Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Desarrollo y diseño de un sistema de gestión y control de entrega de paquetes en entornos interiores para un robot móvil de servicio

INGE-2320

Proyecto Integrador

Previo la obtención del Título de:

Ingeniero en Mecatrónica

Presentado por:

Emmanuel Alejandro Párraga Guerrero

Cristopher Paul Valladarez González

Guayaquil - Ecuador

Año: 2023

Dedicatoria

El presente proyecto lo dedico a mi abuela, que siempre creyó en mí y me apoyó en todo lo que me propuse; a mi mamá, que supo guiarme y acompañarme en todo momento de mi vida, que me apoyó y siempre fue y será el pilar fundamental de mi vida; a mis hermanos, que me dieron las fuerzas para seguir adelante y me enseñaron lo importante y hermoso que es ser su hermano mayor; y finalmente, también se lo dedico a todas esas personas que me acompañaron en mi camino y contribuyeron para formar al ingeniero que soy hoy. ¡Gracias!

Emmanuel Párraga.

El presente proyecto va dedicado primeramente a Dios, quien me ha guiado durante todo este camino.

A mi madre, Reina Valladarez, que supo darme su amor y apoyo incondicional en todo momento.

A la memoria de mi abuelo, Miguel Valladarez, quien estaría orgulloso de todo lo que he logrado.

Finalmente, se lo dedico a todas las personas que me han acompañado en mi formación profesional.

Cristopher Valladarez.

Agradecimientos

Agradezco a Dios por ser mi guía en la trayectoria de mi carrera universitaria y darme salud y sabiduría para culminar con éxitos mis estudios profesionales de Ingeniería. A mi familia, gracias por apoyarme todos estos años y ayudarme a cumplir mi meta. Al M.Sc. Ing. Holger Cevallos, agradezco por brindarme su confianza para ser su ayudante académico y apoyarme durante todos estos años. A mis amigos, quienes alegraban mis días con su compañía. A mi amigo y compañero de tesis Cristopher Valladarez, agradezco por nunca rendirse y confiar en mí para lograr nuestros objetivos. A CIDIS, gracias por abrirnos sus puertas y confiar en nosotros. Finalmente, le agradezco al Ph.D. Ing. Dennys Paillacho y al Ph.D. Ing. Carlos Saldarriaga por darnos la oportunidad de trabajar con WALTER, guiarnos en nuestro proyecto y por ser unos excelentes tutores.

Emmanuel Párraga.

Agradezco primeramente a Dios, por brindarme fuerza, salud y sabiduría para culminar mis estudios universitarios.

A mi madre y toda mi familia, por el inmenso apoyo físico y emocional a lo largo de mi carrera. A mis amigos, con quienes compartí muchos momentos de risas y felicidad. A mi amigo y compañero de tesis Emmanuel Párraga, por demostrar inmensa dedicación y compromiso a pesar de los tiempos difíciles.

A mis profesores, por brindarme sus conocimientos que han aportado a mi formación profesional

Agradezco al PhD Dennys Paillacho junto al PhD Carlos Saldarriaga por permitirnos compartir junto a ellos el hecho de trabajar con WALTER.

Finalmente, un agradecimiento a mi gatito Pancho, por siempre acompañarme en aquellas noches largas de estudio.

Cristopher Valladarez.

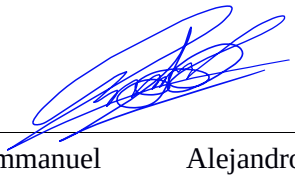
Declaración Expresa

Nosotros Emmanuel Alejandro Párraga Guerrero y Cristopher Paul Valladarez González acordamos y reconocemos que:

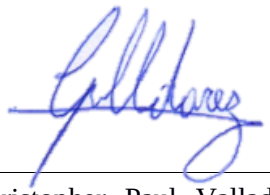
La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores. La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique a los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, enero del 2024.



Emmanuel Alejandro Párraga Guerrero



Cristopher Paul Valladarez González

Evaluadores

Firmado electrónicamente por:
CARLOS XAVIER
SALDARRIAGA MERO

Carlos Xavier Saldarriaga Mero. PhD.

Profesor de Materia

Firmado electrónicamente por:
DENNYS FABIAN
PAILLACHO CHILUIZA

Dennys Fabian Paillacho Chiluiza. PhD.

