Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Diseño e implementación de vehículo a escala de código abierto para navegación autónoma en ROS 2

Proyecto Integrador

Previo la obtención del Título de:

Ingeniero en Mecatrónica

Presentado por:

Luis Alberto Andrade Proaño

Guayaquil - Ecuador

Año: 2023

Dedicatoria

El presente proyecto lo dedico a mis padres, hermana, familia, amigos, compañeros de carrera, profesores de la universidad quienes me apoyaron y que gracias a ellos he llegado a cumplir este gran reto.

Luis Andrade Proaño

Agradecimientos

Mi más sincero agradecimiento a la universidad, facultad y docentes de la carrera de Ingeniería en Mecatrónica, por su trabajo y dedicación.

Al laboratorio RAMEL, por darme la oportunidad de trabajar para realizar el presente trabajo.

A mi hermana Kenia Andrade Proaño, quien me pudo contribuir con apoyo en los momentos más difíciles del desarrollo de esta tesis.

Luis Andrade Proaño

Declaración Expresa

"Los derechos de titularidad y explotación, me corresponden conforme al reglamento de propiedad intelectual de la institución; Luis Alberto Andrade Proaño doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Luis Andrade Proaño

Evaluadores

Firmado electrónicamente por:
MARCELO RODOLFO
FAJARDO PRUNA

Marcelo Fajardo Pruna, Ph.D.

Profesor de Materia

Firmado electrónicamente por:
FRANCISCO XAVIER
YUMBLA AREVALO

Francisco Yumbla Arevalo, Ph.D.

Tutor de proyecto

Resumen

Los vehículos con navegación autónoma es un campo de investigación en auge tanto en el mundo comercial como en el académico, pero los framework ya existentes no fueron diseñados para el ambiente latinoamericano, tanto en costo de manufactura como en implementación. Por lo tanto, se planea diseñar, construir y programar un vehículo autónomo a escala con la infraestructura de ROS2 con el fin de lograr una alternativa más económica para este producto. Se realizó el diseño y dimensionamiento de las piezas en donde se colocarán los sensores, la placa de control de actuadores y la CPU y por medio de manufactura aditiva se realizó un prototipo. A este prototipo se le implementaron herramientas SLAM, el cual le permite de realizar un mapeo del entorno en tiempo real y también poderlo navegar de forma autónoma. Aparte se desarrolló un entorno simulado en Gazebo, junto con un modelo del vehículo que cumple con las mismas funciones. El costo final de este proyecto es considerablemente menor al de su competencia. Tanto el manual de instrucciones como el proyecto fueron subidos a un repositorio de GitHub. Debido a eso, este producto puede ser considerado una alternativa viable como una plataforma educativa en el mercado actual.

Palabras Clave: Vehículos autónomos, ROS 2, accesibilidad, control de movimiento.

Abstract

Self-driving vehicles are a rising field for investigation; both in the commercial and academic world, but the already existing frameworks were not designed for a Latin American environment, especially in manufacturing costs and implementation. Therefore, the plan is to design, build, and program a scaled self-driving vehicle based on the ROS2 infrastructure to make a more economical alternative for this kind of product. The design and measurements of the pieces were the sensors, the actuators control board, and the CPU will be set were made and through additive manufacturing, a prototype was made. SLAM tools were implemented in the prototype, so that is capable of mapping its environment in real-time and self-navigate through it. Aside from that, a simulated environment designed in Gazebo and a virtual model of the vehicle, which accomplishes the same functions, were designed. The final cost is comparably lower than its competition. Both the instruction manual and the project itself are uploaded to a GitHub repository. Because of this, this product can be considered a viable alternative as a learning platform in the actual market.

Keywords: *Self-driving vehicles, ROS 2, Accessibility, movement control.*

Índice general

Abreviaturas

ESPOL	Escuela Superior Politécnica del Litoral
RAMEL	Robotics, Automation & Mechatronics Engineering Laboratory
SAE	Society of Automotive Engineers
SMA	Sistema de Manejo Automático
ROS	Robot Operating System
LiDAR	Light Detection and Ranging
IMU	Inertial Measurement Unit
ISO	International Organization for Standardization
PLA	Polylactide
USB	Universal Serial Bus
3D	Tres dimensiones
URDF	Unified Robot Description Format
SLAM	Simultaneous Localization and Mapping
AMCL	Adaptive Monte Carlo Localization
SKD	Sofware Development Kit

Simbología

m	Metro
V	Voltaje
mAh	Miliamperio hora
°	Grado
Hz	Hercio

Índice de figuras

Índice de tablas

Capítulo 1

1. Introducción

Los vehículos autónomos son considerados el futuro del mundo automovilístico. Esto se debe a no solo se busca brindar comodidad al conductor, ya que lo libera de la responsabilidad de conducir, permitiendo mayor tiempo para que pueda descansar o ser productivo, también se busca solucionar problemas que normalmente se producen como consecuencia del error humano, como los congestionamientos de largas distancias y alta duración, accidentes provocados por el estado de conciencia del conductor o por condiciones externas, entre otros. Los líderes en el mercado son: Waymo, el cual ha estado realizando pruebas con sus vehículos por más de 20 millones de millas en vías públicas, Tesla, los cuales han estado en el mercado con la función de "piloto automático" desde el 2014, Audi, Volvo, Nvidia, Ford, entre otros [1].

La navegación autónoma inteligente de los vehículos que se usan a diario es de alto interés no solo en compañías automovilísticas, sino también para investigación académica, gracias a esto, se ha acelerado la investigación y desarrollo de la arquitectura para este tipo de vehículo [2]. Esto ha provocado que varias universidades en todo el mundo excepto hayan desarrollado su propio modelo de vehículo autónomo a escala y tanto el diseño del hardware como el software se lo publica en una plataforma en línea de libre acceso.

1.1 Descripción del problema

La investigación sobre los vehículos autónomos comenzó alrededor de 1980, cuando las universidades estaban desarrollando dos tipos de vehículos: uno que requiera la infraestructura de carreteras y otro que no [3]. Desde el 2003, la agencia de investigación avanzada de proyectos de defensa (DARPA) de los Estados Unidos ha estado haciendo competencias para poner a prueba

los avances del desempeño de los vehículos autónomos [1]. Esto ha provocado una aceleración en la tecnología aplicada a estos vehículos y un aumento en el interés hacia el público.

En el 2015 se fundó la compañía Shenzen Yahboom Technology Co., Ltd. cuyos productos son robots educacionales con inteligencia artificial, plataformas para hardware Open Source y kits relacionados [4]. Quanser es una compañía que se fundó en 1989, cuyos productos están enfocados en la investigación y aplicación de control, robótica y mecatrónica [5]. Ambas compañías tienen su propia línea de vehículos autónomos, unos más sofisticados que otros.

Alrededor del mundo se encuentran activos equipos de varias universidades y comunidades sin fin de lucro (F1TENTH- Universidad de Pensilvania [6], MIT RACECAR- Instituto de tecnología de Massachusetts [7], MuSHR- Universidad de Washington [8]) que han desarrollado su propia plataforma de desarrollo para su modelo de vehículo autónomo a escala

Aunque existan este tipo de comunidades internacionalmente con sus respectivas plataformas, el enfoque que estas proveen son para el ambiente en donde fueron desarrollados. En Latinoamérica las infraestructuras de las calles, la cantidad de personas que cruzan la calle simultáneamente (incluyendo vendedores en carretilla) y las leyes de tráfico son diferentes a las de Estados Unidos y Europa. A través de diferentes regiones, la calidad de carreteras y caminos puede variar, también hay que considerar que las señalizaciones de trafico varían dependiendo de país en país a diferencia de Europa, donde la mayoría de países comparte señalización [9].

1.2 Justificación del problema

El desarrollo e investigación de vehículos autónomos conlleva una combinación de hardware avanzado, algoritmos de software y hasta modelos de inteligencia artificial. Los tipos de sensores sofisticados que se utilizan son LiDAR, cámaras, radar y/o GPS, los cuales tienen un

costo alto de producción e implementación en vehículos. Son sofisticados porque se requiere un alto nivel de precisión y deben ser confiables a la hora de la conducción.

El tipo de software que se utiliza tiende a ser bastante complejo, debido a que se debe recopilar una extensa cantidad de información del mundo real, la cual se utiliza a la hora de entrenar y mejorar los algoritmos de navegación.

El mercado para vehículos autónomos a escala es considerado un nicho, por lo que los costos de demanda tienden a ser relativamente altos. La compañía que los produce diseña y agregan sus propios elementos, los cuales pueden llegar a proveer funciones que sobrepasan las necesidades del cliente, de esta manera reduciendo la adaptabilidad del producto. Como consecuencia el nivel de accesibilidad hacia sus potenciales clientes (estudiantes e investigadores) es reducido.

Debido a esto, lo que se busca realizar con este proyecto es diseñar y ensamblar un vehículo autónomo a escala y levantar una plataforma que contendrá las instrucciones para imprimir las piezas requeridas en 3D, ensamblarlo y programarlo con el objetivo de ser ofrecer un modelo accesible para investigadores y estudiantes en Latinoamérica y promover el aprendizaje sobre el mundo de la navegación autónoma de los vehículos.

1.3 Objetivos

1.3.1 *Objetivo general*

Diseñar e implementar un vehículo a escala Open Source basado en la infraestructura de ROS 2 y levantar su respectiva plataforma, para promover el aprendizaje en el campo de navegación autónoma en Latinoamérica.

1.3.2 *Objetivos específicos*

1. Diseñar e implementar por medio de manufactura aditiva soportes para los sensores del vehículo.
2. Desarrollar una arquitectura de software para el control del vehículo incluido sensores y actuadores para navegación autónoma con la infraestructura en ROS 2.
3. Aplicar algoritmos SLAM utilizando un radar LiDAR para realizar un mapeo del entorno y aplicar algoritmos para la percepción, localización, planificación, y control del movimiento del vehículo.
4. Desarrollar un ambiente simulado utilizando Gazebo.
5. Desarrollar un repositorio en línea en donde se encuentren las instrucciones de la instalación del software y del hardware del vehículo.

1.4 Marco teórico

En la siguiente sección se detallan los fundamentos teóricos necesarios para el entendimiento general del problema, proyectos ya existentes y soluciones alternas actuales a la problemática.

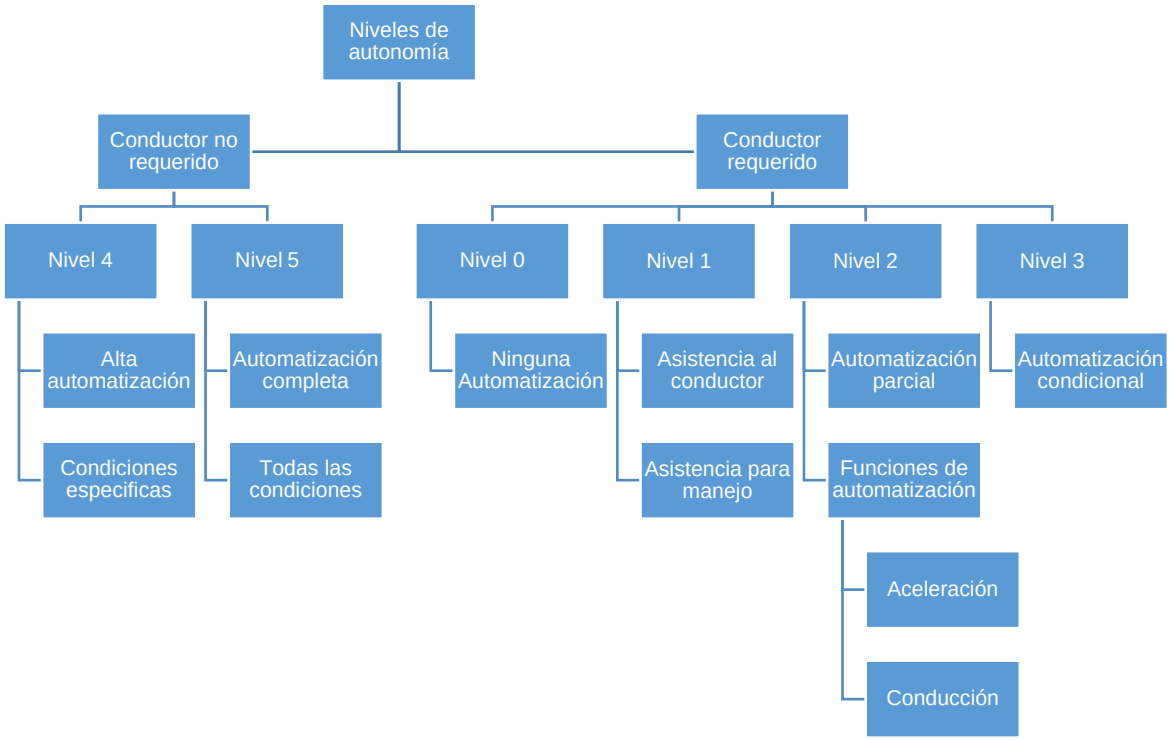
1.4.1 *Niveles de navegación autónoma*

Acorde a la Sociedad de Ingenieros de Automoción (SAE), se definen 6 niveles de navegación autónoma. En el nivel 0 no hay navegación autónoma, el conductor realiza la conducción completa del vehículo, incluso con sistema de seguridad activo. En el nivel 1, el SMA realiza un control lateral o longitudinal sobre el vehículo (ej. modo crucero) con la expectativa que el conductor realizara el resto de la conducción. En el nivel 2 el SMA es parecido al del nivel 1, pero no puede detectar y responder a objetos y eventos de forma completa, por lo tanto, el

conductor debe seguir con las manos en el volante durante estos procesos automatizados, con el fin de poder tomar control del vehículo en cualquier instante (ej. Frenado de emergencia automático). En el nivel 3 el SMA realiza toda la conducción. El vehículo puede detectar con eficiencia el ambiente alrededor y poder navegarlo por su cuenta. Se tiene un sistema de aviso al conductor en caso de que los límites en el diseño operacional vayan a ser sobrepasados y en caso de que llegue a haber una falla en el sistema. En el nivel 4 no requiere ningún tipo de interacción humana en la mayoría de las circunstancias. Para este tipo de navegación autónoma el SMA realizará toda la conducción con mínimo riesgo en sitios donde el sistema de operación tenga sus respectivos mapas, caminos delineados y las señalizaciones son claras de leer. El conductor puede interferir en cualquier momento, pero el SMA se encargará de todo. En el nivel 5 el carro es 100% automatizado. El SMA puede realizar todas las operaciones diseñadas sin ninguna limitación. No se requiere ningún tipo de asistencia de un humano, ya que no es necesario el uso de un volante o pedales. La siguiente figura muestra una representación de los diferentes niveles de autonomía [10].

Figura 1.1

Niveles de navegación autónoma [2]



1.4.2 *Sistemas autónomos*

Para poder ejecutar programas de software y poder navegar entre ubicaciones sin un operador humano, los vehículos autónomos requieren el uso combinado de sensores, actuadores, sistemas de aprendizajes autónomos (Machine Learning) y algoritmos complejos. El enfoque para el desarrollo de un sistema autónomo se divide en 3 subtareas que se deben realizar: Percepción y localización, planificación, y control.

1.4.2.1 Percepción

Consiste en la adquisición de información del entorno en el que se encuentra el vehículo por medio de sensores como: LiDAR, radar, sensores ultrasónicos, cámaras térmicas o RGB.

El LiDAR específicamente es un tipo de radar que funciona enviando millones de pulsos de luz cada segundo mientras gira sobre su eje. Gracias a esto es que se puede generar de forma

dinámica un mapa en 2D o 3D dependiendo del tipo del sensor. Aunque la información nunca es perfecta, usualmente hay puntos que faltan, patrones que no se pueden reconocer y en el caso del sensor 2D, solo puede adquirir datos en el nivel en el que se encuentra [11] [12].

1.4.2.2 Localización

Localización consta en la aptitud de poder determinar el movimiento y posición del vehículo, por medio de sensores propio-receptivos como: IMU, INS, odómetros o incluso cámaras y radar LiDAR. Hay algoritmos que funcionan con ayuda del mapa los cuales pueden producir una localización altamente precisa, especialmente SLAM. SLAM permite utilizar el mapa mientras este está siendo construido en tiempo real. Este algoritmo maneja objetos que ya han sido observados por el sensor para poder estimar la posición del vehículo en el mapa junto con la ubicación de nuevos objetos. SLAM utiliza un modelo estadístico el cual utiliza la odometría del vehículo para remover el conflicto entre la predicción de donde los objetos pueden estar con su ubicación basada en la lectura de los sensores [11] [12].

1.4.2.3 Planificación

Consiste en la toma de decisiones sobre el destino al que debe llegar el vehículo, esto se lo realiza por medio de la elaboración de recorridos que permitan que el este no colisione con obstáculos estáticos y/o dinámicos. El sistema de detección de vía en tiempo real esta propuesto con un sistema principal de visión. Se utiliza un tipo de algoritmo de Deep Learning basado en convolución. El sistema detecta las líneas del camino marcadas y sus bordes para las curvas. Para el cálculo del recorrido se utiliza un tipo de optimización discreta, el cual calcula un numero finito de caminos con su respectivo costo, al final se decide por el camino con menor costo. Se utilizan imágenes de caminos rectos y curvos donde están marcados los caminos propuestos [13].

1.4.2.4 Control

Permite que el vehículo pueda seguir la trayectoria planificada por medio de los actuadores necesarios. Un controlador PID es un tipo clásico de control, en cambio los más modernos tienden a ser lo modelos de control predictivos (MPC) o los que son a base de machine Learning como Reinforced Learning (RL) [13]. Acorde con Dang Dongfang et al. se sugieren tres formas de abarcar el diseño de planificación de control de movimiento: Campo potencial artificial (APF), método basado en muestreo y el método de control óptimo [14].

1.4.3 *ROS2*

Una de las infraestructuras de software más utilizadas para trabajar con vehículos autónomos es Robot Operating System (ROS). ROS es provee las herramientas requeridas para poder acceder con facilidad información de sensores, procesarla y poder generar una "respuesta" adecuada al actuador necesario del robot. Esta infraestructura es aplicada primordialmente a robots, pero un vehículo autónomo se lo puede interpretar como otro tipo de robot, por lo tanto, también se pueden utilizar los mismos programas para poder controlarlos [15].

