
% INTEGRATION
%-------------------------------------------------------------------
shi = shi + w*ts;
vx = v*cos(shi);
vy = v*sin(shi);
x = x + vx*ts;
y = y + vy*ts;

if TIMER >= 0.1-ts/2
    % Payload definition
    PAYLOAD = [typecast(single(x),'uint8')'
               typecast(single(y),'uint8')'
               typecast(single(shi),'uint8')'
               ones(1,1,'uint8')];
    TIMER = 0;
```

```
else
    PAYLOAD = zeros(13,1,'uint8');
end

return
```

Anexo E: Modelo dinámico del coche

Vehículo de dos ruedas con tracción diferencial

1. Descripción del problema

El vehículo mostrado en la figura 1.1 es un vehículo de dos ruedas accionado eléctricamente mediante un sistema de control digital. El sistema de control modifica la componentes común y diferencial de la tensión aplicada a los motores que accionan las ruedas para determinar las velocidades de avance y giro del vehículo de forma independiente.

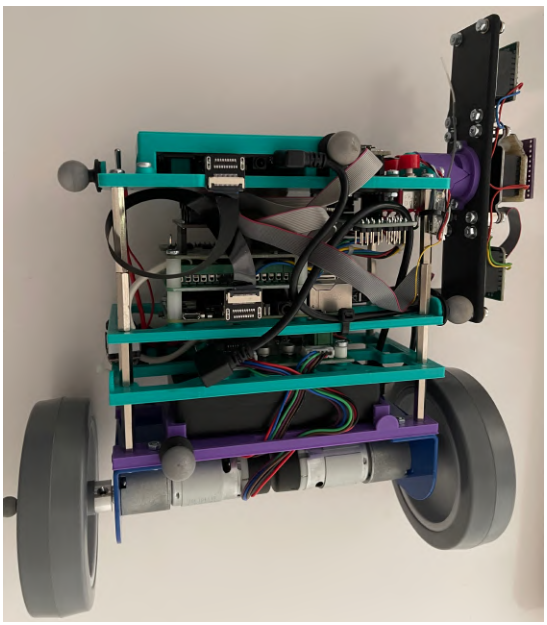


Figura 1: Vehículo con tracción diferencial empleado en el laboratorio

El vehículo que se utiliza en el laboratorio dispone de tracción diferencial basada en dos motores de corriente continua que hacen avanzar el vehículo mediante un par común, o lo hacen girar sobre sí mismo mediante un par diferencial. La velocidad angular de cada motor se mide mediante un *encoder* incremental magnético. A partir de las velocidades angulares de los motores se pueden estimar las velocidades de avance y de giro del vehículo. Además, el vehículo está equipado con una IMU (*Inertial Measurement Unit*) que incluye un giróscopo y un acelerómetro de 3 ejes. El giróscopo mide la velocidad de giro, mientras que el acelerómetro mide la aceleración de avance. Además, dos sensores laterales VL6180X ToF (*Time of Flight*) permiten medir la distancia y la orientación con respecto a cualquier superficie vertical.

El sistema de control se implanta mediante un algoritmo que se ejecuta en un dispositivo de cálculo, en este caso una Raspberry pi 3B+. Los sistemas de control digitales o en tiempo discreto se ejecutan de forma intermitente, es decir, cada cierto tiempo se adquieren las medidas de los sensores (giróscopo, acelerómetro y *encoders*), se procesan mediante el algoritmo de control y se calculan los mandos (tensiones de los motores) que se van a aplicar.

Para que el diseño del algoritmo de control resulte relativamente sencillo, los instantes de tiempo en los que actúa el control deben estar separados por el mismo intervalo temporal o período de muestreo. Cada vez que se calcula un nuevo valor de los mandos, estos se mantienen constantes durante el siguiente período de muestreo.

Si el período de muestreo es suficientemente pequeño, es posible realizar el diseño del sistema de control en tiempo continuo, por resultar más sencillo, y posteriormente convertir dicho sistema de control en un algoritmo en tiempo discreto. Para períodos de muestreo mayores, los efectos de la implantación en tiempo discreto del sistema de control pueden producir una respuesta muy diferente a la esperada. En estos casos, conviene realizar el diseño del sistema de control directamente en tiempo discreto.