Tutor de proyecto

Resumen

El sector terciario representa el 62,3% del Producto Interno Bruto (PIB) de Ecuador en 2023 según un estudio del Banco Central del Ecuador, nuestro proyecto propone diseñar, controlar y programar un robot móvil autónomo de servicio que cumpla funciones de servicio de entrega de paquetes en ambientes cerrados, este robot utilizado en cualquier ambiente que involucre tareas de servicio como restaurantes, hoteles y eventos.

El robot realizó todas las tareas de reconocimiento de espacios y navegación utilizando técnicas de SLAM como AMCL o Cartographer, también utilizó las cámaras para detectar personas y el Lidar para mapear espacios cerrados y ubicar al robot en el espacio.

Se realizó un análisis mecánico en el cual se estableció las cargas respectivas de un funcionamiento normal, se obtuvo que el robot tiene un factor de seguridad igual a 15 y un torque máximo requerido en los motores igual a 1,36 Nm, se diseñó y programó una estructura de nodos en ROS con la que el robot realizó todo su funcionamiento, así como también se simuló el robot utilizando Gazebo y Rviz, también se diseñó un control PI de los motores utilizando un enfoque dinámico y utilizando el firmware del controlador DHB-10, finalmente se realizó un análisis de costos el cual determinó que el precio inicial del robot es de $2600 y se podría comercializar fácilmente entre un rango de $3000 - $4000.

Se concluye finalmente que nuestra estructura de nodos en ROS junto al robot es capaz de desempeñar tareas de servicio correctamente con tiempos de trabajo correctos de 4 horas, esquivando obstáculos dinámicos con tiempos de reacción de pocos segundos y entregando paquetes de una estación base a otra así cumpliendo su propósito.

Palabras clave: ROS, robot, diseño, navegación, servicio, odometría

Abstract

The tertiary sector represents 62.3% of Ecuador's Gross Domestic Product (GDP) in 2023, according to a study by the Central Bank of Ecuador. Our project proposes to design, control, and program an autonomous mobile service robot that performs package delivery tasks in enclosed environments, this robot can be used in any location that involves service tasks such as restaurants, hotels, and events.

The robot executed all space recognition and navigation tasks using SLAM techniques such as AMCL or Cartographer. It also utilized cameras for person detection and Lidar for mapping enclosed spaces and locating the robot within the environment.

A mechanical analysis was conducted to establish the respective loads for normal operation. The results indicated that the robot has a safety factor of 15 and a maximum required torque in the motors equal to 1.36 Nm. A ROS node structure was designed and programmed for the robot to perform its functions, and the robot was simulated using Gazebo and Rviz. A PI control of the motors was designed using a dynamic approach and the DHB-10 controller firmware. Finally, a cost analysis determined that the initial price of the robot is $2600, and it could be marketed in the range of $3000 - $4000.

In conclusion, our ROS node structure, along with the robot, can successfully perform service tasks with correct working times of 4 hours, avoiding dynamic obstacles with reaction times of a few seconds, and delivering packages from one base station to another, thus fulfilling its purpose.

Keywords: ROS, robot, design, navigation, service, odometry.

Índice general

Abreviaturas

ESPOL	Escuela Superior Politécnica del Litoral
ROS	Robot Operating System
RVIZ	ROS visualization
SLAM	Simultaneous Localization and Mapping
AMCL	Adaptive Monte-Carlo Localizer
PWM	Pulso De Ancho Modulado
YOLO	You Only Look Once
URDF	Unified Robotic Description Format
PLA	Ácido Poliláctico
PID	Proporcional, integral y derivativo
PI	Proporcional e integral
DC	Corriente directa
LIDAR	Light Detection and Ranging
CIDIS	Centro de Investigación, Desarrollo e Innovación de Sistemas Computacionales

Simbología

m Metro

m^2 Metro cuadrado

mm Milímetro

s Segundo

W Watt

Wh Watt/hora

Kg Kilogramo

lb Libra

N Newton

m^3 Metro cúbico

kPa Kilopascal

Pa Pascal

s^2 Segundo cuadrado

min Minuto

$ Dólar

Índice de figuras

Índice de tablas

Índice de planos

Capítulo 1

1 Introducción

Durante décadas, la industria de servicios en Ecuador, famosa por su cultura y hospitalidad, ha mantenido tradicionalmente sus métodos y costumbres. No obstante, en un entorno tecnológico en constante evolución, la falta de adaptación puede resultar en un estancamiento en la calidad y eficiencia de los servicios brindados. Un problema evidente en este contexto es la falta de adopción de tecnologías de vanguardia, como robots de servicio, por parte de empresas relacionadas con el sector hotelero, la gestión de eventos sociales y restaurantes.