1.4.4 *Estado del arte*

F1TENTH es una organización fundada originalmente en la universidad de Pensilvania en el 2016. El modelo que ellos proponen es basado en la infraestructura de ROS con el enfoque para control autónomo de vehículos de carrera y es Open Source [6]. La lista de materiales que se requieren deben ser compradas por el cliente, pero el software y los pasos para instalar y utilizar los programas los provee el equipo mismo. El proyecto está diseñado para que el vehículo pueda conducirse de forma autónoma y de forma manual por un cliente, utiliza visión por computadora

para percibir su entorno y poder identificar objetos, un ambiente virtual en el cual los clientes pueden probar sus propios programas, entre otros.

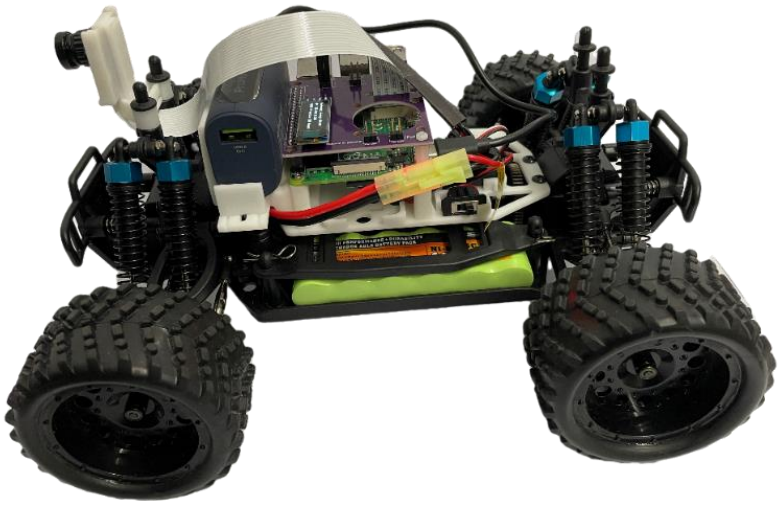
Figura 1.2

Modelo F1TENETH [6]

Donkeycar es una plataforma Open Source para vehículos a escala creada por la comunidad "DIY Robocars" ubicada en Berkeley, California. El modelo que proponen es basado en librerías de Python escritas por ellos mismos. Utiliza como CPU una Jetson Nano/Raspberry Pi junto con una cámara para poder navegarse a través de una pista de carreras de forma autónoma. El cliente primero controla manualmente el vehículo, durante esta acción por medio de Tensorflow e imágenes que se guardan durante el recorrido, el algoritmo programado lo utiliza para "aprender" como conducirse solo. La plataforma también ofrece un simulador virtual para poder aprender sobre el vehículo en caso de no poder adquirirlo físicamente [16]. La plataforma ofrece un manual de instrucciones y lista de materiales junto con sus respectivos costos listos para la compra o se puede adquirir el kit completo directamente de los proveedores.

Figura 1.3

Modelo Donkey car [16]



Rosmaster R2 es un vehículo autónomo desarrollado utilizando la infraestructura de ROS. Puede implementar como CPU la Jetson Nano, TX2 NX, Jetson Xavier NX y Raspberry PI 4B.

Viene incorporado con un radar laser (LiDAR), cámara de profundidad y módulo de reconocimiento de voz. Está programado con navegación para mapear, esquivo de obstáculos, conducción autónoma, reconocimiento de movimientos por parte del cliente, interacción por medio de comandos de voz, entre otros. Viene con su propia aplicación para teléfonos inteligentes, control manual por joystick inalámbrico o por teclado [17]. El costo de venta depende si el cliente quiere comprar con o sin el controlador (Jetson Nano). El distribuidor Yahboom ofrece en su repositorio un manual paso a paso sobre como instalar sus paquetes de control.

Figura 1.4

Modelo ROSMASTER R2 [17]

Multi-agent System for non-Holonomic Racing (MuSHR) es una plataforma Open Source diseñado con el fin de educación e investigación. Creado originalmente por el personal del laboratorio de robótica de la universidad de Washington. Viene incorporado con un radar laser (LiDAR), un controlador de velocidad electrónico para el motor, una Jetson Nano, un convertidor reductor, un joystick inalámbrico. El proveedor ofrece una lista de materiales con sus respectivos costos, archivos para impresión 3D y su propio software para manejo del vehículo [8].

Figura 1.5

Modelo MuSHR [8]

Qcar es un modelo de vehículo a escala diseñado para investigación académica. Tiene incorporado un radar LIDAR, visión 360º, un sensor de profundidad, un sensor IMU, encoders y utiliza como CPU una NVIDIA Jetson TX2. Viene con un software desarrollado por la compañía Quanser, el cual le permite al cliente realizar investigaciones sobre mapeo, navegación, machine Learning, inteligencia artificial, entre otros [5].

Figura 1.6

Modelo Qcar [5]

Capítulo 2

2. Metodología.

En el presente capítulo se describen las alternativas de solución para el problema establecido en el capítulo anterior y se elige una solución a desarrollar que más se apegue a los criterios de diseño. Se describe de manera más detallada la solución propuesta con énfasis en el diseño mecánico, electrónico y computacional, puntualizando las decisiones de diseño y los componentes que se utilizaran.

2.1 Criterios del diseño.

Para que un vehículo pueda navegar autónomamente en un espacio determinado, se detallaron en la siguiente tabla los criterios para su infraestructura, que describe los elementos que deben tener, las tareas que debe realizar y las funciones que se le pueden agregar.

Tabla 2.1

Criterios del diseño del proyecto

Concepto	Determinaciones
Escalabilidad	El sistema electrónico del vehículo debe ser capaz de poder adaptarse a cualquier elemento que el cliente quiera o pueda incorporar aparte de los elementos necesarios para su funcionamiento.
Robustez	El vehículo debe ser capaz de poder conocer el ambiente en el que se encuentra para poder realizar la navegación de forma autónoma lo más eficiente posible.
Robustez / Experiencia de cliente	El vehículo debe ser capaz de poder ser manejado manualmente por control remoto por el cliente y autónomamente.
Robustez	El vehículo debe ser capaz de evitar autónomamente obstáculos que se encuentren en su camino.
Robustez	El vehículo debe tener una estructura de manejo de tipo Ackermann.
Experiencia del cliente	El cliente debe poder experimentar el vehículo y sus funciones por medio de un simulador.

2.2 Soluciones.

2.2.1 *Alternativas y selección de solución.*

Una vez establecidos los criterios del diseño y la investigación previa sobre los diferentes modelos de vehículos de navegación autónoma descritos en el estado del arte del capítulo 1, se determinaron las siguientes alternativas para la solución:

- **Alternativa 1:** El modelo F1TENTH utiliza una Jetson Xavier NX como procesador. Un LiDAR 2D de marca HOKUYO, el cual tiene una distancia de detección de máximo 30 metros [18]. Un controlador electrónico VESC para el motor de velocidad, el cual le permite al cliente configurar los RPM, corriente y voltaje para el control [19]. Finalmente, una cámara de profundidad Intel RealSense D345i, la cual tiene un IMU implementado [20].
- **Alternativa 2:** El modelo Mushr utiliza una Jetson Nano como procesador. Un LiDAR 2D de marca YLIDAR, el cual tiene una distancia de detección de máximo 11 metros [21]. Un controlador electrónico VESC para el motor de velocidad, el cual le permite al cliente configurar los RPM, corriente y voltaje para el control [22]. Finalmente, una cámara de profundidad Intel RealSense D345i, la cual tiene un IMU implementado [20].
- **Alternativa 3:** El modelo Rosmaster R2 de Yahboom utiliza una Jetson Nano B01 como procesador. Un LiDAR 2D de marca YLIDAR, el cual tiene una distancia de detección de máximo 11 metros [21]. Una placa de control con interfaces para motores con encoder, motores PWM, luces RGB entre otros [23]. Finalmente, una cámara de profundidad Astra Pro.

2.2.2 *Selección y norma de la selección.*

Para poder seleccionar la alternativa más viable que se asemeje a los criterios de diseño concretados se tuvo en cuenta las siguientes normas:

- **Robustez:** Se evaluó la complejidad de las funciones que debe realizar el vehículo de forma autónoma y de con la participación del cliente.
- **Escalabilidad:** El cliente debe tener la libertad y facilidad de poder aumentar o añadir elementos alternativos que desee para utilizarlos en su versión del modelo.

- **Costo final:** Se evaluó los costos de los elementos básicos que requiere el producto, los cuales el cliente debe ser capaz de financiar.
- **Accesibilidad:** La dificultad para poder adquirir los elementos necesarios, como la complejidad de instalación del software deben ser de forma comprensibles o sencillas para el cliente.
- **Experiencia del cliente**: El cliente debe ser capaz de poder comprender y utilizar correctamente el software implementado en el vehículo.

En la siguiente tabla se ordenarán los criterios acordes a su relevancia y se les asignarán sus pesos para consideración.

Tabla 2.2

Criterios de alternativas de solución junto con su relevancia

Criterio	Relevancia	Peso
Robustez	1	5
Escalabilidad	2	6
Experiencia de cliente	3	5
Accesibilidad	4	4
Costo final	5	2

En la siguiente tabla se realizó una matriz de decisión, la cual se usó para comparar las alternativas de solución a base de los criterios de diseño establecidos previamente y con su respectivo peso, de esta manera se pudo determinar cuál sería la mejor opción para el desarrollo del proyecto.

Tabla 2.3

Matriz de decisión para determinar la selección de solución alternativa.

	Peso	Opciones	Alternativa 1	Alternativa 2	Alternativa 3
Criterio	6	**Escalabilidad**	1	1	3
	5	**Robustez**	3	2	3
	2	**Costo final**	1	2	2.5
	4	**Accesibilidad**	2.5	3	2.5
	5	**Experiencia de cliente**	2	2	2
Resultados		**Puntaje sin peso**	9.5	10	13
		Puntaje con peso	40	42	59.5
		Prioridad	**3**	**2**	**1**

Debido a los resultados obtenidos, se pudo concluir que el modelo de framework propuesto por la alternativa 3 es el mejor para la solución, debido a que la placa de control que utiliza, ya le puede dar al cliente la libertad de poder añadir más periféricos al sistema y poder experimentar con ellos. Esto le da un alto nivel de escalabilidad y robustez al diseño. El costo final es también un factor relevante a considerar, debido a que el modelo propio elaborado para el mercado tiene un costo de $865 [17], el cual puede ser reducido aún más dependiendo de los sensores que se usen.

A pesar que tanto la alternativa 1 y 2 tienen un alto nivel de robustez, sus sistemas están diseñados para trabajar exactamente con los elementos que tiene, no tienen una alta escalabilidad. El costo final en ambos casos resultaba con valores muchos más altos. Según la plataforma en línea del modelo de la alternativa 1, se puede estimar un costo total de $3,478 [6]. Esto se debe a

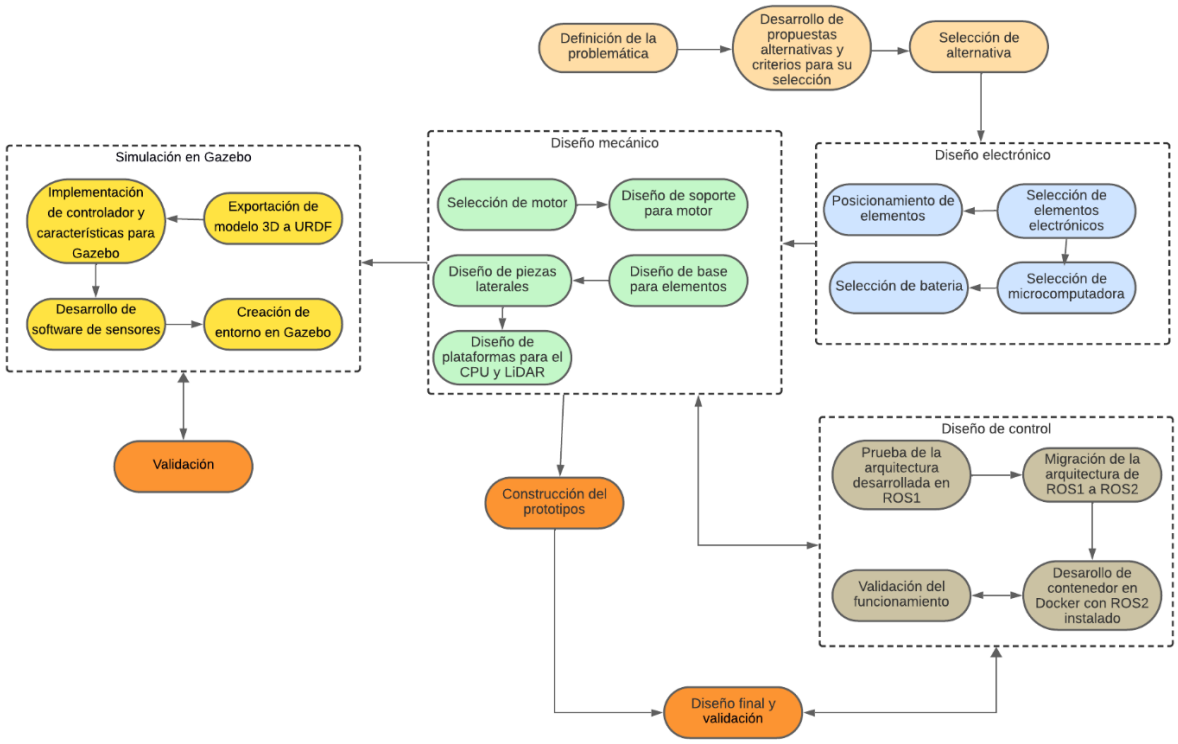
que los elementos más caros son el LiDAR Hokuyo 10LX y la Jetson Xavier NX. En cambio según la plataforma en línea del modelo de la alternativa 2, se puede estimar un costo total de $2,090 [8]. Esto se debe a que los elementos más caros son el YDLidar que implementan y la cámara de profundidad Realsense D435. Ambas alternativas están diseñadas para ser lo más accesibles posibles con alta experiencia para el cliente, pero el costo si es un factor relevante para motivar a los potenciales clientes en conseguir un producto de este tipo.

2.3 Metodología del diseño.

Para el desarrollo del diseño final, se realizaron diferentes etapas, cada una enfocada en un aspecto complementario del proyecto. En la Figura 2.1 se puede observar con más detalle el orden en el que se abordó cada etapa. Primero se definió la problemática a afrontar. Luego se desarrolló una lista de propuestas alternativas para la solución, después de considerar los criterios de elección y la prioridad que se les darán a estos, se escogió la solución más óptima. El proceso de diseño comenzó con la etapa electrónica y a base de esta, se desarrolló el diseño de piezas de la estructura para acoplar al chasis del vehículo a escala. Una vez realizado el diseño conceptual de la solución se realizó el diseño de control y programación, el cual tuvo que pasar por un proceso de validación para poder garantizar su efectividad. Aparte, una vez concluido el diseño de las piezas del prototipo, se generó un modelo 3D del prototipo final para su implementación en un entorno virtual de simulación en Gazebo, el cual una vez diseñado y programado, se realizaron pruebas de validación para garantizar su funcionamiento.

Figura 2.1

Proceso del diseño

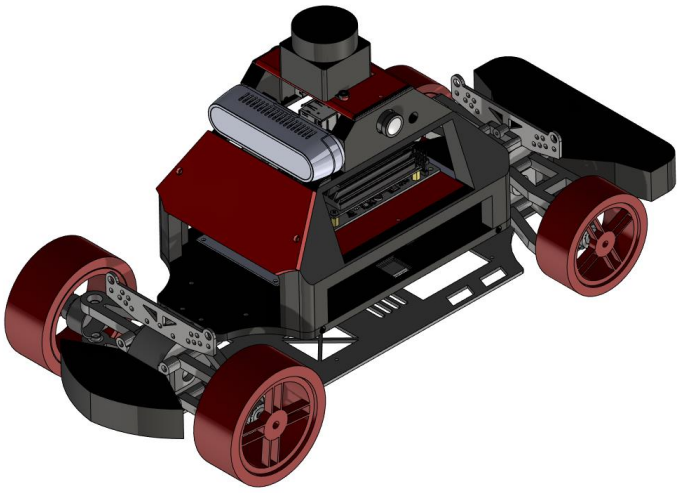


2.4 Diseño conceptual

En la Figura 2.2 se presenta la idea principal del diseño que debe tener las plataformas en donde se posicionaran los elementos electrónicos. Para el desarrollo del modelo más apropiado, se dimensionaron tanto el chasis en el que se colocarán las piezas, como los elementos electrónicos que se utilizarán. Se debe garantizar en el diseño que el cliente pueda colocar de forma secuencial los periféricos de forma segura. La estructura ya ensamblada debe proveer la mayor rigidez posible garantizando la seguridad de los periféricos. También se considera que el manejo de cables sea lo más ordenado posible.

Figura 2.2

Diseño conceptual de las piezas para ensamble



Nota. La figura muestra la versión inicial del diseño de las piezas que se emplearan para ensamblar los periféricos electrónicos.