2. Modelo del vehículo

El propósito de esta sección es obtener un modelo matemático en tiempo continuo, que describa el comportamiento del vehículo y que será útil para evaluar diferentes estrategias de control mediante simulación. Se considera como signo positivo para las variables eléctricas de los motores y para el giro de motores y ruedas el que hace avanzar al vehículo hacia adelante. El ángulo de giro se considera positivo si se produce en el sentido de las agujas del reloj visto desde arriba.

Los parámetros eléctricos, geométricos y mecánicos que intervienen en el modelo del vehículo son los siguientes:

- Los parámetros del motor: resistencia R_m, constante eléctrica K_e y mecánica K_t, la inercia del motor + rueda en el eje del motor I_m, la constante de fricción viscosa D_m, la relación de engranajes del motor n_1 y el par de fricción máximo T_{rmax}. Todos los parámetros están referidos al eje de salida del motor.
- La masa M del chasis.
- La masa m de cada rueda, incluyendo el rotor del motor.
- El momento de inercia I_z de giro del vehículo con respecto al eje vertical o guiñada.
- La fricción viscosa D_z de giro del vehículo con respecto al eje vertical o guiñada.
- La distancia W de separación entre las ruedas.
- El radio R de las ruedas.
- La relación de engranajes n_2 de la rueda.

En las ecuaciones del modelo, se manejarán variables asignadas a los motores izquierdo (subíndice *l*) y derecho (subíndice *r*) y variables en modo común (subíndice *c*) y diferencial (subíndice *d*). Si x es una variable cualquiera (tensión, corriente, par, velocidad angular o velocidad lineal), la relación entre los valores asignados a los motores izquierdo x_l y derecho x_r y a las componentes común x_c y diferencial x_d son las siguientes:

$$x_c = \frac{x_l + x_r}{2} \qquad x_d = \frac{x_l - x_r}{2}$$

$$x_l = x_c + x_d \qquad x_d = x_c - x_d$$

El modelo del vehículo es un modelo no lineal en espacio de estado donde las entradas son las tensiones aplicadas a los dos motores de corriente continua: u_l para el motor izquierdo y u_r para el motor derecho. Las variables de salida son la velocidad de avance v y la velocidad angular de giro ω.

La velocidad diferencial de las ruedas v_d y las velocidades angulares común ω_c y diferencial ω_d de las ruedas están relacionadas con las velocidades del vehículo:

$$v_d = \frac{W}{2}\omega$$
$$\omega_c = \frac{v_c}{R} = \frac{v}{R}$$
$$\omega_d = \frac{v_d}{R} = \frac{W\omega}{2R}$$

Las velocidades angulares común ω_{mc} y diferencial ω_{md} de los motores se pueden calcular a partir de las velocidades angulares común y diferencial de las ruedas:

$$\omega_{mc} = n_2\omega_c$$
$$\omega_{md} = n_2\omega_d$$

Las ecuaciones en modo común y diferencial de los motores son las siguientes:

- Ecuación eléctrica del rotor:

$$u_c = R_m i_c + L_m \frac{di_c}{dt} + e_c$$
$$u_d = R_m i_d + L_m \frac{di_d}{dt} + e_d$$

  Siendo u_c y u_d las tensiones de alimentación de los motores en modo común y diferencial, e_c y e_d las fuerzas contraelectromotrices inducidas en el circuito del rotor para ambos modos, i_c e i_d las corrientes que circulan por el rotor para cada modo de operación, R_m la resistencia del rotor y L_m la inductancia de dispersión.

- Ecuaciones electromecánicas:

$$T_{mc} = K_t i_c \quad T_{md} = K_t i_d$$
$$e_c = K_e \omega_{mc} \quad e_d = K_e \omega_{md}$$

  siendo T_{mc} y T_{md} los pares motores provocados por las corrientes de los motores en modo común y diferencial, ω_{mc} y ω_{md} las velocidades angulares relativas entre rotor y estator en cada modo, K_t la constante de par y K_e la constante eléctrica del motor. Ambas constantes tienen prácticamente el mismo valor numérico en el Sistema Internacional, aunque las pérdidas provocan que la constante de par K_t sea ligeramente inferior a la constante eléctrica K_e.