En este contexto, la falta de robots de servicio es un problema que limita el potencial de crecimiento y desarrollo del sector. Los robots de servicio pueden realizar tareas fácilmente gestionables o repetitivas de forma eficiente, lo que puede liberar a los empleados para que se centren en tareas más complejas y de mayor valor creativo.

Un estudio realizado por el diario “EL MERCURIO” se aborda temas importantes sobre el uso de los robots de servicio, este estudio habla sobre diferentes ejemplos que se podrían incorporar en Ecuador como en el hospital Circolo di Varese (Italia) donde seis pequeños androides Sanbot recorren las instalaciones ayudando en la asistencia de enfermos de covid-19, en algunos supermercados de Alemania, el androide Pepper dice a los clientes que deben guardar una distancia de seguridad y usar mascarilla, estos son ejemplos de cómo los robots de servicio pueden influir en la vida diaria de las personas [1].

La implementación de este concepto robótico de servicio no solo impulsaría el progreso tecnológico en el sector de servicios ecuatoriano, sino que también mejoraría la experiencia de los usuarios, ofreciéndoles un servicio más eficiente y personalizado.

A lo largo de esta tesis, se explorarán los desafíos y oportunidades de la implementación de robots autónomos de servicio en el contexto ecuatoriano. Se analizarán las ventajas

tecnológicas y económicas de esta solución, así como los posibles obstáculos culturales y regulatorios que deben superarse. Además, se diseñarán y evaluarán algoritmos de programación, el uso de tecnologías avanzadas, como sensores LiDAR, cámaras de profundidad, algoritmos SLAM e incluyendo control PID para control de velocidad de motores DC.

1.1 Descripción del problema

En Ecuador, el sector de servicios, que abarca desde la hospitalidad en hoteles hasta la gestión de eventos sociales y la atención en restaurantes, se ha mantenido inmutable en gran medida a lo largo de las décadas. A pesar de la rica tradición de calidez y hospitalidad del país, la falta de actualización en los métodos de servicio se ha convertido en un desafío palpable. Esta resistencia al cambio ha llevado a una marcada desactualización, y en un mundo en constante evolución tecnológica, la necesidad de modernizar y optimizar los servicios se hace evidente.

La falta de soluciones tecnológicas que puedan realizar tareas repetitivas y rutinarias en el sector de servicios es un aspecto específico de esta preocupación. La incorporación de tecnologías avanzadas, como robots de servicio, se presenta como una solución prometedora para empresas relacionadas con hoteles, eventos sociales y restaurantes, donde ciertas actividades pueden ser predecibles y repetitivas.

En conjunto, estos elementos resumen la problemática que enfrenta el sector de servicios en Ecuador. La falta de actualización en los métodos de servicio, la carencia de soluciones económicas y prácticas para tareas repetitivas y la necesidad de implementar rutas de entrega en espacios interiores representan desafíos significativos que deben abordarse para impulsar la modernización y la eficiencia en este sector crucial de la economía ecuatoriana.

1.2 Justificación del problema

Según el Banco Central del Ecuador, el sector terciario o también conocido como sector de servicio representó el 62.3% del Producto Interno Bruto (PIB) de Ecuador en 2023. Esto significa que el sector terciario es el sector más importante de la economía ecuatoriana, representando más de la mitad de la actividad económica, y es responsable de generar alrededor de dos tercios del empleo. Sin embargo, este sector enfrenta una serie de desafíos, entre los que se encuentra la falta de innovación tecnológica [2].

Se puede observar y destacar que el uso de diferentes tipos de robots de servicio es importante debido a que tiene un sector amplio en el cual influir, la carencia o la poca implementación de robots de servicio en el Ecuador es evidente debido a que no tienen el personal requerido para dar mantenimiento a estos robots o los precios son muy elevados para una pequeña o mediana empresa.