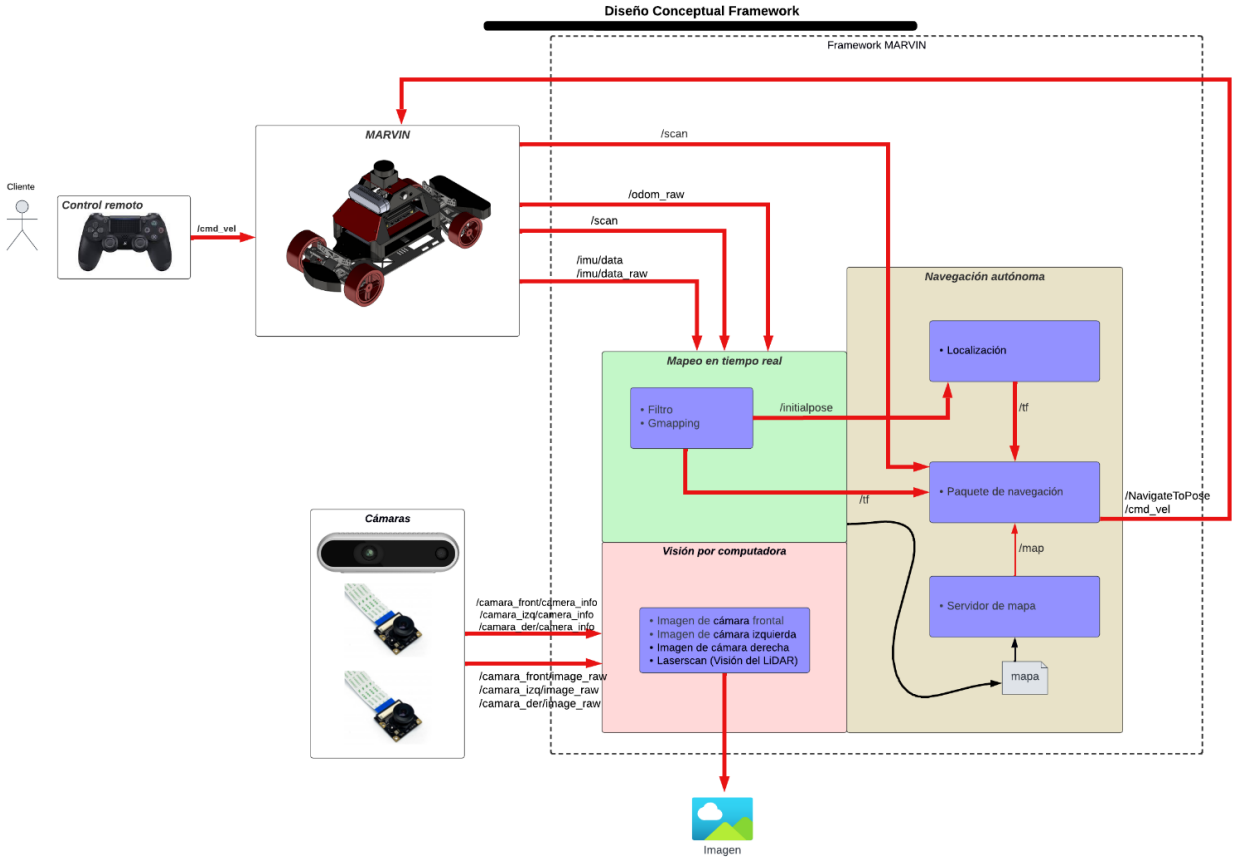
El framework de MARVIN consiste en mapeo en tiempo real, navegación autónoma y visión por computadora. La Figura 2.3 muestra el diagrama conceptual del framework implementado. Las entradas del sistema son los comandos que ingresa el operador, en este caso son los de control manual del vehículo. Los tópicos de entrada para el mapeo en tiempo real provienen del sensor IMU, del odómetro y del LiDAR. Primero el operador debe realizar un mapeo del ambiente en el que se encuentra el vehículo y guardarlo en un archivo. Luego para el módulo de navegación autónoma, el tópico de entrada es la información del mapa guardado, la información que proviene del LiDAR (en caso de que se encuentre un nuevo obstáculo que no se grabó previamente), la transformación de la localización del robot en el ambiente y la/s posición/es que este recorrerá. La salida del módulo de navegación consiste en los comandos de velocidad que se enviaran a los controladores de los actuadores, la distancia restante entre el robot y su posición

final. La salida del módulo de navegación son los de velocidad y de posición final que se envían al controlador del robot, para que pueda realizar los cálculos de las trayectorias.

Además, los tópicos que publican las cámaras (frontal y laterales) contienen información sin procesar. Estas se publican como imágenes para poder obtener una visión en tiempo real desde la perspectiva del vehículo. También la visión de computadora se subscribe al tópico escaneo del LiDAR, de esta manera siempre se tiene una visión completa del vehículo en todo momento. navegación

Figura 2.3

Framework de MARVIN



2.5 Parámetros del diseño electrónico.

El diseño electrónico es el primer aspecto que se cubre al momento de realizar el producto, ya que el vehículo requiere de periféricos específicos que le permitan navegar de forma autónoma y poder realizar las acciones de percepción, localización, planificación y control.

2.5.1 *Selección de sensores para control del vehículo.*

Para realizar la operación de percepción y localización se requerirán elementos como escáneres laser y cámaras para la visión.

Para poder generar un mapa en 2D se requerirá un LiDAR que cumpla con los siguientes requerimientos:

- **Cobertura y precisión:** Dado que es un vehículo a escala, se requerirá un rango entre 3-15 m de cobertura.
- **Rango de escaneo:** Debe tener un rango de 360º.
- **Integración y montaje:** Su tamaño debe ser lo suficientemente pequeño y fácil de montar para poder adaptarlo al vehículo.
- **Compatibilidad y software:** Debe ser compatible con ROS2.

El LiDAR LD19 que se muestra en la Figura 2.4 demuestra las siguientes características:

Figura 2.4

LiDAR_LD19

Tabla 2.4

Información del LiDAR_LD19

LiDAR_LD19			
Rango de detección	3m~12m	**Apoyo para ROS**	ROS1/ROS2
Frecuencia de medición	4500 Hz	**Dimensiones**	35.59*38.59*34.8 mm
Angulo de medición	360º		

Nota: La tabla de información del LiDAR es original de la página de venta en Amazon. [24]

Para que un robot con pueda realizar reconocimiento de imagen y pueda navegar con mayor precisión en un entorno, se debe implementar una cámara de profundidad, la cual debe cumplir las siguientes características:

- **Rango de profundidad y precisión:** El rango de profundidad debe cubrir distancias relevantes para el vehículo.
- **Resolución:** A mayor resolución, los mapas tendrán más detalle.
- **Integración y montaje:** Su tamaño debe ser lo suficientemente pequeño y fácil de montar para poder adaptarlo al vehículo.

La cámara de profundidad Intel Realsense D435 que se muestra en la Figura 2.5 demuestra las siguientes características:

Figura 2.5

Intel Realsense D435

Tabla 2.5

Información de la cámara de profundidad

Intel Realsense D435			
Rango de detección	10 m	**Resolución máxima de video**	1920 x 1080
Resolución RGB	1920x1080	**Dimensiones**	900*25*25 mm
Rango de fotograma en RGB	30 fps	**Mecanismo de montaje**	2 puntos M3, Trípode

Nota: La tabla de información de la cámara de profundidad es original de la página de venta en Amazon. [25]

Para el control y procesamiento de los sensores y actuadores del vehículo, se requiere una computadora relativamente compacta pero robusta. También debe ser capaz de poder ser compatible con plataformas de inteligencia artificial.

La Jetson Nano Developer Kit que se muestra en la Figura 2.6 demuestra las siguientes características:

Figura 2.6

Jetson Nano Developer Kit

Tabla 2.6

Información de la Jetson Nano

Jetson Nano Developer Kit			
CPU	Quad-core ARM A57 @ 1.43GHz	**GPU**	Maxwell 128 núcleos
Almacenamiento	Soporte de ranura para tarjetas Micro-SD	**Tamaño**	100x80x29 mm

Nota: La tabla de información de la Jetson Nano es original de la página de venta en Amazon. [26]

Para el control de los actuadores y con la posibilidad de poder agregar más periféricos al sistema se requiere la placa de expansión de Yahboom. Esta placa tiene un controlador adaptado a ROS.

Figura 2.7

Placa de control Yahboom

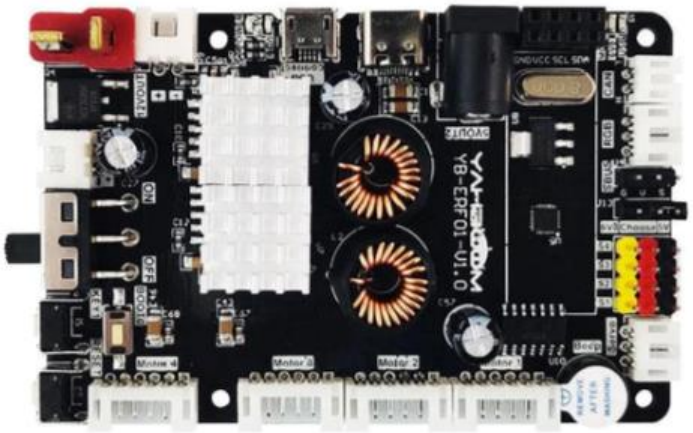


Tabla 2.7

Información del controlador de actuadores

Placa de control Yahboom			
Canales para motores	4- Servo 4- Encoder	**Alimentación**	12 V
IMU	Sensor IMU con 9 ejes	**Apoyo para ROS1/ROS2**	Si

Nota: La tabla de información de la placa de control Yahboom es original de la página de venta Yahboom. [23]

Para la distribución más efectiva energía hacia la Jetson Nano y para evitar que esta consuma más energía de lo que requiere, se seleccionó una placa de expansión USB. Esta es compatible con la placa de control.

Figura 2.8

Placa de expansión USB

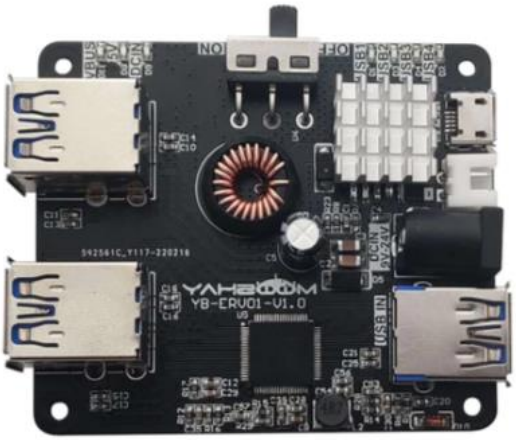


Tabla 2.8

Información de la placa de expansión USB

Placa de expansión USB			
Interfaz USB	USB 3.0	**Alimentación**	9-24 V
Compatible con placa de control Yahboom	Si		

Nota: La tabla de información de la placa de expansión USB es original de la página de venta Yahboom. [27]

2.5.2 *Selección de fuente de energía.*

Debido a que la energía de todo el sistema es distribuida por la placa de control Yahboom, la fuente de energía que se debe elegir debe seguir las siguientes características:

- **Voltaje requerido de entrada:** DC 12V con entrada de tipo T.
- **Capacidad:** (mAh): 1800 – 6000 mAh.
- **Tamaño y peso:** 100-300 g / ~13*4.5*2.8 mm.

- **Reusabilidad:** La batería debe ser recargable para que el producto pueda ser reusado múltiples veces.

Figura 2.9

Batería Li-PO

Tabla 2.9

Información la batería LIPO

Batería LIPO			
Voltaje	11.1 V	**Dimensiones**	88*35*26 mm
Capacidad	1800 mAh	**Peso**	106 g

Nota: La tabla de información de la batería LIPO es original de la página de venta ROBOTS. [28]

2.6 Parámetros del diseño mecánico.

Para poder obtener el diseño más optimizado se ha considerado los siguientes parámetros mecánicos: dimensiones de cada pieza, posicionamiento de los elementos electrónicos, comodidad a la hora de ensamblar las piezas junto con los periféricos, rigidez de la estructura. Aparte se ha ideado que el diseño debe ser atractivo para el cliente. El diseño está desarrollado con el objetivo de que el cliente pueda ir ensamblando secuencialmente toda la estructura (periféricos incluidos) y poder colocarla en la parte superior del chasis del vehículo.

2.6.1 *Selección de actuadores para el movimiento del vehículo.*

El vehículo debe ser capaz de navegar con una velocidad variable en el entorno mientras realiza un mapa virtual en tiempo real, por lo tanto, el motor debe un encoder de alta precisión incorporado. Se prefiere un motor que tenga un balance entre torque y velocidad, este tipo de motores tienen un rango de reducción de engranes de 1:30. La siguiente tabla muestra las características del motor elegido.

Figura 2.10

Motor 520 con encoder

Tabla 2.10

Información del motor Hall 520 con encoder

Motor Hall 520 con encoder			
Rango de voltaje	12V	**Corriente stall**	0.3 A
Tipo de motor	Motor de corriente continua con escobillas de imán permanente	**Voltaje de alimentación del encoder**	3.3-5V
Stall torque	4.8 kg*cm	**Rango de reducción**	1:30
Rango de torque	3.3 kg*cm	**Tipo de interface**	PH2.0 6Pin
Rapidez antes de la desaceleración	11000 rpm		

Nota: La tabla de información del motor 520 es original de la página de venta del modelo ROSMASTER R2 de Yahboom. [17]

Para el manejo del manubrio se debe seleccionar un motor servo que pueda ser introducido en su espacio designado en el chasis. El tiempo de respuesta del motor debe ser rápido y debe ser controlado con un alto nivel de precisión. La siguiente tabla muestra las características del motor elegido.

Figura 2.11:

Motor servo

Tabla 2.11

Información del motor Servo

Motor Servo			
Rango de voltaje	4.8 – 6.8V	**Par de torsión (5V)**	41.9 lbs/cm
Tipo de motor	Motor de corriente continua	**Par de torsión (6V)**	47.4 lbs/cm
Tamaño	40.00x20.00x 40.50 mm	**Modo de control**	Pulso PWM con control de ancho

Nota: La tabla de información del motor servo es original de la página de venta del motor en Amazon. [29]

2.6.2 *Consideraciones de diseños de piezas para ensamble.*

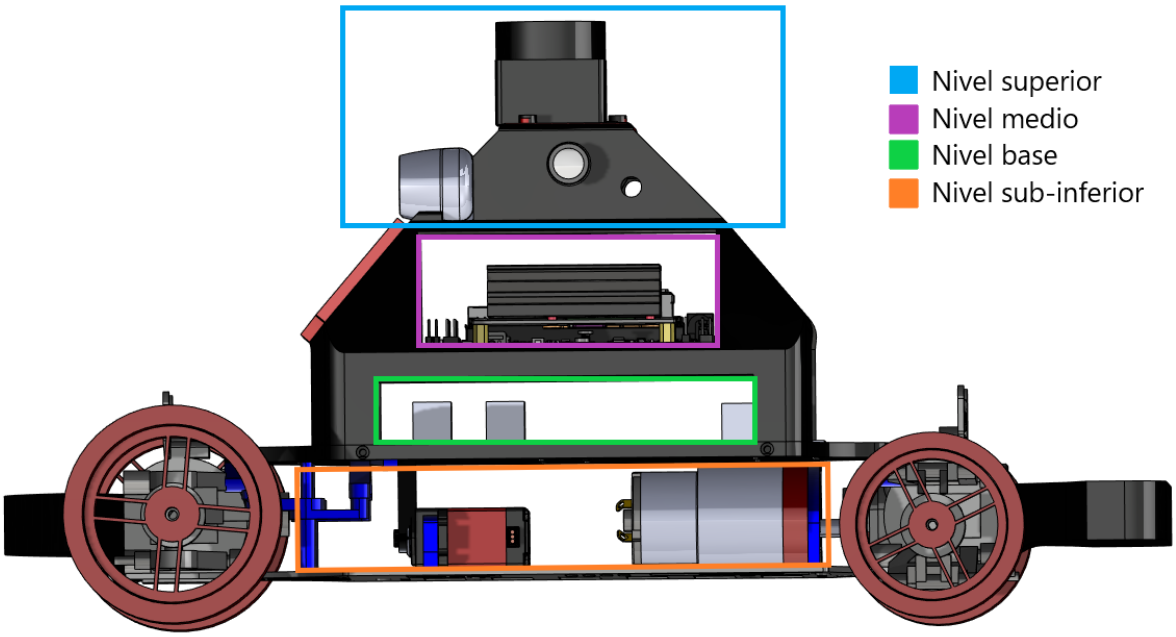
Para poder desarrollar un modelo que ira montado sobre un chasis de vehículo, se deben considerar los siguientes aspectos: las medidas exactas de la estructura del chasis y el cliente debe poder por medio de manufactura aditiva imprimir la pieza con el diseño final sin que se pierdan los detalles.

El prototipo tendrá cuatro niveles:

1) Nivel sub-inferior (naranja): En este nivel estarán colocados el motor que manipule el manubrio y el motor de avance del vehículo.
2) Nivel base (verde): En este nivel estarán colocadas las placas de control para actuadores.
3) Nivel medio (morado): En este nivel debe estar el controlador principal (la computadora) del sistema.
4) Nivel superior (azul): En este nivel deben estar colocados los periféricos de reconocimiento del vehículo (LiDAR, cámara de profundidad y cámaras laterales).

Figura 2.12

Niveles del vehículo donde se colocarán los elementos electrónicos



Debido a que el motor de avance del vehículo (motor Hall 520 con encoder) no se acopla directamente en el chasis, se debe diseñar una pieza que lo haga encajar de tal forma que el eje junto con su engrane con dientes puedan entrar en el espacio delimitado y también el diseño de esta pieza debe permitir que el motor pueda deslizarse sobre el eje del soporte del chasis y poder permitir al cliente cambiar el engrane con dientes sin la necesidad de tener que desarmar toda la estructura.

En el nivel base (verde) está diseñado para colocar la placa de control y la placa de expansión USB de Yahboom.

En el nivel medio (morado) está diseñado para tener el controlador principal, en este caso será la Jetson Nano. La apertura debe ser las dimensiones necesarias para que los puertos USB sean de fácil acceso y también para que los cables que conectan con las cámaras IMX no estén tan comprimidas.

En el nivel superior (azul) es donde estarán posicionadas las cámaras IMX en ambos lados y la cámara de profundidad Realsense a la misma altura, de esta forma se puede tener una visión de 180º que no se vea interrumpida por otros elementos. El LiDAR se lo coloca en la parte más alta, para que, a la hora de realizar el mapeo, no tenga ningún obstáculo bloqueándolo.

Se estima que el prototipo final tendrá una altura de 150 mm, mientras que las cámaras estarán posicionadas a aproximadamente 90 mm de altura.

2.7 Parámetros del diseño de control

2.7.1 *Control del vehículo a escala*

La inicialización del vehículo debe activar todos los nodos necesarios para el control del vehículo. El vehículo puede ser controlado manualmente por medio de un joystick o por teclado. El nodo de control (/joy_ctrl) publica tópicos de velocidad (/cmd_vel). El nodo que manipula todos los aspectos de manejo del vehículo (/driver_node) se subscribe a estos tópicos.