- Pares de fricción estáticos: los pares de fricción estáticos T_{rl} y T_{rr} de cada motor son iguales a los correspondientes pares motores si la velocidad del motor es nula y el

valor absoluto del par motor es inferior al par de fricción máximo T_{rmax}. En cuanto el valor absoluto del par motor supera el par de fricción máximo T_{rmax}, el motor comienza a girar y, justo en ese momento, el par de fricción T_r es igual a su valor máximo T_{rmax} con el signo del par motor ($T_{rl} = \frac{T_{ml}}{|T_{ml}|} T_{rmax}$ y $T_{rr} = \frac{T_{mr}}{|T_{mr}|} T_{rmax}$). Cuando el motor gira, el par de fricción estático mantiene siempre su valor máximo T_{rmax} con el signo de la velocidad de giro del motor ($T_{rl} = \frac{\omega_{ml}}{|\omega_{ml}|} T_{rmax}$ y $T_{rr} = \frac{\omega_{mr}}{|\omega_{mr}|} T_{rmax}$). Los pares de fricción estática común T_{rc} y diferencial T_{rd} de los motores se calculan de la siguiente forma:

$$T_{rc} = \frac{T_{rl} + T_{rr}}{2}$$
$$T_{rd} = \frac{T_{rl} - T_{rr}}{2}$$

- Las componentes común T_{dc} y diferencial T_{dd} de los pares de fricción viscosa de los motores resultan:

$$T_{dc} = D_m \omega_{mc}$$
$$T_{dd} = D_m \omega_{md}$$

- Los pares netos de los motores en modo común y diferencial T_c y T_d se obtienen sumando todos los pares anteriores y los pares provocados por las fuerzas de rozamiento de las ruedas con el suelo:

$$T_c = T_{mc} - T_{dc} - T_{rc} - F_{rc} \frac{R}{n_2}$$
$$T_d = T_{md} - T_{dd} - T_{rd} - F_{rd} \frac{R}{n_2}$$

siendo F_{rc} y F_{rd} las fuerzas de rozamiento en el punto de contacto de cada rueda con el suelo, R el radio de las ruedas y n_2 la relación de engranajes de la rueda.

- Ecuación dinámica de los motores en modo común y diferencial: el movimiento de rotación de los motores está impulsado por el par neto en el eje del motor:

$$I_m \frac{d\omega_{mc}}{dt} = T_c$$
$$I_m \frac{d\omega_{md}}{dt} = T_d$$

siendo I_m el momento de inercia del motor + rueda en el eje del motor.

Para obtener las ecuaciones del modelo se considerará que el avance y el giro del vehículo están desacoplados. El avance se controla mediante la tensión común u_c aplicada a ambos motores, mientras que el giro se controla mediante la tensión diferencial u_d. A continuación, se presentan por separado el modelo de avance (modo común) y el modelo de giro (modo diferencial).

2.1 Modelo de avance

En todas las ecuaciones que aparecen a continuación, el subíndice c en cada variable representa la media aritmética de los valores de esa variable en ambos motores. La ecuación dinámica de rotación de las ruedas en modo común resulta:

$$T_c = T_{mc} - T_{dc} - T_{rc} - F_{rc}\frac{R}{n_2} = I_m\frac{d\omega_{mc}}{dt}$$

Despejando la fuerza de rozamiento F_{rc}, se obtiene:

$$F_{rc} = \frac{n_2}{R}\left(T_{mc} - T_{dc} - T_{rc} - I_m\frac{d\omega_{mc}}{dt}\right) = \frac{n_2}{R}\left(T_{mc} - T_{dc} - T_{rc} - n_2 I_m\frac{d\omega_c}{dt}\right)$$
$$= \frac{n_2}{R}\left(T_{mc} - T_{dc} - T_{rc} - \frac{n_2 I_m}{R}\frac{dv}{dt}\right)$$

Por otra parte, la ecuación dinámica de traslación del vehículo, teniendo en cuenta que las únicas fuerzas que lo propulsan son las de rozamiento de las ruedas con el suelo, es la siguiente:

$$F_{rl} + F_{rl} = 2F_{rc} = (2m + M)\frac{dv_c}{dt}$$