En primer lugar, es fundamental destacar que Ecuador, como país en desarrollo, depende en gran medida de la industria de servicios para su crecimiento económico. La falta de actualización y modernización de los métodos de servicio se traduce en una pérdida de eficiencia, competitividad e innovación en general del mercado.

Los robots de servicio, desde la perspectiva de un ingeniero mecatrónico, son una solución versátil y valiosa para automatizar tareas monótonas y repetitivas, liberando a los trabajadores humanos para tareas más creativas y estratégicas. La falta de su implementación en el entorno de servicios de Ecuador representa una oportunidad perdida en términos de eficiencia operativa y mejora en la calidad de los servicios.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar, controlar y programar un robot autónomo de servicio con sistema de detección de obstáculos y odometría preconfigurada para la gestión y entrega de paquetes en entornos cerrados.

1.3.2 Objetivos específicos

1. Realizar un mapeo detallado y preciso del entorno en tiempo real utilizando un sensor LiDAR y la técnica SLAM.
2. Generar de trayectorias de movimiento basado en coordenadas iniciales y finales empleando odometría del robot para tareas de servicio.
3. Diseñar la arquitectura de Nodos en ROS para comunicación con Aplicación Móvil para el control de posicionamiento y movimiento.

1.4 Estado del arte

1.4.1 Opciones en el Mercado

Actualmente en Ecuador no existen robots autónomos dedicados a la entrega de paquetería o comida en espacios cerrados, pero esta tecnología ya es existente en países desarrollados, sobre todo en hoteles y restaurantes de renombre que buscan brindar una mayor experiencia al usuario. Dentro de los servicios robóticos y soluciones existentes se encuentra Servi Robot, de la empresa Bear Robotics [3].

Este es un robot de servicio multiuso con diferentes niveles, su controlador se basa en la lectura de sensores LiDar de último modelo además de múltiples cámaras, Servi tiene la capacidad de navegar por el espacio inteligentemente, cuenta de hecho, con un sistema de sincronización que permite enlazar tareas entre varios robots, permitiendo un trabajo colaborativo y personalizable.

Este robot tiene un peso de 34 Kg y una carga útil de hasta 30 Kg, su controlador corresponde a una tableta externa o una pantalla táctil en el robot, en donde el usuario puede indicar a Servi a que posición dirigirse y darle futuras órdenes [4].