Paralelamente se combina información del IMU y de la velocidad que publica la placa de expansión para generar por medio de una salida de datos de odometría, esta información se la utiliza al momento de utilizar las funciones de posicionamiento.

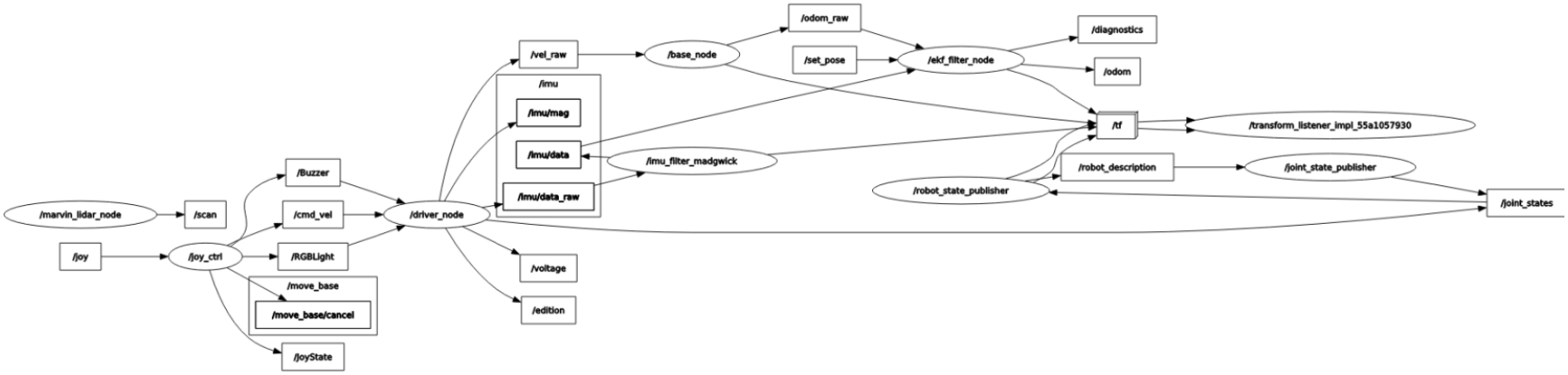
También se deben inicializar la visualización del modelo 3D del vehículo, junto con la visión misma del robot.

Como parte del proceso de iniciación, también se activa el LiDAR del vehículo, de esta manera todos los sensores necesarios para la navegación autónoma están disponibles desde el primer momento.

En la siguiente Figura 2.13 se muestran los nodos activos al momento de inicializar el vehículo y los respectivos tópicos que se publican.

Figura 2.13

Inicialización del vehículo a escala

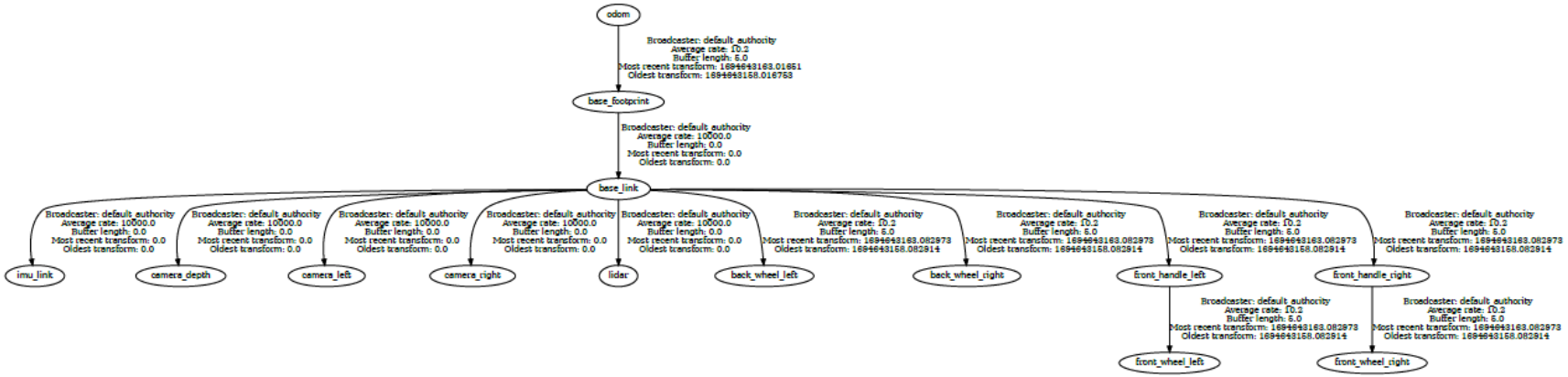


Nota. Figura obtenida a partir de la interfaz rqt diseñado para ROS.

El árbol de transformadas que se forma se lo muestra en la Figura 2.14.

Figura 2.14

Árbol de transformadas del vehículo a escala al activarse



Nota. Figura obtenida a partir de la interfaz rqt diseñado para ROS2.

2.7.2 *Mapeo y localización del ambiente*

Para que el vehículo pueda navegar en un ambiente, debe primero ser capaz de conocer su entorno. Para esto se utilizó la herramienta Slam Toolbox. Este paquete incorpora la información recopilada por el LiDAR y las transformaciones TF del link de origen concatenado con la base del robot (odom -> base_footprint) para poder generar un mapa en 2D de un espacio [30] . Por medio del modo “en línea asíncrono”, el cual le permite trabajar con un flujo de información en vez de datos guardados y siempre procesa el escaneo más reciente, para evitar retraso se genera un mapa en el ambiente.

Para la localización del vehículo se implementa el algoritmo probabilístico AMCL (Adaptive Monte Carlo Localization), el cual representa la posición y orientación del robot como una distribución de partículas, en las cuales cada una representa una posible posición/orientación del robot. Con cada movimiento, el robot toma medidas y las compara con el mapa del ambiente para determinar cuál de las partículas es la verdadera posición del robot. Este algoritmo es de bajo consumo de procesamiento en tiempo real [31].

2.7.3 *Navegación autónoma*

Para la navegación autónoma del vehículo se implementó el paquete de Nav2. Este framework de navegación utiliza behaviour tree (BT) en formato XML, las transformaciones TF, un mapa ya generado para utilizarlo como “mapa de costo” y valores de los sensores, en este caso el LiDAR, para poder proveer comandos de velocidad para los motores. Nav2 tiene apoyo para robots de tipo Ackermann, que es el tipo de vehículo del proyecto. El behaviour tree se los implementa para que se puedan crear comportamientos de navegación inteligentes [32]. La

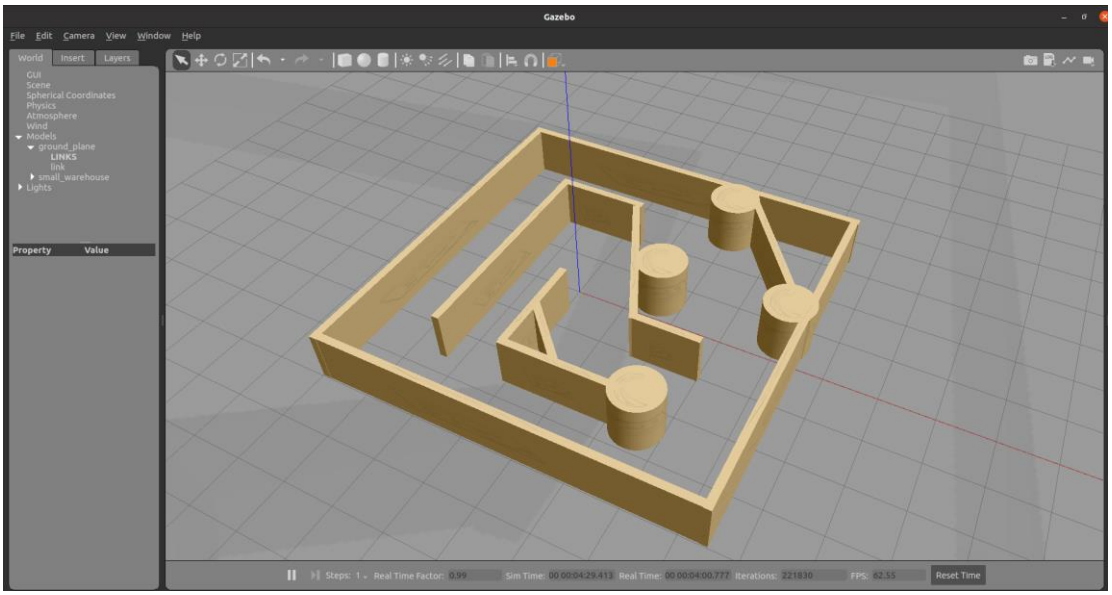
implementación de este framework permitirá que el vehículo pueda navegar autónomamente de una posición A a una posición B evadiendo los obstáculos que interfieran en su recorrido.

2.7.4 *Simulación y control del vehículo simulado*

La simulación del vehículo se la realizara en un ambiente de Gazebo. El control del vehículo tanto manual como autónomo tendrá la misma estructura que el vehículo a escala, pero adaptado para Gazebo, ya que la mayoría de implementación de los sensores y funciones son por medio de plugins ya establecidos, facilitando así el desarrollo.

Figura 2.15

Mapa de simulación en Gazebo



Nota: La figura muestra un ambiente que se utilizara para realizar pruebas en la simulación.

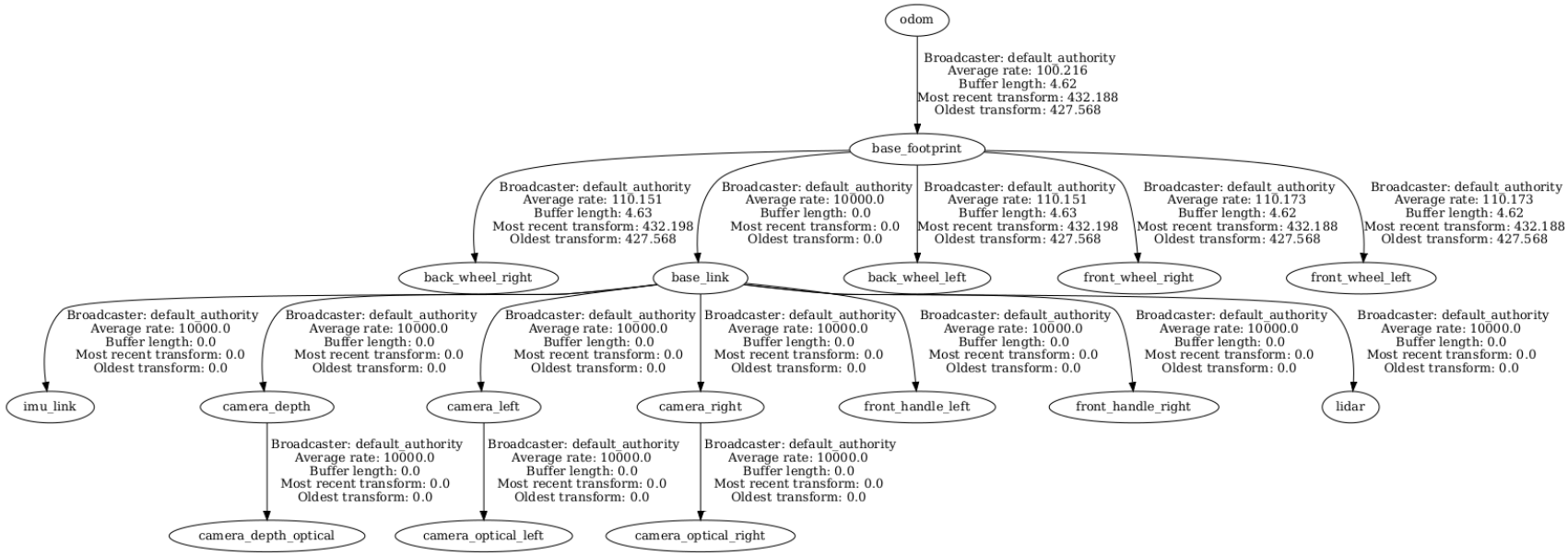
Figura 2.16

Control del vehículo y encapsulación de sensores

Nota: Figura obtenida de la simulación en Gazebo

Figura 2.17:

Árbol de transformadas del vehículo simulado



Nota: Figura obtenida de la simulación en ROS2.

Capítulo 3

3. Resultados y análisis

En esta sección se muestran los resultados obtenidos del framework MARVIN. Los resultados de cada módulo se expondrán de forma individual y luego se expondrán las diferentes funciones que puede realizar el robot con las funciones de cada módulo trabajando en conjunto. Finalmente se concluye la sección con el análisis de costos.

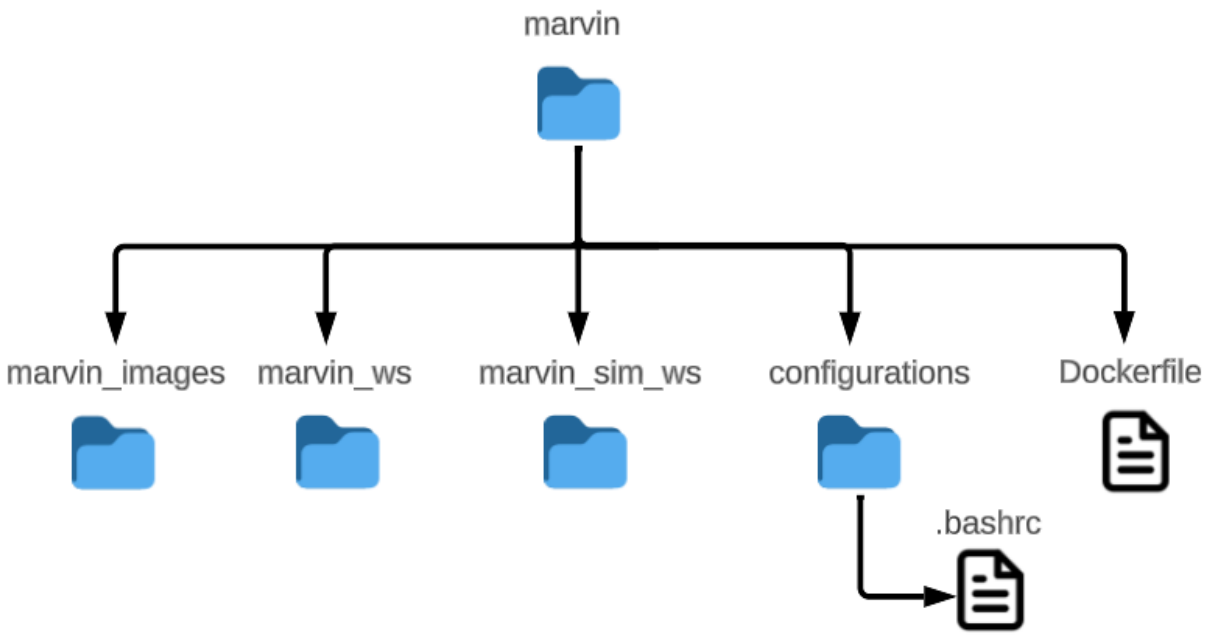
Para poder visualizar la topología de los nodos y la comunicación entre los componentes en ROS2 se utilizó la herramienta “Gráfico de nodos) de rqt.

3.1 Diseño del repositorio

Se diseñó un repositorio en GitHub con los paquetes necesarios para construir el framework de MARVIN tanto para la simulación como para el vehículo a escala. El repositorio está organizado para que el cliente se lo descargue a su máquina y/o directamente en la Jetson Nano. La carpeta marvin_ws contiene los paquetes para poder controlar el vehículo a escala, mientras que la carpeta marvin_sim_ws contiene los paquetes para la simulación. En el caso del vehículo a escala, el repositorio debe ser descargado directamente al /root de la Jetson Nano, luego construir el contenedor del Docker utilizando la imagen ubicada dentro del Dockerfile, este se encargará de crear el contenedor, copiar las carpetas del repositorio dentro del contenedor y editar el archivo. /bashrc del contenedor para que los paquetes de ROS2 funcionen correctamente. Dentro del contenedor, el cliente debe compilar 1 vez solo la paquetería de marvin_ws para poder utilizarlo.

Figura 3.1

Diseño del repositorio del proyecto



3.2 Mapeo del ambiente (vehículo a escala)

Para el modelo a escala (figura 3.2) se validó la navegación autónoma en un ambiente libre de obstáculos. Primero se inicializó el vehículo en una esquina del mapa se lo manipulo manualmente para que junto con Slam Toolbox pueda mapear su entorno de forma eficiente. El vehículo a escala se lo manejó a baja velocidad y evitar giros abruptos, ya que las herramientas implementadas dependen de la velocidad del internet de su ambiente. El mapa generado tiene un alto nivel de semejanza con el entorno real, en la siguiente figura se puede apreciar el entorno real junto con el mapa realizado por el vehículo a escala.