Igualando las ecuaciones anteriores para eliminar la componente común de la fuerza de rozamiento F_{rc} de las ruedas con el suelo, resulta:

$$(2m + M)\frac{dv_c}{dt} = \frac{2n_2}{R}\left(T_{mc} - T_{dc} - T_{rc} - \frac{n_2 I_m}{R}\frac{dv_c}{dt}\right)$$

$$\frac{2n_2}{R}(T_{mc} - T_{dc} - T_{rc}) = \left(2m + M + \frac{2n_2^2 I_m}{R^2}\right)\frac{dv}{dt}$$

Recopilando las ecuaciones anteriores y eliminando variables intermedias, resulta el siguiente modelo no lineal:

$$T_{mc} = \frac{K_t}{R_m}\left(u_c - \frac{\overbrace{n_2 K_e v}^{e_c}}{R}\right)$$

$$\frac{2n_2}{R}\left(T_{mc} - \frac{n_2 D_m v}{R} - T_{rc}\right) = \left(2m + M + \frac{2n_2^2 I_m}{R^2}\right)\frac{dv}{dt}$$

$$\frac{2n_2}{R}\left(\frac{K_t}{R_m}\left(u_c - \frac{n_2 K_e v}{R}\right) - \frac{n_2 D_m v}{R} - T_{rc}\right) = \left(2m + M + \frac{2n_2^2 I_m}{R^2}\right)\frac{dv}{dt}$$

$$\frac{2n_2 K_t}{R R_m}u_c - \frac{2n_2}{R}T_{rc} = \left(2m + M + \frac{2n_2^2 I_m}{R^2}\right)\frac{dv}{dt} + \frac{2n_2^2}{R^2}\left(\frac{K_t K_e}{R_m} + D_m\right)v$$

Finalmente, agrupando parámetros resulta la siguiente ecuación diferencial de primer orden:

$$\tau_m \frac{dv}{dt} + v = K_m \left(u_c - \frac{R_m}{K_t} T_{rc} \right) \quad \Rightarrow \quad \begin{cases} \tau_m = \dfrac{2m + M + \dfrac{2n_2^2 I_m}{R^2}}{\dfrac{2n_2^2}{R^2}\left(\dfrac{K_t K_e}{R_m} + D_m\right)} \\ K_m = \dfrac{\dfrac{2n_2 K_t}{R R_m}}{\dfrac{2n_2^2}{R^2}\left(\dfrac{K_t K_e}{R_m} + D_m\right)} \end{cases}$$

2.2 Modelo de giro

El giro del vehículo viene determinado por la tensión diferencial u_d aplicada a ambos motores. El subíndice d en cada variable representa la mitad de la resta de los valores de esa variable en ambos motores (su valor en el motor derecho menos su valor en el motor izquierdo). Si se desprecian la inductancia de dispersión, las ecuaciones electromecánicas de los motores en modo diferencial quedan:

$$i_d = \frac{u_d - e_d}{R_m}$$

$$e_d = K_e \omega_{md} = K_e n_2 \omega_d = \frac{K_e n_2 W}{2R} \omega$$

$$F_{rd} = \frac{n_2}{R}\left(T_{md} - T_{dd} - T_{rd} - I_m \frac{d\omega_{md}}{dt}\right) = \frac{n_2}{R}\left(T_{md} - T_{dd} - T_{rd}\right) - \frac{W n_2^2 I_m}{2R^2}\frac{d\omega}{dt}$$

$$T_{md} = K_t i_d = K_t \frac{u_d - e_d}{R_m} = K_t \frac{u_d - K_e \omega_{md}}{R_m} = \frac{K_t}{R_m} u_d - \frac{K_t K_e W n_2}{2 R_m R} \omega$$

$$T_{dd} = D_m \omega_{md} = D_m n_2 \omega_d = \frac{D_m n_2}{R} v_d = \frac{W D_m n_2}{2R} \omega$$

$$F_{rd} = \frac{n_2 K_t}{R R_m} u_d - \frac{W n_2^2}{2R^2}\left(\frac{K_t K_e}{R_m} + D_m\right)\omega - \frac{n_2}{R} T_{rd} - \frac{W n_2^2 I_m}{2R^2}\frac{d\omega}{dt}$$

Por otra parte, la ecuación dinámica del giro del vehículo con respecto al eje vertical, teniendo en cuenta que F_{rd} es la única fuerza externa que produce aceleración angular, es:

$$F_{rd} W = I_z \frac{d\omega}{dt} + D_z \omega$$

$$\frac{n_2 W}{R}\left(T_{md} - T_{dd} - T_{rd}\right) - \frac{W^2 n_2^2 I_m}{2R^2}\frac{d\omega}{dt} = I_z \frac{d\omega}{dt} + D_z \omega$$

$$\frac{n_2 W}{R}\left(T_{md} - T_{dd} - T_{rd}\right) = \left(\frac{W^2 n_2^2 I_m}{2R^2} + I_z\right)\frac{d\omega}{dt} + D_z \omega$$

I_z y D_z representan, respectivamente, la inercia y la fricción viscosa asociadas a la rotación con respecto al eje vertical del vehículo.