Figura 1.1

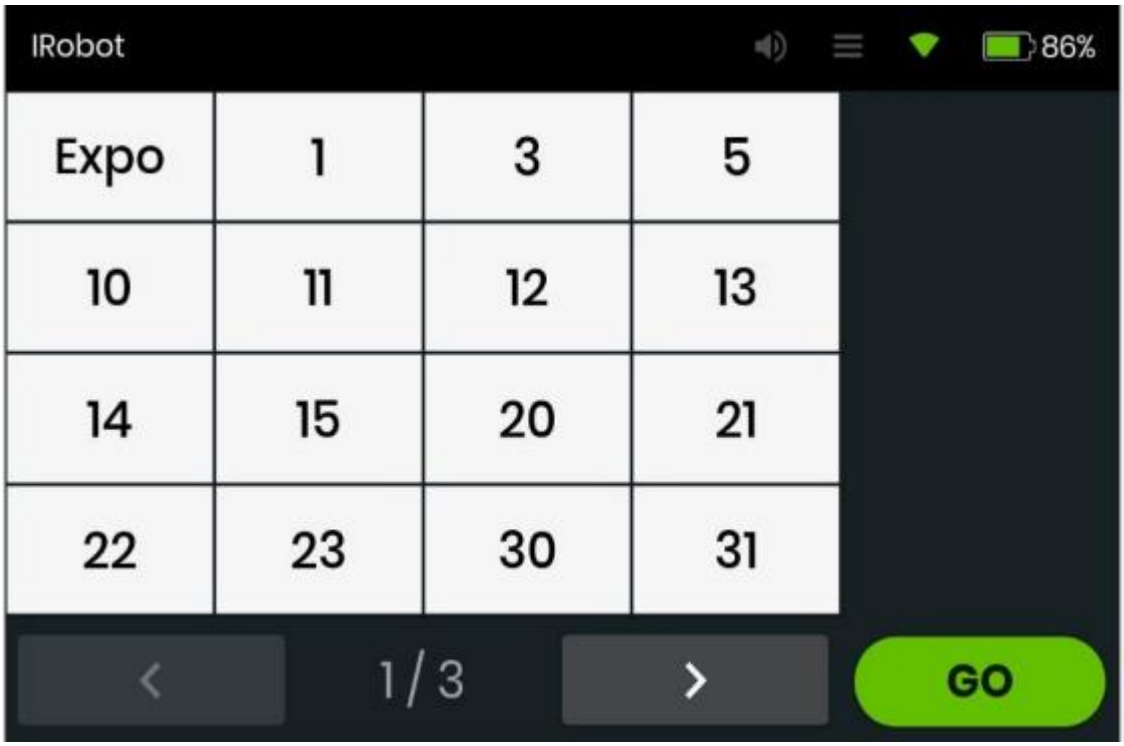
Robot Servi de la empresa BearRobotics

Nota. Gráfica obtenida de página de especificaciones de productos robóticos ofrecidos por BearRobotics [4]

En lo que corresponde a su controlador, el robot cuenta con ubicaciones preconfiguradas que permiten al usuario realizar un movimiento directo desde la posición actual del robot, hasta la posición deseada, ya sea con una configuración numérica o con localización Qr [5].

Figura 1.2

Selección de Posición para movimiento autónomo de Servicio



Nota. Gráfica obtenida del manual de Usuario de Servi [5]

La navegación del robot se basa en un mapeo proporcionado por sensor LiDar y enviando esta información leída por protocolo Wi-fi, en este mismo mapa, es donde el robot conoce sus puntos de ubicación, al posicionar el robot Servi, hay que asegurarse de la correcta posición en el punto deseado, además también de apuntar a la misma dirección preestablecida.

Figura 1.3

Mapa de ejemplo de posiciones preestablecidas para Servi

Nota. Gráfica obtenida del manual de Usuario de Servi [5]

Como Segunda solución existente en el mercado, se tiene el robot Relay2 de la compañía Relay Robotics [6], este robot autónomo está enfocado en el trabajo de entrega de paquetería en hoteles, puede moverse y navegar entre multitud de personas e inclusive tiene la capacidad de utilizar elevadores.

Dentro del mercado de este robot, se encuentra operando en Hoteles de renombre como el Hotel Hilton y el Hotel Marriot Bonvoy, ambos ubicados en Los Ángeles, California, donde realiza tareas de entregas de productos del tipo servicio a la habitación.

Relay2 es un Robot con alta capacidad de espacio, con hasta 41 litros de almacenamiento interno, utiliza cámaras y LiDAR para identificación de obstáculos tiene una

altura de 92 centímetros y un peso de 40.8 kg, el robot además logra velocidad de 2.5 km/h [7]. La información sobre su controlador o guía de usuario no es pública.

Figura 1.4

Robot Relay2 de la empresa Relay Robotics

Nota. Gráfica obtenida de Blog oficial de Relay Robotics [8]

1.5 Marco teórico

1.5.1 Robot Autónomo de Servicio

Un Robot Autónomo es un sistema mecatrónico diseñado para llevar a cabo tareas específicas de manera independiente o parcialmente independiente, es decir, no requiere una supervisión continua por parte de un operador humano [9], a diferencia de un Robot Autónomo, un Robot Autónomo de Servicio o Service Robot en inglés, está diseñado para ser usado fuera de un entorno industrial, los robots que se utilizan en trabajos de primera línea con el objetivo de tener interacción con el cliente, se los denomina Professional Social Services Robots [10], por lo general brindan servicios de hospitalidad, asistencia en atención a pacientes en hospitales, y servicios de delivery.

Los fundamentos tecnológicos de un Robot Autónomo de servicio son [10]:

- Arquitectura y Plataforma: Se refiere al diseño general de robot y su propósito, incluye la selección de componentes de software, hardware, sensores y cualquier componente tangible del robot.
- Interacción con el Entorno: El robot debe tener la capacidad de interactuar con su entorno, ya sea con humanos, obstáculos u objetos, el objetivo principal de cualquier robot autónomo de servicio es brindar ayuda en tareas específicas a los humanos.
- Navegación: La arquitectura del robot debe tener la capacidad de realizar tareas de navegación, en las cuales se incluyen detectar obstáculos, mapear y auto percibir su entorno, garantizando así un movimiento autónomo y fluido.