Figura 3.2

Prototipo real del vehículo a escala

Figura 3.3

Mapeo del entorno real

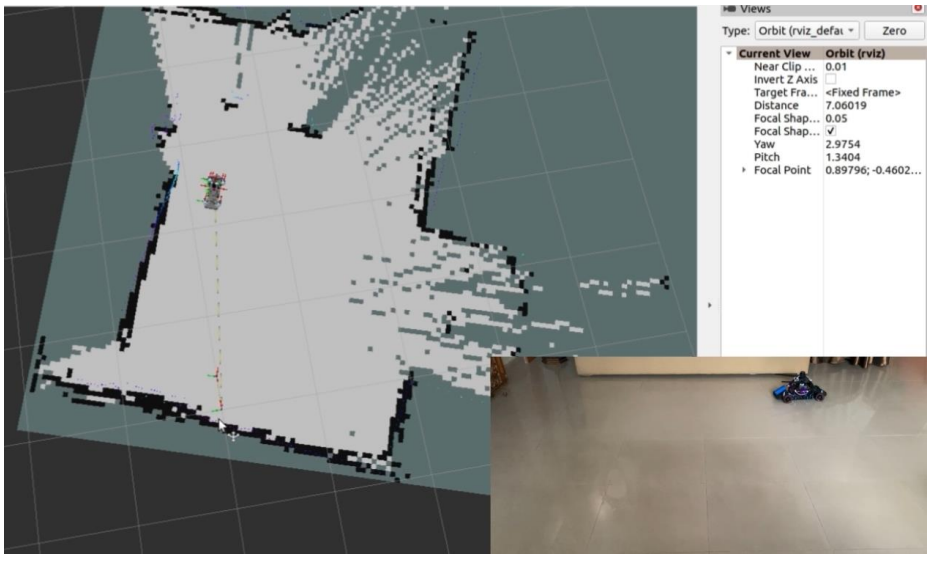
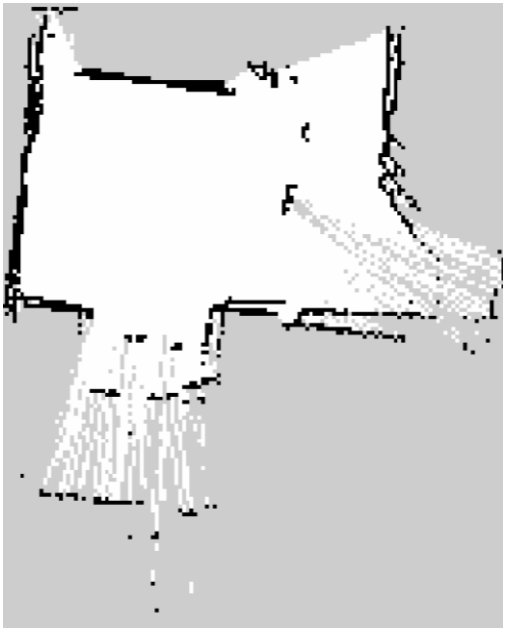


Figura 3.4

Mapa creado por Slam Toolbox



Nota. Mapa creado por el paquete Slam Toolbox

3.3 Navegación autónoma (vehículo a escala)

Cuando se guarda el mapa, el vehículo puede cambiar su posición dentro del mapa para poder ser localizado, por medio de amcl y la herramienta de rviz 2D Pose Estimate, se ubica y declara la posición y orientación del vehículo actualmente. Esto hace que el modelo aparezca en el mapa, listo para comenzar a navegar autónomamente.

Figura 3.5

Localización utilizando amcl

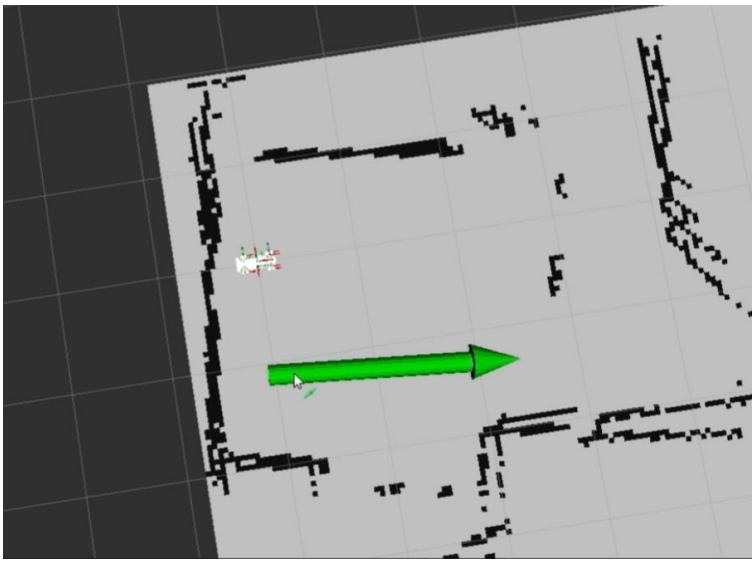


Figura 3.6

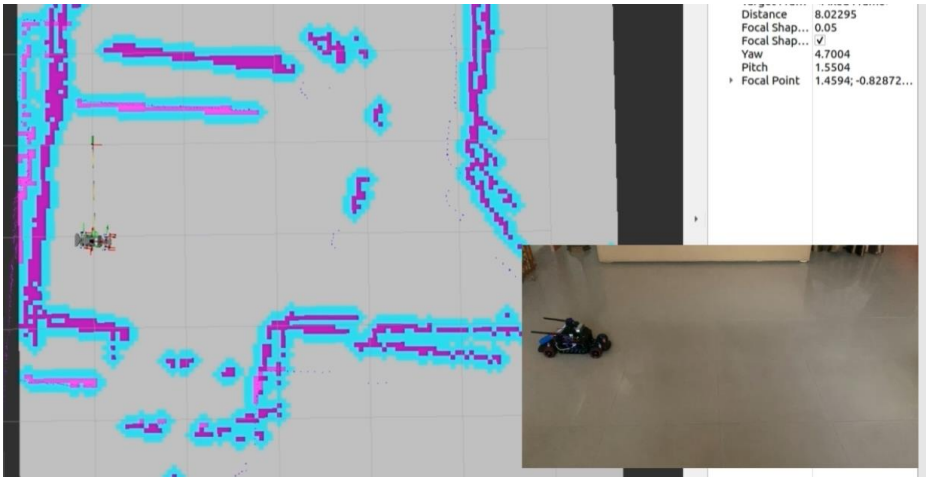
Localización real y ubicada por amcl

No siempre se logra una ubicación con 0% error, ya que se lo hace manualmente, pero la desviación que se puede generar se corrige de poco en poco una vez que el vehículo comienza a moverse en el mapa.

Para que el vehículo pueda navegar de forma autónoma, como explicado en el capítulo anterior, se empleara la herramienta Nav2. Esta generara un mapa de costo global, el cual delimitara los límites que el robot no puede acercarse, sino debe evitarlo.

Figura 3.7

Mapa de costos global

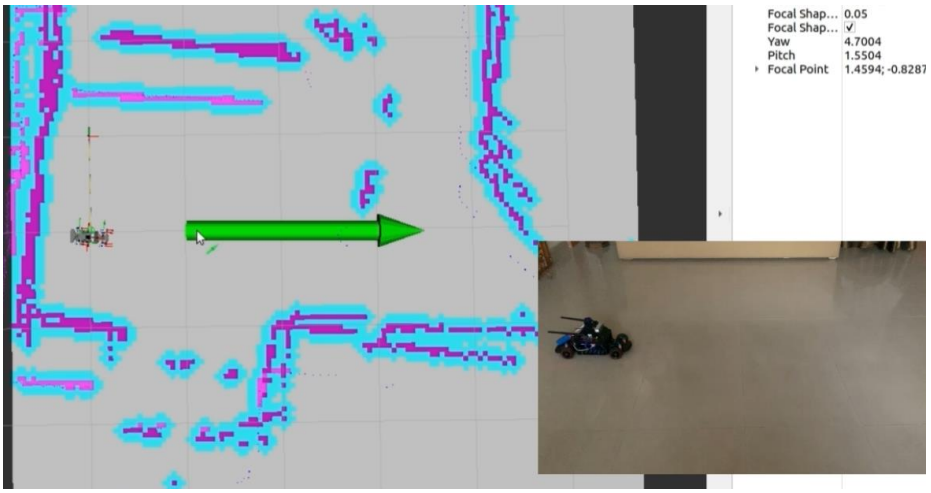


En la siguiente Figura 3.8 se muestra la una secuencia de navegación del vehículo de forma secuencial. Primeramente, se le asigna una posición y orientación final al vehículo con la herramienta 2D Goal Pose de Rviz. El vehículo calcula la ruta más eficiente que debe recorrer y la ejecuta.

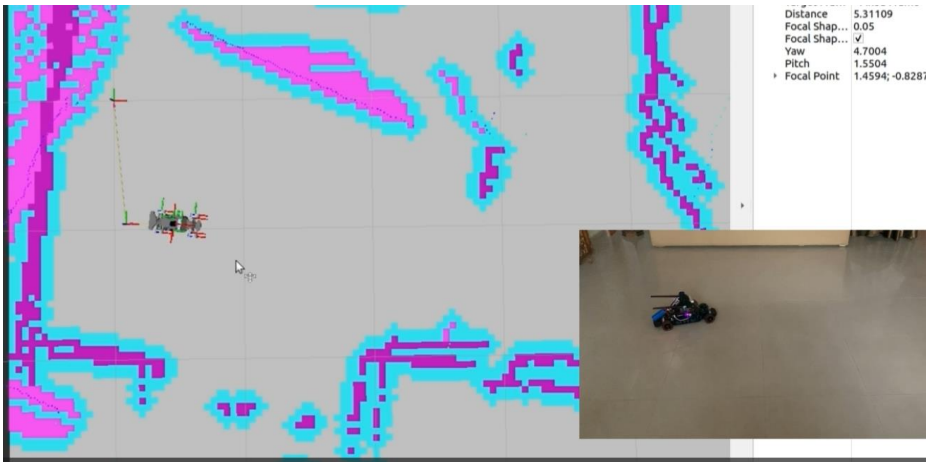
Figura 3.8

Navegación autónoma y evasión de obstáculos

a.



b.



Nota: a. El vehículo se le asigna una posición, b. El vehículo llega a su destino y se le agrega un objetivo a través de unos obstáculos,

3.4 Implementación de cámara frontal y laterales

Debido a que el framework funciona dentro de un contenedor creado en Docker, la instalación del SDK (software development kit) funciono mejor dentro del sistema propio de la

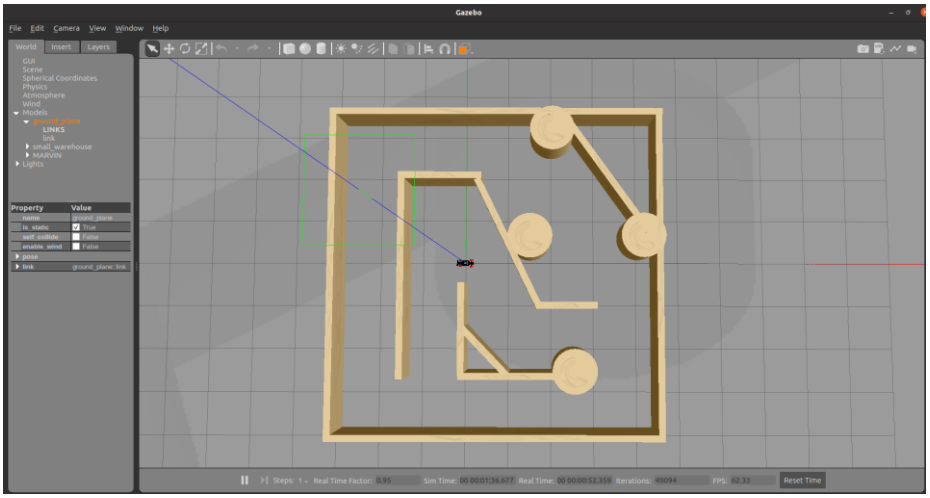
Jetson, pero dentro de un contenedor, produce inconsistencias, lo cual causo que no se pudieran activar las cámaras, tanto frontal como laterales.

3.5 Simulación y control del vehículo simulado

Se agrega el vehículo simulado en el ambiente de Gazebo. Como visto en el capítulo 2, se inicializa todos los componentes principales del framework al momento de agregar el modelo al ambiente.

Figura 3.9

Ambiente simulado en Gazebo



Una vez cargado todos los componentes, se continuo a hacer un mapeo del entorno utilizando Slam Toolbox, el proceso y resultado se los muestra en las siguientes figuras.

Figura 3.10

Ambiente simulado y SLAM activo

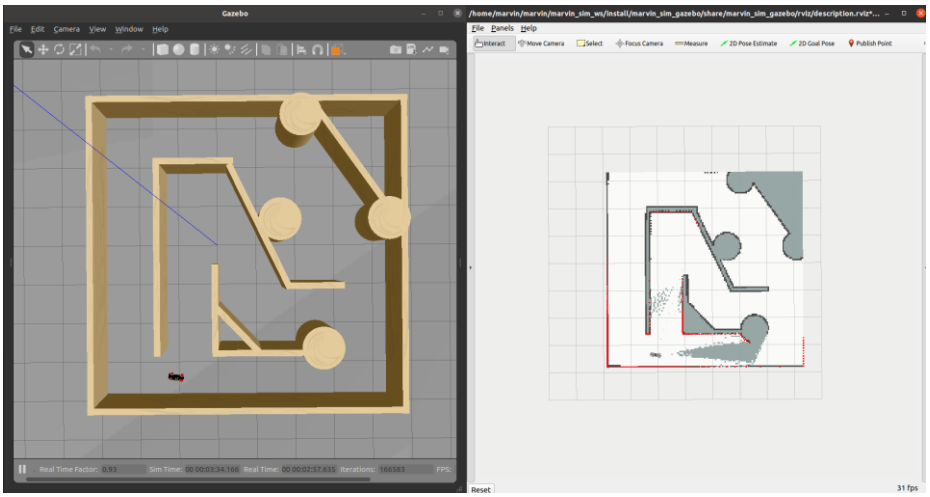
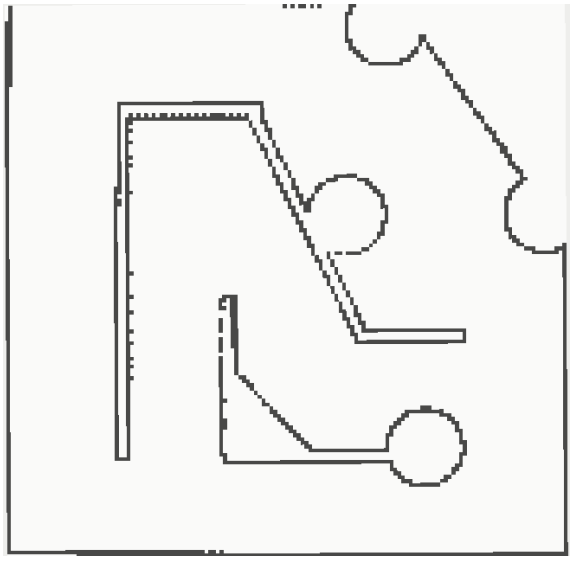


Figura 3.11

Mapa creado por Slam Toolbox



Nota. Mapa creado por el paquete Slam Toolbox

Cuando se guarda el mapa, el vehículo no tiene que estar en su punto de origen para ser localizado, en vez de eso, por medio de amcl y la herramienta de rviz 2D Pose Estimate, se ubica

y declara la posición y orientación del vehículo actualmente. Esto hace que el modelo aparezca en el mapa, listo para comenzar a navegar autónomamente.

Figura 3.12

Localización utilizando amcl

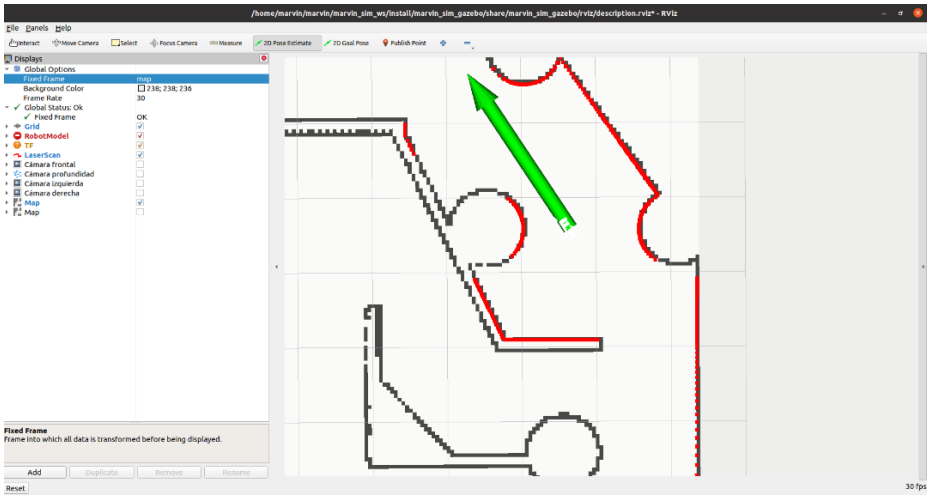
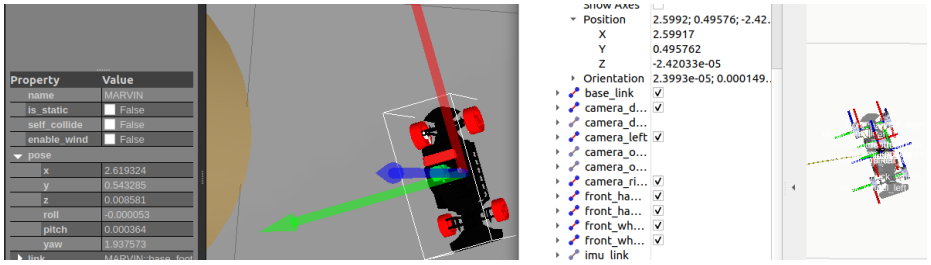


Figura 3.13

Localización real y ubicada por amcl

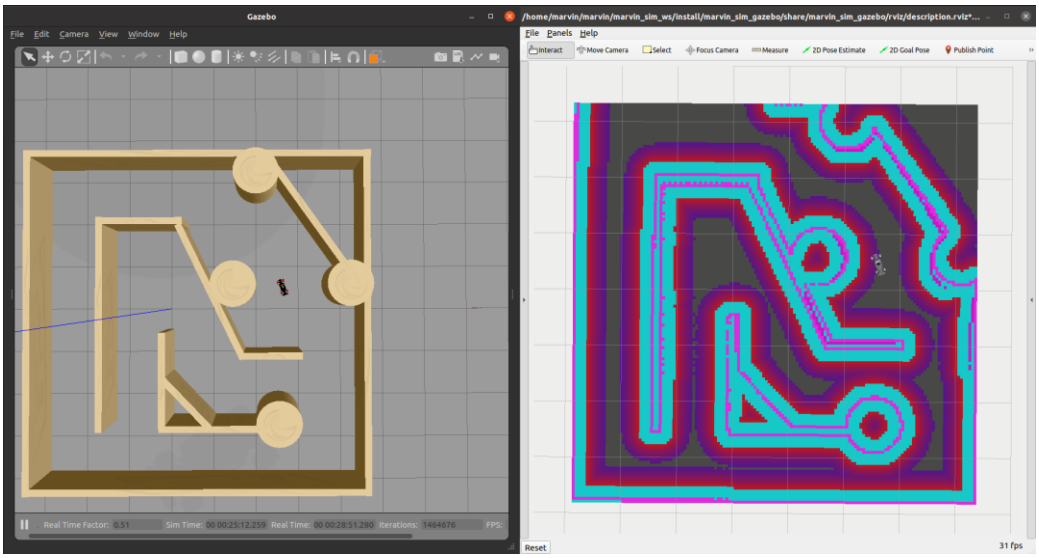


No siempre se logra una ubicación con 0% error, ya que se lo hace manualmente, pero la desviación que se puede generar se corrige de poco en poco una vez que el vehículo comienza a moverse en el mapa.