Si en la ecuación anterior se sustituye por su valor la fuerza de rozamiento diferencial F_{rd} y se agrupan términos, resulta:

$$\frac{n_2 W K_t}{R R_m} u_d - \frac{W n_2}{R} T_{rd} = \left(I_z + \frac{W^2 n_2^2 I_m}{2R^2} \right) \frac{d\omega}{dt} + \left(D_z + \frac{W^2 n_2^2}{2R^2} \left(\frac{K_t K_e}{R_m} + D_m \right) \right) \omega$$

En la deducción de la ecuación anterior se ha supuesto que el ángulo de cabeceo es despreciable, objetivo que debe garantizar el sistema de control del cabeceo.

Finalmente, agrupando parámetros resulta la siguiente ecuación diferencial de primer orden:

$$\tau_m \frac{d\omega}{dt} + \omega = K_m \left(u_d - \frac{R_m}{K_t} T_{Rd} \right) \quad \Rightarrow \quad \begin{cases} \tau_m = \dfrac{I_z + \dfrac{n_2^2 W^2 I_m}{2R^2}}{D_z + \dfrac{n_2^2 W^2}{2R^2} \left(\dfrac{K_t K_e}{R_m} + D_m \right)} \\ K_m = \dfrac{\dfrac{n_2 W K_t}{R R_m}}{D_z + \dfrac{n_2^2 W^2}{2R^2} \left(\dfrac{K_t K_e}{R_m} + D_m \right)} \end{cases}$$

Anexo F: Referencias de imágenes utilizadas

https://en.reichel-versand.de/images/L/DEV-MD25.jpg

https://www.sigmaelectronica.net/wp-content/uploads/2008/05/rd02-motor-wheel-kit.jpg

https://raspi.tv/wp-content/uploads/2018/03/Raspberry-Pi-3B-top_1500.jpg

https://www.amazon.com/-/es/Convertidor-de-nivel-l%C3%B3gico-bidireccional/dp/B00M7U5DV2

https://www.newark.com/productimages/large/en_US/31AC5355-40.jpg

https://www.electrokit.com/uploads/productimage/41014/41014411.png

https://cdn1-shop.mikroe.com/img/product/shuttle-click/shuttle-click-thickbox_default-2.jpg

https://media.parallax.com/wp-content/uploads/2020/10/08143827/64214c.jpg

https://sc04.alicdn.com/kf/Haffec27d68994f6eb9bd82b11109c525g/239992620/Haffec27d68994f6eb9bd82b11109c525g.jpg

https://www.waveshare.com/media/catalog/product/cache/1/image/800x800/9df78eab33525d08d6e5fb8d27136e95/3/_/3.5ap-bs2.jpg

https://ae01.alicdn.com/kf/H56f0afbc26934b45a8205c0bebc04cdcg.jpg?width=1000&height=1000&hash=2000

https://cdn1-shop.mikroe.com/img/product/mpu-imu-click/mpu-imu-click-thickbox_default-12x.jpg

https://www.creativefabrica.com/wp-content/uploads/2020/06/12/Compass-Black-and-White-Li
jpg

https://5.imimg.com/data5/PP/GS/MY-22169517/two-step-push-buttons-500x500.jpg

https://ae01.alicdn.com/kf/H90b6f7adfe0e43bba6dc828d0be760f8c.jpg

https://www.e-ika.com/images/thumbs/0005105_diodo-led-rgb-5mm-10uds.jpeg

https://www.creativefabrica.com/wp-content/uploads/2018/12/Computer-icon-by-rudezs
jpg

https://img.myloview.es/posters/camera-icon-camera-symbol-camera-vector-icon-700-1
jpg