Figura 1.5

Robots de Servicio de diferentes marcas para Servicio al Cliente

Nota. Gráfico Obtenido de Kedglobal [11]

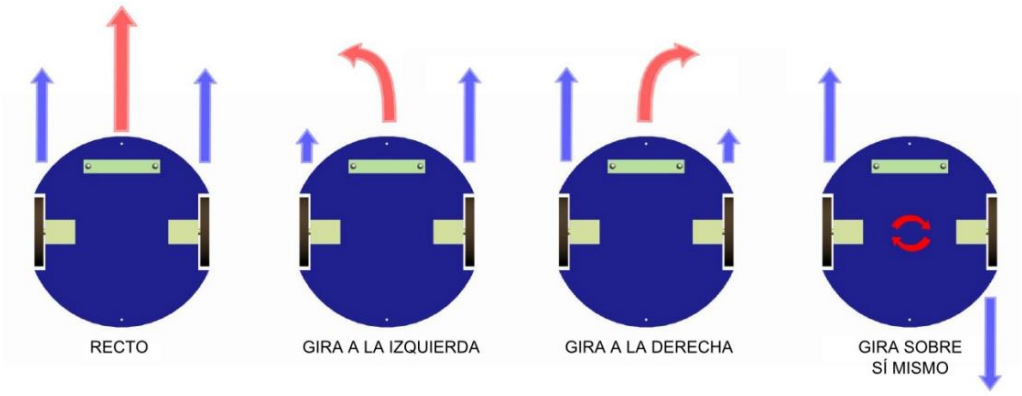
1.5.2 Robot Diferencial

Un robot diferencial o también conocido como robot de tracción diferencial es aquel robot cuyas ruedas son controladas y accionadas por motores independientes, esta locomoción se caracteriza por el uso de dos sistemas de tracción independientes, que, por lo general, están ubicadas en el mismo eje, para añadir estabilidad al robot, se colocan ruedas pasivas para mejorar el cambio de movimiento, estas ruedas se conocen como “rueda loca” o castor Wheel en inglés [12].

Al tener en consideración que las ruedas pueden girar a diferentes velocidades y direcciones sin depender la una de la otra, se puede controlar tanto la velocidad lineal como la velocidad angular [12], permitiendo realizar distintos tipos de movimiento, hacia posiciones diferentes o giros sobre su propio eje a partir de su configuración de velocidades y dirección:

Figura 1.6

Trayectoria de un robot diferencial en función de la velocidad de sus ruedas



Nota. Gráfica de Configuración diferencial obtenida de [13]

1.5.3 ROS

ROS [14] o Robotic Operation System, es un conjunto de librerías, herramientas y frameworks de software libre, que permiten construir aplicaciones robóticas, su objetivo principal es brindar una comunicación sencilla y simplificada entre los actuadores, sensores y controladores a través de la creación de Nodes, Topics y Messages, en ROS1.

1.5.3.1 Estructura de Comunicación en ROS

La estructura de comunicación de ROS se basa en 4 aspectos:

a) Activación de roscore: Este comando base activa todas las funcionalidades de los parámetros y servidores de ROS.