Para que el vehículo pueda navegar de forma autónoma, como explicado en el capítulo anterior, se empleara la herramienta Nav2. Esta generara un mapa de costo global, el cual delimitara los límites que el robot no puede acercarse, sino debe evitarlo.

Figura 3.14

Mapa de costos global

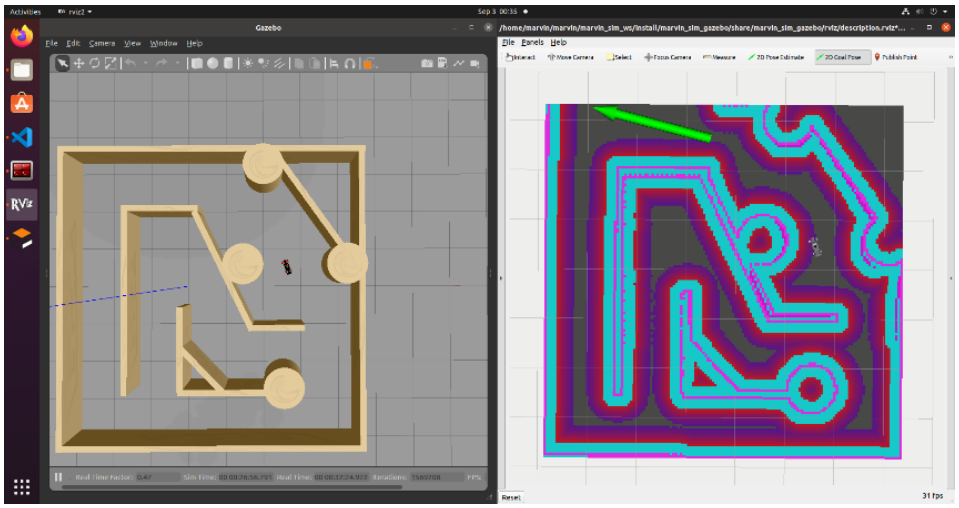


En la Figura 3.15 se muestra la una secuencia de navegación del vehículo de forma secuencial. Primeramente, se le asigna una posición y orientación final al vehículo con la herramienta 2D Goal Pose de Rviz. El vehículo calcula la ruta más eficiente que debe recorrer y la ejecuta. En el caso del a figura c. se le agrego una posición y orientación final al vehículo en el cual debe arribar sin chocarse con ningún obstáculo, cuando esto ocurre, el robot recalcula en tiempo real la dirección más óptima para ejecutarla y continuar con su objetivo

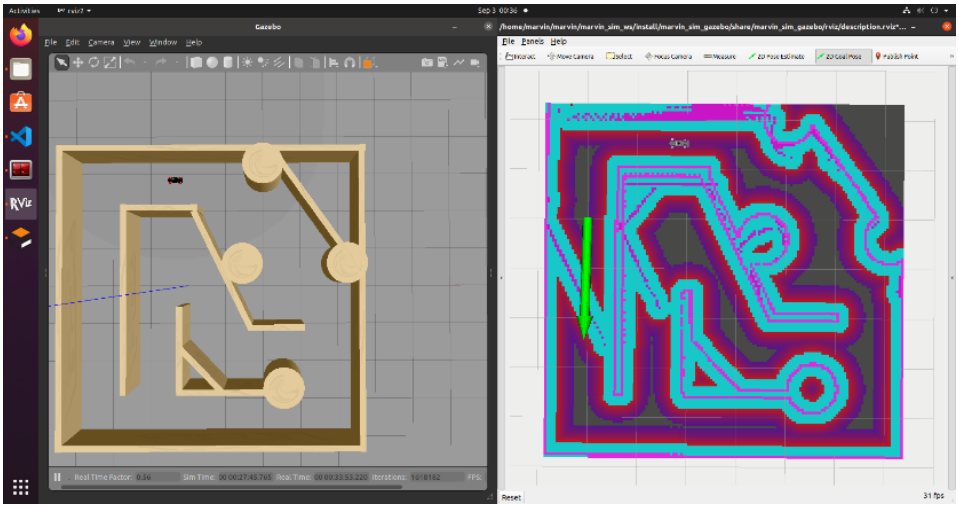
Figura 3.15

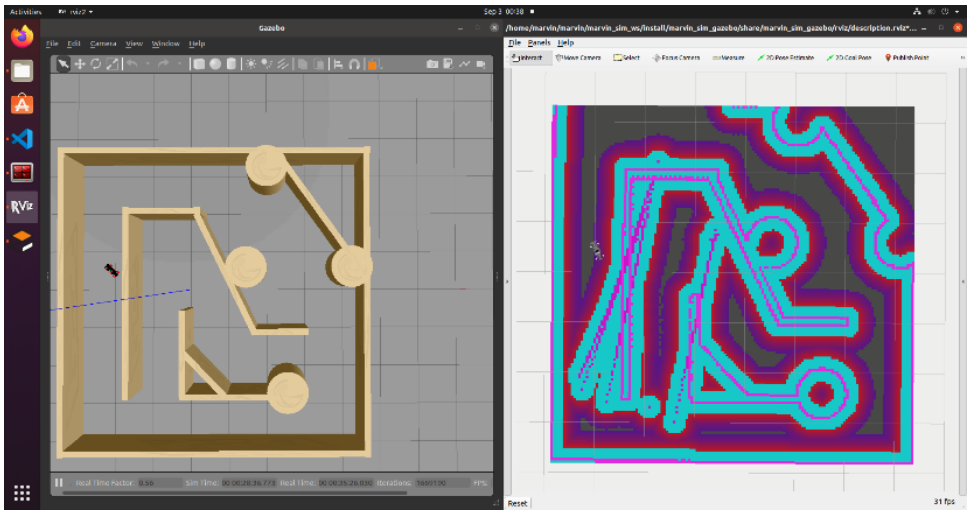
Navegación autónoma y evasión de obstáculos

a.



b.



c. 

Nota: a. El vehículo se le asigna una posición, b. El vehículo llega a su destino y se le agrega un objetivo a través de unos obstáculos, c. El vehículo evita los obstáculos y llega a su posición final.

3.6 Análisis de costos

Ya que el framework MARVIN está diseñado para ser Open Source, la única inversión que debe realizar el cliente es en los elementos tanto electrónicos como mecánicos para que este pueda funcionar efectivamente. La tabla 3.1 se incorporan los elementos utilizados en el proyecto con su respectivo costo aproximado encontrado en tiendas en línea. Hay que recalcar que, si el cliente desea reemplazar o sustituir el sensor LiDAR con uno que sea de otro tipo, tiene que adaptarlo al framework.

Tabla 3.1

Costos de inversión para el proyecto

Modelo proyecto	Costo
Chasis	$110.75
LiDAR LD19	$65.00
Motor 520 con encoder	$7.90
Intel Realsense Depth Camera D435	$309.99
Motor Servo DS3218 MG	$16.99
Yahboom ROS robot expansion control board	$76.99
Yahboom Hub USB 3.0	$25.00
Nvidia Jetson Nano Developer Kit	$145.00
Cámara IMX219-160 x2	$52.58
Batería Li-PO 11.1V- 1800mAh	$59.60
Módulo inalámbrico para Jetson Nano	$21.99
Pack de tornillos: M2- M2.5	$16.99
TOTAL	$908.78

Capítulo 4

4. Conclusiones y recomendaciones

En esta sección se describe las conclusiones del proyecto a base de los resultados que se obtuvieron en referencia a los objetivos planteados al principio del documento, haciendo énfasis en la relevancia del proyecto y su respectiva aplicación en el campo. También se agregan recomendaciones para poder mejorar u optimizar mejor el framework propuesto, implementación de sensores y posibles problemas para resolver en futuros proyectos.

4.1 Conclusiones

La navegación autónoma en vehículos para uso comercial es una industria que está creciendo gradualmente, por lo tanto, el acceso al aprendizaje de esta tecnología está en crecimiento, el framework MARVIN es una nueva contribución para este avance. Al emplear ROS2 para realizar un control tanto manual como autónomo de un sistema mecatrónica, se pudo obtener un modelo relativamente robusto y escalable.

- Se logró diseñar e implementar un modelo de piezas y soportes para los sensores del vehículo considerando que serán creados por medio de manufactura aditiva. El diseño realizado ofrece una estructura flexible que le permite al cliente poder extraer una pieza sin necesidad de tener que desarmarla toda.
- Se logró realizar una arquitectura de software que permitió una comunicación y coordinación efectiva entre los diferentes sensores y actuadores, principalmente entre la computadora principal y la placa de control de actuadores.
- Los algoritmos SLAM implementados en el vehículo a escala por medio del sensor LiDAR posibilitaron la navegación autónoma tanto en un ambiente real como simulado. Debido a esto el vehículo a escala es capaz de poder realizar un mapeo de su entono en tiempo real,

localizarse en este mapa sin importar que no se encuentre en su punto de origen y poder navegar de a través de su entorno mientras que simultáneamente compara la información del mapa con la del LiDAR, de esta forma puede evitar obstáculos nuevos.

- Se desarrolló un ambiente simulado en el cual se implementó con éxito el framework MARVIN y permitió poder utilizar más sensores simultáneamente.

En conclusión, el diseño de la estructura y framework de software implementado a un vehículo a escala basado en ROS2 ofrece un sistema mecatrónico robusto, eficiente, accesible y asequible para clientes con interés en el campo de navegación autónoma en Latinoamérica. El framework está diseñado para ser escalable, adaptable y flexible, con el objetivo de aumentar su robustez, promoviendo de esta manera el aprendizaje y la investigación.

4.2 Recomendaciones

En este proyecto mientras se pudieron realizar los aspectos suficientes para que el vehículo pueda navegar autónomamente en un entorno tanto real como simulado, hay aspectos que están abiertos para poder aumentar la robustez del sistema. Se consideran los siguientes aspectos:

- Implementar una imagen que tenga implementado la versión de ROS2 que utiliza el framework (Foxy) junto con las dependencias requeridas para el funcionamiento de la cámara de profundidad Intel RealSense.
- Emplear visualización SLAM por medio de la cámara de profundidad, para el mapeo. De esta manera el mapa generado puede ser tanto en 2D como en 3D.
- Implementar una computadora de control con más moderna, cuyo sistema operativo le permita trabajar directamente con ROS2. La Jetson Orin es una alternativa viable, ya que su firmware le permite trabajar con Ubuntu 20.04 y ROS2.

- Se recomienda trabajar con una computadora que Ubuntu como sistema operativo principal o una máquina virtual altamente optimizada, esto aumentará el procesamiento y proveerá una distribución de recursos más eficiente.
- Implementar algoritmos de reconocimiento tanto para la cámara frontal como las laterales, de esta forma por medio de la visión del robot, se puede identificar los tipos de señales para vehículos como para peatones y de esta forma realizar una navegación con más precaución hacia los conductores humanos.
- Se recomienda añadir un método de calibración para el robot, ya que una desincronización de interpretación entre el mundo real y el de la visión del robot lleva a que se interpreten de forma errónea la información que adquieren los sensores y la comunicación con los actuadores.

Referencias

[1] University of Michigan, « Autonomous Vehicles Factsheet,» *Center for Sustainable Systems*, nº CSS16-18, 2022.

[2] D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal, N. Kumar, G. Prasad Joshi y W. Cho, «A Review on Autonomous Vehicles: Progress, Methods,» *Electronics*, vol. 11, nº 14, p. 2162, Julio 2022.

[3] J. M. Anderson, N. Kalra, . K. D. Stanley, P. Sorensen, C. Samaras y T. A. Oluwatola, Autonomous Vehicle Technology: A Guide for Policymakers, Santa Monica, CA: RAND Corporation, 2016.

[4] Yahboom, «About Us,» 2023. [En línea]. Available: http://www.yahboom.net/aboutus. [Último acceso: 06 Junio 2023].

[5] Quanser, «QCar,» [En línea]. Available: https://www.quanser.com/about/. [Último acceso: 06 Junio 2023].

[6] F1TENTH, «Building the F1TENTH Car,» [En línea]. Available: https://github.com/f1tenth/f1tenth_doc .

[7] «MIT RACECAR,» [En línea]. Available: https://racecar.mit.edu.

[8] S. Siddhartha Srinivasa, . P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, . J. R. Smith, . S. Choudhury, . C. Mavrogiannis y F. Sadeghi, «MuSHR: A Low-

Cost, Open-Source Robotic Racecar for Education and Research,» *CoRR*, vol. abs/1908.08031, 2019.

[9] Lily, «howlanders,» 15 11 2021. [En línea]. Available: https://www.howlanders.com/blog/en/south-america/driving-latin-america/#:~:text=Unlike%20Europe%2C%20where%20most%20traffic,different%20to%20European%20traffic%20signs.. [Último acceso: 12 04 2023].

[10] SAE International, Taxonomy and Definitions for Terms Related to Driving Automation Systems for on-Road Motor Vehicles J3016, vol. J3016, Warrendale, PA: SAE International, 2021, pp. 30-32.

[11] S. D. Pendleton , H. Andersen , X. Du , X. Shen, M. Meghjani, Y. H. Eng , D. Rus y M. H. Ang, «Perception, Planning, Control, and Coordination for Autonomous Vehicles,» *Machines*, vol. 5, nº 1, p. 6, Febrero 2017.

[12] X. Dauptain, A. Koné, D. Grolleau, V. Cerezo, . M. Gennesseaux y M.-T. Do, «Conception of a High-Level Perception and Localization System,» *Sensors*, vol. 22, nº 24, p. 9661, Diciembre 2022.

[13] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi y R. Mangharam, «Autonomous Vehicles on the Edge:,» *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458-488, 2022.

[14] D. Dang, F. Gao y Q. Hu, «Motion Planning for Autonomous Vehicles Considering Longitudinal and Lateral Dynamics Coupling,» *Applied Sciences,* vol. 10, nº 9, p. 3180, Mayo 2020.

[15] R. Tellez, «The Construct,» 29 09 2017. [En línea]. Available: https://www.theconstructsim.com/start-self-driving-cars-using-ros/#:~:text=Robot%20Operating%20System%20(ROS)%20is,other%20actuators%20of%20the%20robot..

[16] «Donkey Car S1,» [En línea]. Available: https://github.com/autorope/donkeycar.

[17] Yahboom, «ROSMASTER R2,» [En línea]. Available: https://github.com/YahboomTechnology/ROSMASTER-R2.

[18] HOKUYO, «Hokuyo UST-10LX Scanning Laser Rangefinder,» [En línea]. Available: https://www.robotshop.com/products/hokuyo-ust-10lx-scanning-laser-rangefinder. [Último acceso: 30 06 2023].

[19] Trampa, «VESC 6 MKVI Speed Controller,» [En línea]. Available: https://trampaboards.com/vesc-6-mkv-in-cnc-t6-silicone-sealed-aluminium-box-with-genuine-xt90-connectors--vedder-electronic-speed-controller-trampa-special-p-27536.html. [Último acceso: 30 06 2023].

[20] Intel, «Intel® RealSense™ Depth Camera D435i,» [En línea]. Available: https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435i.html. [Último acceso: 30 06 2023].

[21] robotshop, «YDLIDAR X4 360° Laser Scanner,» [En línea]. Available: https://www.robotshop.com/products/ydlidar-x4-360-laser-scanner. [Último acceso: 30 06 2023].

[22] Amazon, «HGLRC Flipsky FSESC V4.12 50A ESC controlador de velocidad electrónico para monopatín eléctrico E-Bike ESK8 construir EScooter,» [En línea]. Available: https://www.amazon.com/HGLRC-FLIPSKY-SK8-ESC-Electric-Skateboard-EScooter/dp/B07GFB55NV. [Último acceso: 30 06 2023].

[23] Yahboom, «Yahboom ROS robot control board with STM32F103RCT6 IMU for Raspberry Pi Jetson Robotics,» [En línea]. Available: https://category.yahboom.net/products/ros-driver-board?_pos=4&_sid=c68121e97&_ss=r. [Último acceso: 06 30 2023].

[24] Amazon, «innomaker Lidar - Radar DTOF integrado de 360 grados de escaneo omnidireccional de 25000 lux, resistencia TOF Rango de tiempo de vuelo (LiDAR_LD19) Compatible con ROS/ROS2,» [En línea]. Available: https://www.amazon.com/dp/B09VKZ9YNT?ref=ppx_yo2ov_dt_b_product_details&th=1. [Último acceso: 30 06 2023].

[25] Amazon, «Intel RealSense Profundidad Cámara D435,» [En línea]. Available: https://www.amazon.com/-/es/Intel-RealSense-Profundidad-Cámara-D435/dp/B07BLS5477/ref=sr_1_1?__mk_es_US=ÅMÅŽÕÑ&crid=32NTG5OXGXBQA&keywords=d435&qid=1690172445&sprefix=d435%2Caps%2C160&sr=8-1. [Último acceso: 30 06 2023].

[26] Amazon, «Jetson Nano Developer Kit 16G eMMC integrado para aprendizaje automático de IA,» [En línea]. Available: https://www.amazon.com/-/es/Jetson-Developer-integrado-aprendizaje-automático/dp/B09Y94MGRZ/ref=sr_1_7?__mk_es_US=ÅMÅŽÕÑ&crid=3G716JV2V1ERS&keywords=jetson+nano&qid=1690172464&sprefix=jetson+nano%2Caps%2C171&sr=8-7. [Último acceso: 30 06 2023].

[27] YAHBOOM, «USB.0 HUB Expansion Board 1 to 4 support 5A current 9-24V Power for Raspberry Pi Jetson RDK-X3,» [En línea]. Available: https://category.yahboom.net/products/usb-hub?_pos=6&_sid=ca23a29a1&_ss=r. [Último acceso: 30 06 2023].

[28] ROBOTIS, «LIPO Battery 11.1V 1800mAh LB-012,» [En línea]. Available: https://www.robotis.us/lipo-battery-11-1v-1800mah-lb-012/ . [Último acceso: 30 06 2023].

[29] Amazon, «ANNIMOS Servo digital de 44.1 lbs de alto par de torsión completo de metal impermeable para modelo RC bricolaje, DS3218MG, ángulo de control de 270°,» [En línea]. Available: https://www.amazon.com/-/es/44-1-lbs-completo-impermeable-bricolaje-DS3218MG/dp/B076CNKQX4/ref=sr_1_6?__mk_es_US=ÅMÅŽÕÑ&crid=31TOB7LW0CL5V&keywords=20+kg+dsservo+digital+servo&qid=1690163502&sprefix=20+kg+dsservo+digital+servo%2Caps%2C143&sr=8-6. [Último acceso: 30 06 2023].

[30] S. Macenski, «wiki.ros,» 16 12 2021. [En línea]. Available: http://wiki.ros.org/slam_toolbox#:~:text=Overview,2D%20map%20of%20a%20space.. [Último acceso: 05 07 2023].

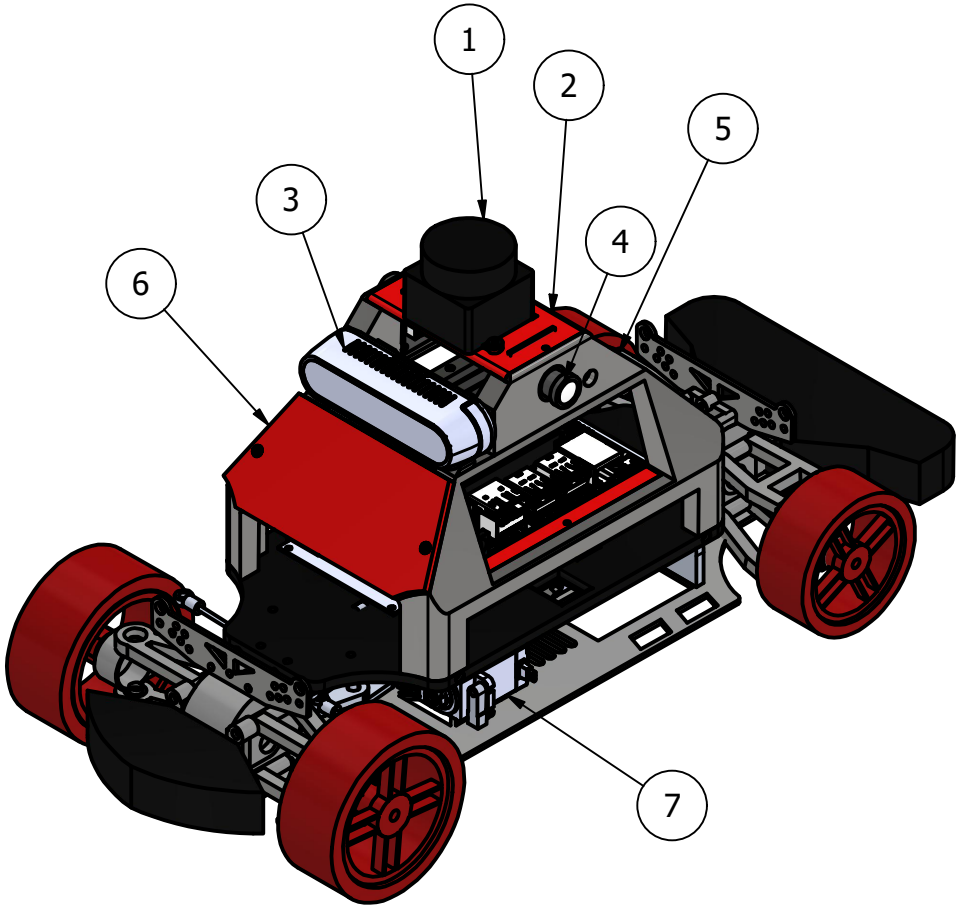
[31] P. Prakash, «Robot Localization and AMCL,» 14 04 2023. [En línea]. Available: https://www.linkedin.com/pulse/robot-localization-amcl-pratheesh-prakash/. [Último acceso: 16 07 2023].

[32] «Open Navigation LLC,» 2020. [En línea]. Available: https://navigation.ros.org/index.html. [Último acceso: 15 07 2023].

Apéndice

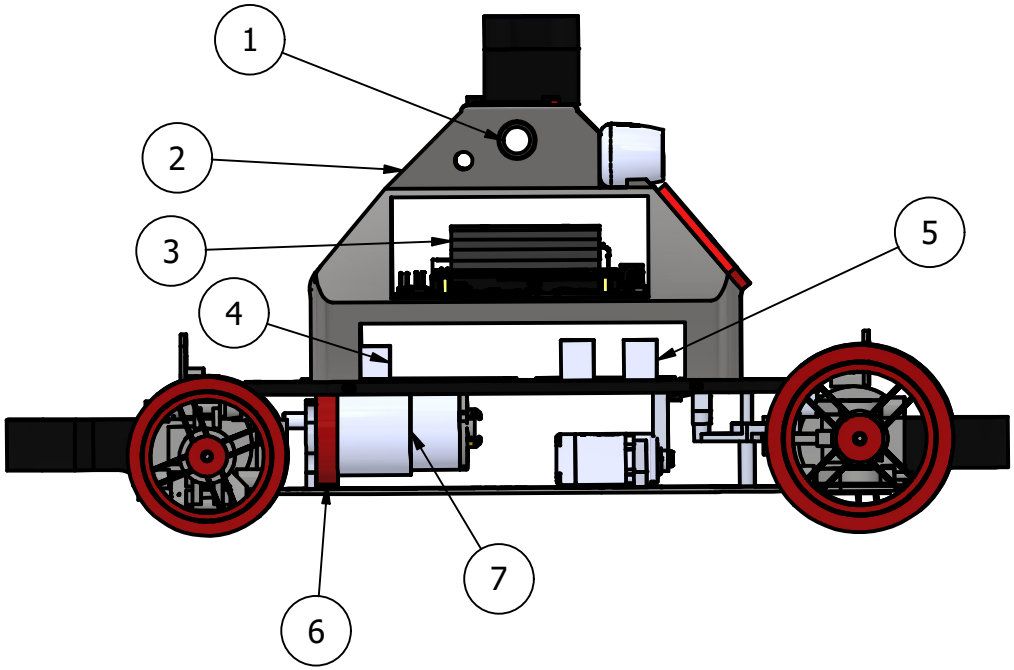
Apéndice A

Planos mecánicos



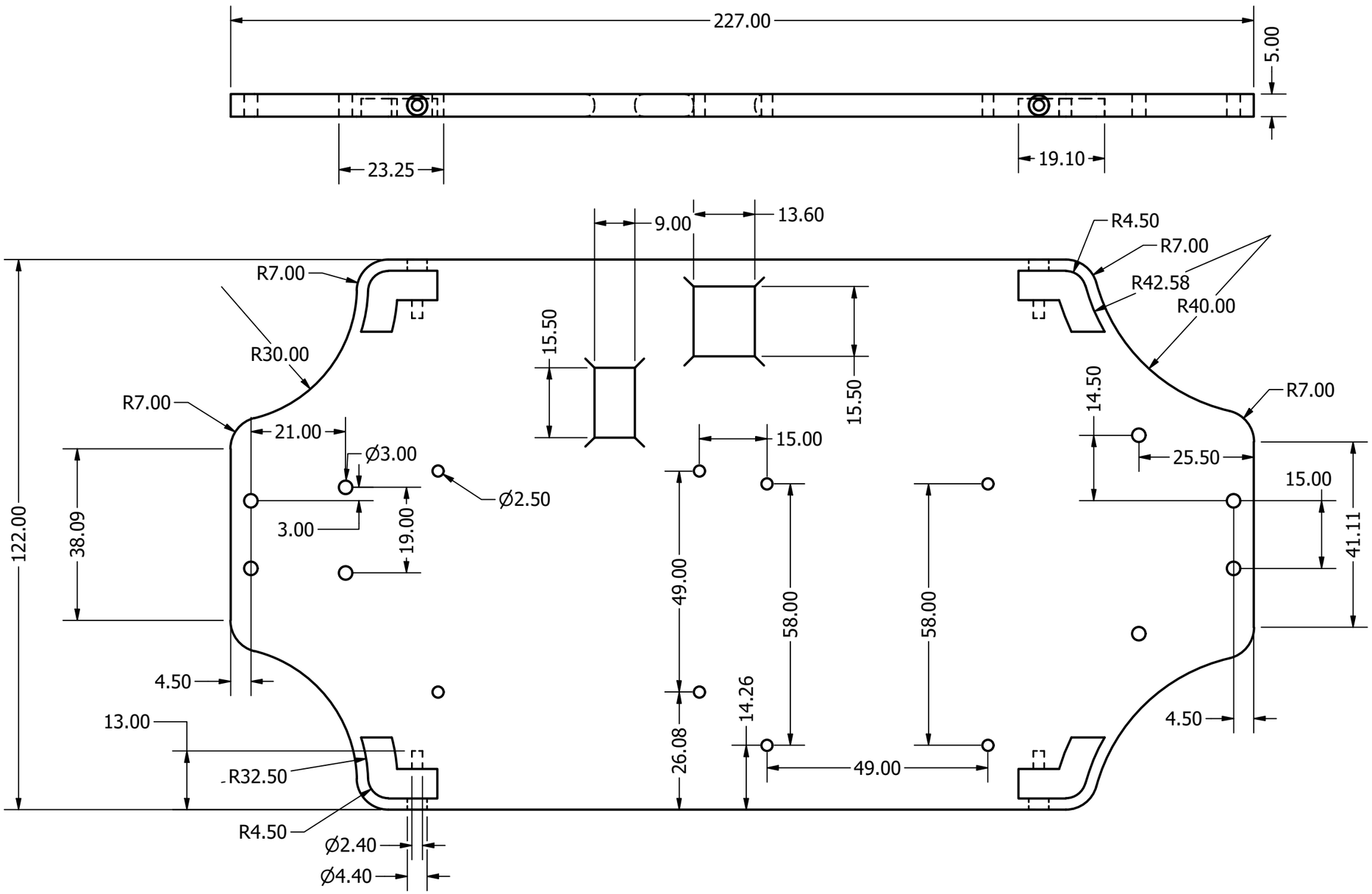
LISTA DE PIEZAS	
ELEMENTO	Nº DE PIEZA
1	Lidar
2	Soporte LIDAR
3	Intel Realsense D435
4	Cámara izquierda
5	Pieza lateral izquierda
6	Pieza con logo
7	Digital Servo 270

	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 1:3	DIBUJO Ensamble de conjunto		MARVIN
			MARVIN - 000

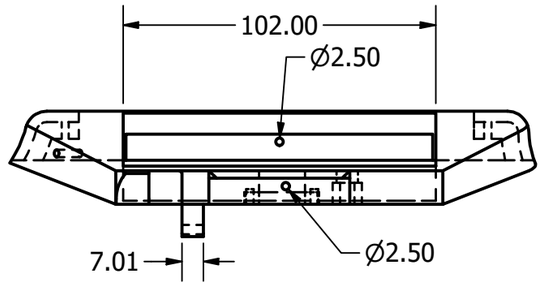


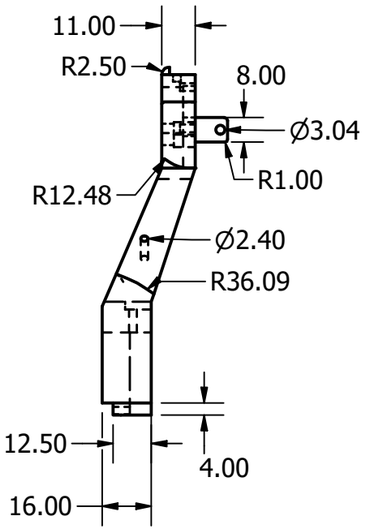
LISTA DE PIEZAS	
ELEMENTO	Nº DE PIEZA
1	Cámara derecha
2	Pieza lateral derecha
3	Jetson nano 4GB
4	Placa de control
5	Placa de expansión USB
6	Soporte para motor
7	Motor con encoder

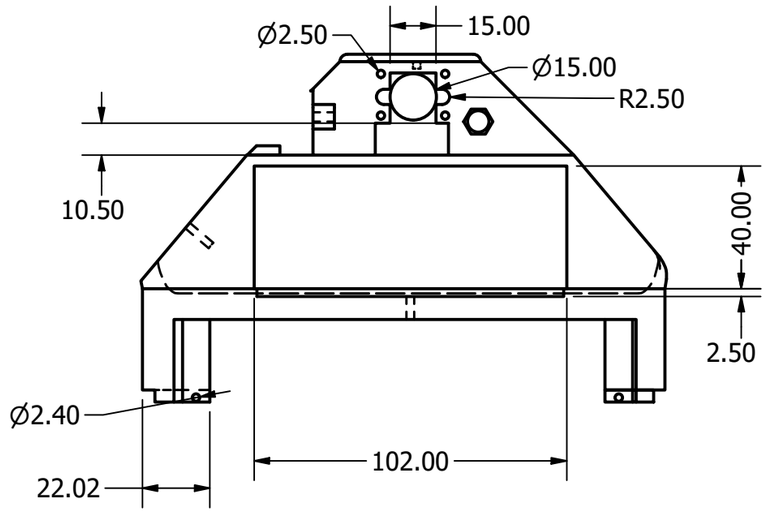
	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 1:3	DIBUJO Ensamble de conjunto		MARVIN MARVIN - 100

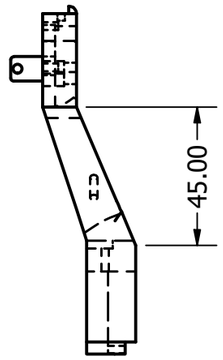


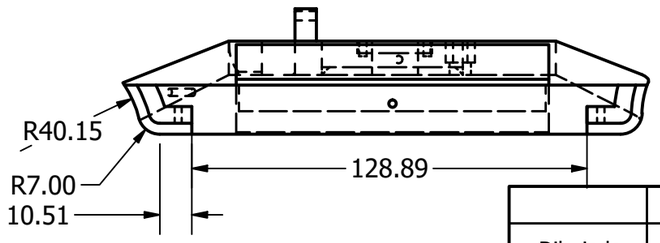
	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 1:1	DIBUJO Pieza base		MARVIN
			MARVIN - 101



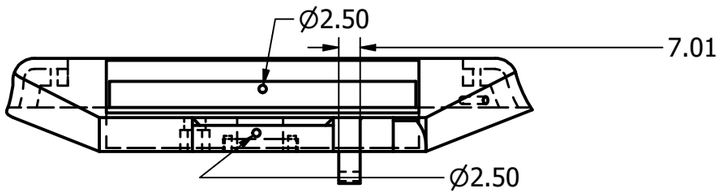


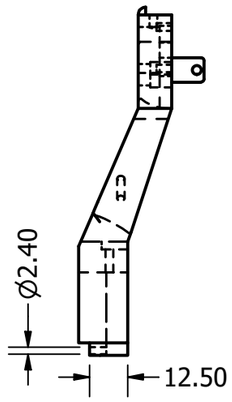


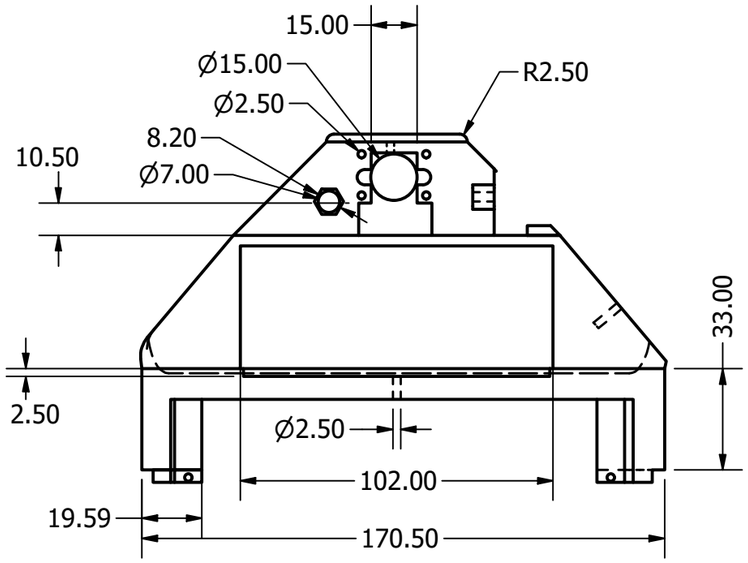


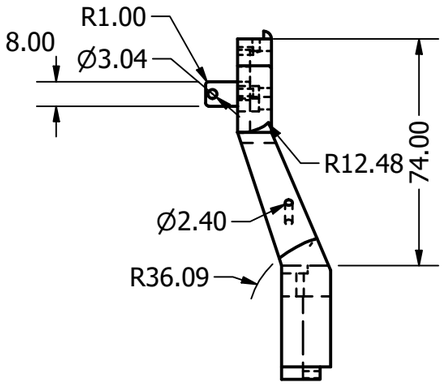


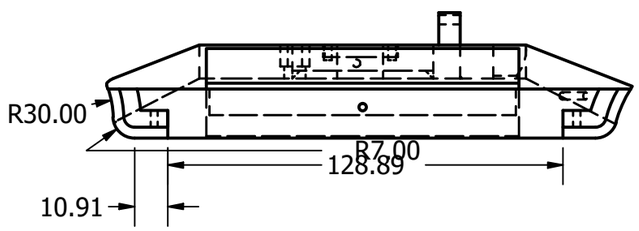
	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 1:2	DIBUJO Pieza lateral derecha		MARVIN
			MARVIN - 102



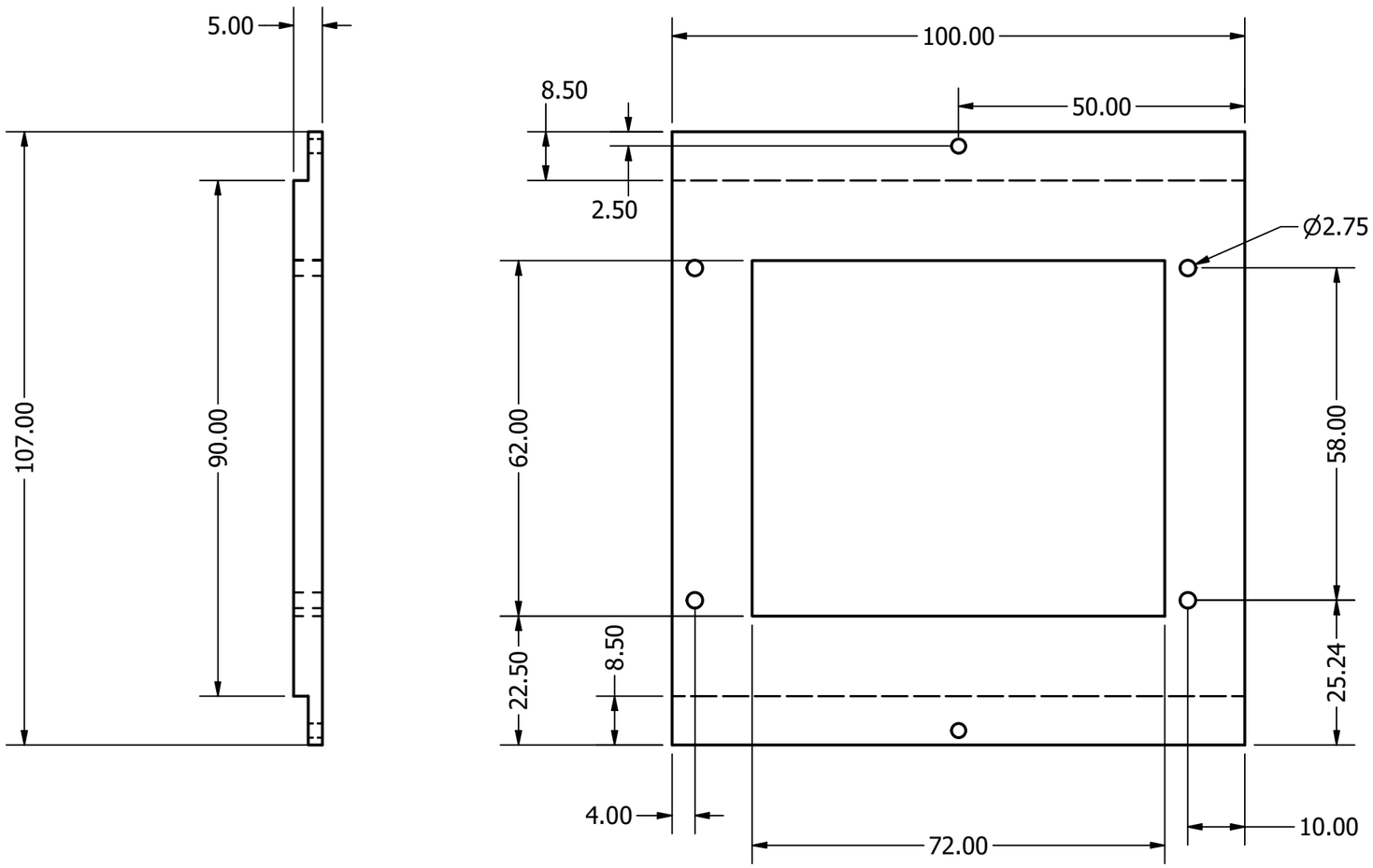








	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 1:3	DIBUJO Pieza lateral izquierda		MARVIN MARVIN - 103

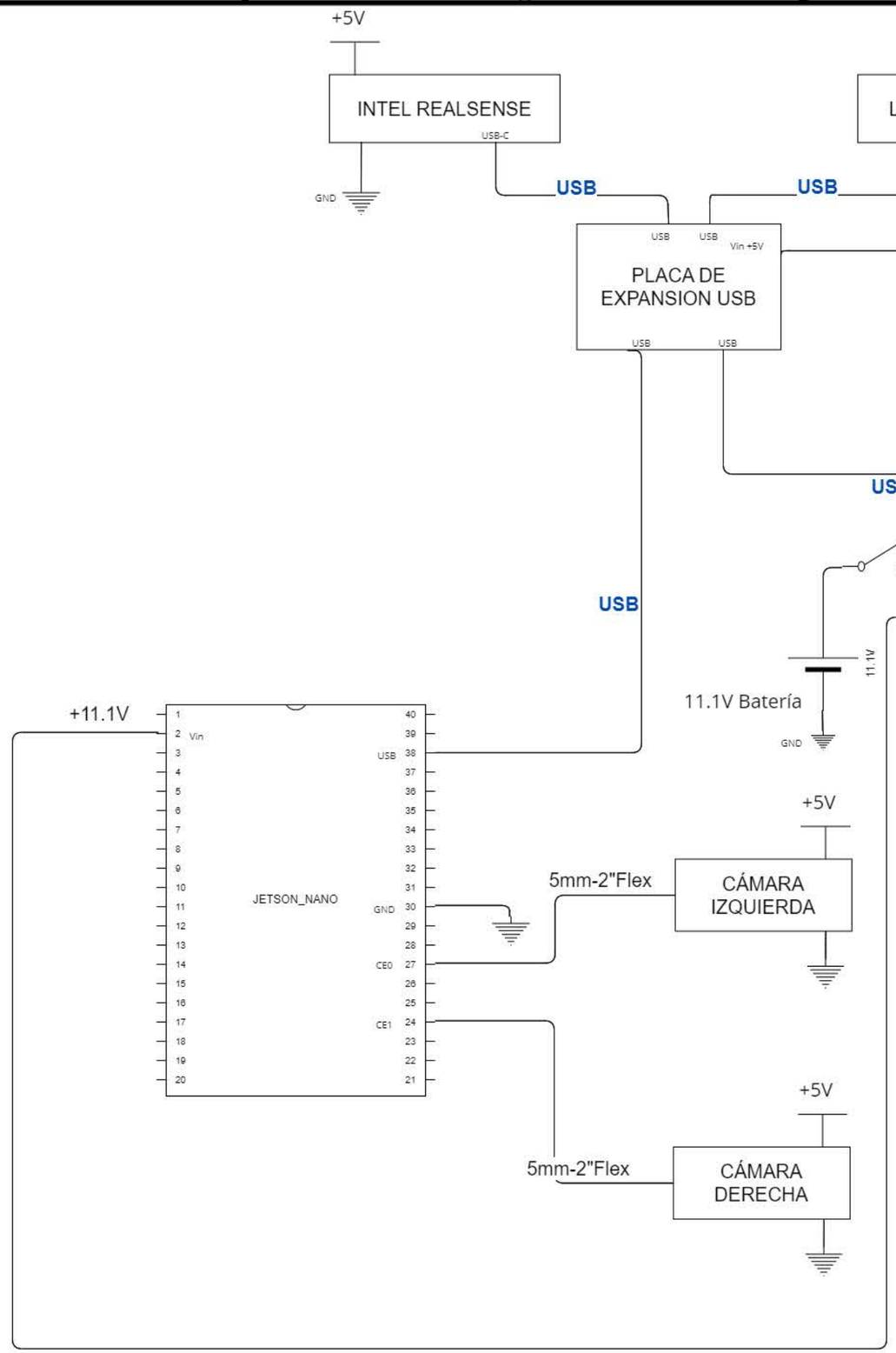


	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 1:1	DIBUJO Pieza soporte Jetson		MARVIN
			MARVIN - 104

	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENÍERIA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA 3:1	DIBUJO Pieza soporte motor		MARVIN
			MARVIN - 105

Apéndice B

Diagrama electrónico



	NOMBRE	FECHA	
Dibujado	Luis Andrade Proaño	05/09/2023	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL FACULTAD DE INGENIERÍA MECÁNICA Y CIENCIAS DE PRODUCCIÓN INGENIERÍA EN MECATRÓNICA
Cliente	Ing. Francisco Yumbla	05/09/2023	
ESCALA	DIBUJO: Diagrama electrónico		MARVIN
			MARVIN - 000

Apéndice C

Archivo Readme del repositorio

Mobile Autonomous Robot Vehicle for Investigation and Navigation

Introduction

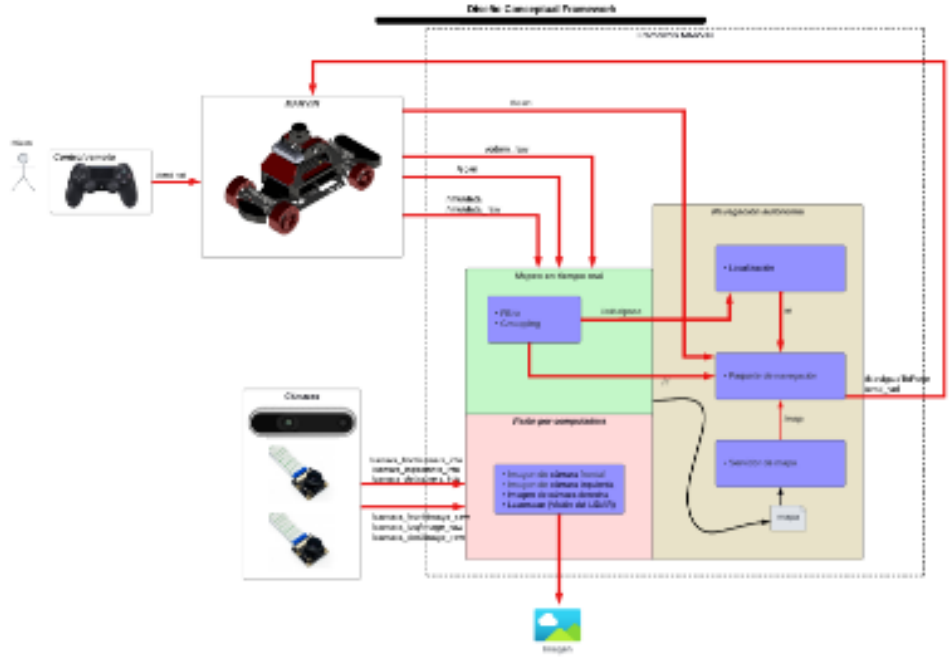
MARVIN is a comprehensive platform (built by Luis Andrade Proaño) designed to facilitate the development and deployment of autnomous vehicles with Ackermann steering , providing a set of essential modules and tools to enable various capabilities. The framework consists of the following core components:

- Manual control over the vehicle.
- Real time mapping of the enviroment sourrounding the robot.
- Localization, once the map is saved in the robots system.
- Autonomous navigation of the vehicle with object avoidance.
- Simulation enviroment designed for gazebo.
- Graphical User Interface (GUI).
- Code developed for ROS2 foxy

MARVIN is designed to be versatile, enabling researchers and developers to experiment with different robot types with the same steering system, test various algorithms, and to navigate complex enviroments in real-world scenarios. By providing these core modules and simulation environments, MARVIN is another tool for investigation in the field of robotics.

Framework Overview

An overview of the framework modules and communications is presented below:



Process

0.- Navigate to the workspace directory (The previous step to perform in all new cmd windows opened)

```
source install/setup.bash
```

1.- Launch the simulation.

```
ros2 launch marvin_sim_gazebo gazebo_spawn.launch.py world:=./src/marvin_sim_gazebo/worlds/obstacles.wo
```

2.- Open a new cmd window, source in the ws and launch SLAM.

```
ros2 launch marvin_navigation slam.launch.py use_sim_time:=true
```

In Rviz activate the map node and spec

3.- Connect a joystick to the computer and use it to move the vehicle manually around the map.

After you've mapped the area, go in Rviz to Panels -> add new Panel and click on SlamToolboxPlugin Make sure that you save the map with the name "marvin_world"

4.- After saving, finish with Ctrl-c the window where is you where running SLAM and in Rviz don't forget to Remove the Map plugin

5.- Open a new cmd window, source in the ws and launch the Navigation.

```
ros2 launch marvin_navigation navigation.launch.py use_sim_time:=true
```

To make the robot navigate autonomously use the 2D Goal Pose tool in Rviz to select a point in the map.

You can now move the robot freely through the map. You can also (during the path) use manual control to change directions, but the robot will stop at the Goal that you previously set.

Requirements:

- Operating System: Ubuntu Linux Jammy Jellyfish (22.04)
- Jetson Nano Developer Kit, Jetson Nano 4Gb
- Operating System: Ubuntu Linux Bionic Beaver 18.04
- For the simulation consider according the computational requirements .

Dependencies:

For the simulation:

- ROS2 Foxy. Installation: https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html
- Verify gazebo installation.
- Verify the acquisition of the gazebo controls library. If it is not installed:

```
sudo apt-get install ros-foxy-gazebo-ros2-control
```

Installation for the simulated vehicle

1.- Clone the github repository in a workspace source folder. (marvin_sim_ws/src)

```
git clone https://github.com/RAMEL-ESPOL/MARVIN.git
```

2.- Go to workspace directory and Install dependencies

```
cd ..
rosdep install -y --from-paths src --ignore-src
```

3.- Build the pacakge.

```
colcon build
```

Installation for the Jetson Nano

1.- Clone the github repository in a workspace source folder. (marvin ws/src)

```
git clone https://github.com/RAMEL-ESPOL/MARVIN.git
```

2.- Go to workspace directory /src and initialize the DockerFile (dont worry, the factory image of the Jetson Nano comes with Docker pre-installed).

```
cd ..
docker build . -t marvin:latest
```

After the docker has been built, you have to run it with the following conditions 3.- Build the pacakge.

```
docker run -it net=host --device=/dev/input/event3 --device=/dev/input/js0 --device=/dev/myserial --dev
```

This will let you run the Docker and create a container to work in.

Exit the docker with the "exit" command. Now that the container is created, all the files and changes that you've made, will remain there. To open the container you must first now its name. Get it with:

```
docker ps -a
```

The last name is the last container that was active. 4.- Start the container

```
docker start <container name>
```

5.- Run the container

```
docker run exec -it <container name> /bin/bash
```

##Quality of life improvements For an easier time running the Jetson nano, first enable remote control (the following video gives the proper instrucions):

```
https://www.youtube.com/watch?v=Rgmw6kPmhXI
```

After enabling the Jetson Nano for remote control, download from the main page "RealVNC Viewer"

Everytime you open a new cmd window, you must execute the running container to acces it

```
docker run exec -it <container name> /bin/bash
```

Process

0. Inside the container you'll be in the MARVIN workspace already built. (Important step to perform in all new cmd windows opened)

```
source install/setup.bash
```

1. Launch the control and LiDAR of the robot

```
ros2 launch marvincar_bringup marvincar_bringup.launch.py
```

2. Open a new cmd window, source in the ws and launch the LiDAR.

```
ros2 launch marvin_lidar ld19.launch.py
```

In Rviz activate the map node and spec

3. Open a new cmd window, source in the ws and launch Rviz

```
ros2 launch marvincar_nav view_map_launch.py
```

4. Start the mapping node

```
ros2 launch marvincar_nav mapping_launch.py
```

Move the vehicle so that SLAM Toolbox maps the area in real time.

After you've mapped the area. 5. After you've mapped the area, go in Rviz to Panels -> add new Panel and click on SlamToolboxPlugin Make sure that you save the map with the name "marvincar"

6. Close the Map Plugin and the cmd Windows that's running the mapping

7. Launch the Navigation.

```
ros2 launch marvincar_nav navigation_dwa_launch.py
```

To make the robot navigate autonomously use the 2D Pose Estimate to select the point in the map where the real robot is, drag the click to specify its orientation. Use the 2D Goal Pose tool in Rviz to select a point in the map.

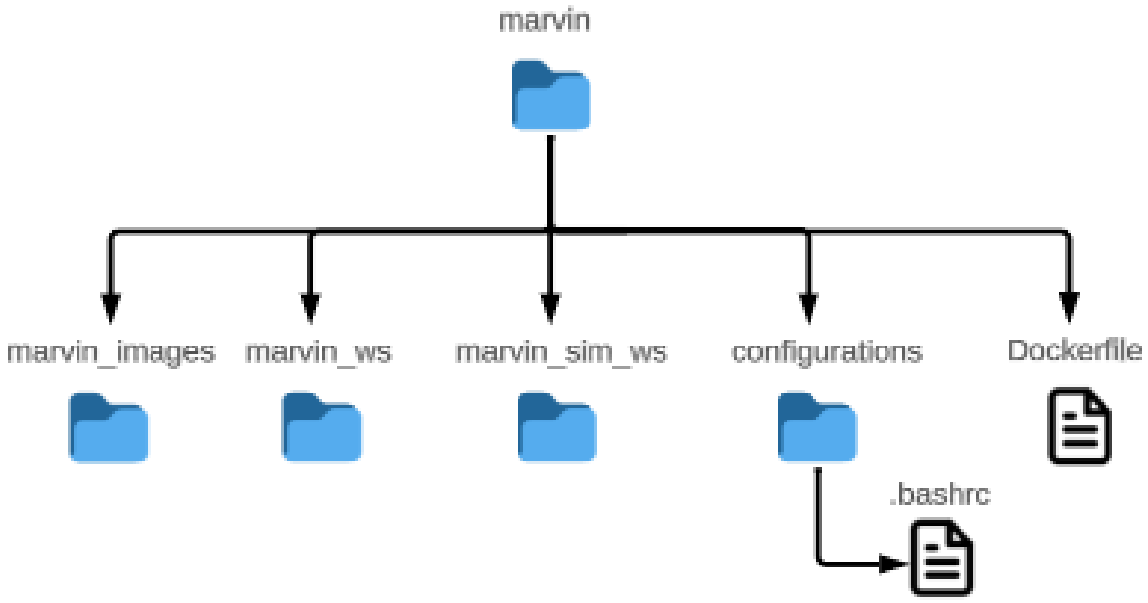
You can now move the robot freely through the map.

=======

Package Structure

The package structure is the following:

- **marvin_images** -> includes images of the project.
- **marvin_ws** -> includes the robots' packages (In this repo are all the packages that control the vehicule including the LiDAR LD19 sensor).
- **marvin_sim_ws** -> includes the robots' simulation packages (In this repo are all the packages that control the vehicule including the different Gazebo enviroments ready for usage).
- **configurations** -> contains the bash file that replaces the original created in the Docker container.
- **Dockerfile** -> contains a list of instrucctions to build a container from a file. (Includes the main container image with ubuntu 20.04 and ROS2 foxy installed).



Apéndice D
Repositorio en GitHub