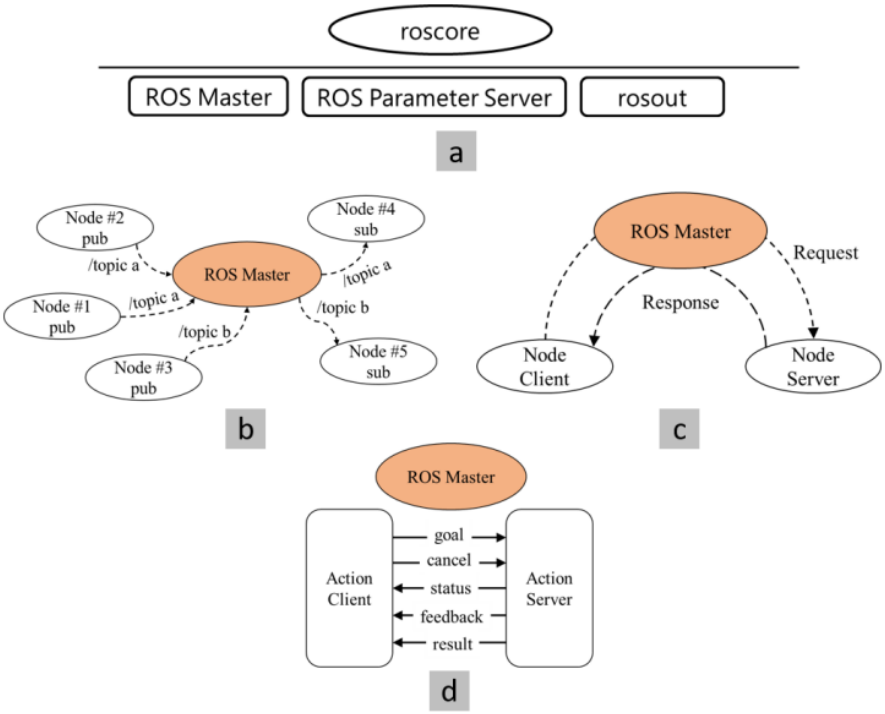
b) Publishers y Susbcribers: El robot continuamente publica información mediante Topics y se subscribe a nodos que brindan información sobre valores obtenidos por sensores o estados del robot.

c) Servidor y Cliente: En ocasiones específicas los nodos requieren obtener información o permisos de un servidor para realizar acciones, el servidor recepta esta petición y decide si aceptarla o negarla.

d) Comunicación de acciones: Las acciones son utilizadas para realizar tareas largas o que requieren alto nivel computacional y retroalimentación continua.

Figura 1.7

Diferentes estructuras de manejo de datos en ROS



Nota. Diagramas de comunicación utilizados en ROS, diagrama obtenido de: [15]

1.5.4 Navegación SLAM (Simultaneous Localization and Mapping)

SLAM es una técnica ampliamente utilizada que permite a un sistema robótico o algoritmos de inteligencia artificial realizar exploraciones y mapeos de entornos a partir de cámaras o sensores que envían información hacia un controlador. SLAM brinda la capacidad de explorar un entorno sin tener conocimiento o entrenamiento previo acerca de las características de la ubicación, la premisa de la navegación SLAM se basa en la capacidad de ubicar a un robot móvil en un ambiente desconocido y forzarlo a construir un mapa y aprender

de su entorno [16]. Por lo general, para implementar un algoritmo SLAM se necesita de sensores que tengan la capacidad de brindar información sobre la odometría del sistema.

1.5.4.1 Odometría

La odometria es el estudio de la estimación de la orientación y posición de un vehículo o sistema robótico en general a partir de recopilar información sobre sus movimientos previos, por lo general la odometría se basa cálculos para la medición de rotación y desplazamiento de las ruedas o ejes de un vehículo [17]. Para reducir el error de la odometría, la mayoría de los sistemas integran la odometría laser u odometría visual.

- Odometría Laser: Implica el uso de sensores laser tales como LIDAR para estimar la posición y orientación del robot, este tipo de odometría brinda información valiosa acerca de la existencia de obstáculos presentes y una capacidad de mapeo rápida. [18].
- Odometría visual: Hace uso de cámaras para estimar la posición y movimiento del robot a partir de la captura de imágenes, además, brinda la capacidad de integrar algoritmos de inteligencia artificial para reconocimiento de objetos u obstáculos [19].

1.5.4.2 Localización

Se refiere al cálculo de la ubicación o coordenadas del robot a partir de información generalmente proveniente de Sensores [20] o GPS, la localización permite determinar la posición y orientación del robot con respecto al mapa.

1.5.4.3 Mapeo

Es el proceso de generar un mapa del entorno a partir de distintos sensores [20], el mapa puede ser una representación tanto 2D como 3D del espacio, la calidad del mapa generado depende directamente de la resolución de los sensores y el algoritmo de planeación implementado para distinguir entre objetos movibles o límites del entorno.

1.5.4.4 Planificación

La planificación de camino corresponde a la capacidad del sistema para movilizarse entre puntos o acorde a una trayectoria predefinida, el análisis de planificación se realiza a partir de dos aspectos: Planificación global y planificación local, y las técnicas de planificación se basan en que tanta información de su entorno se conoce, un algoritmo clásico utilizado es el algoritmo Dijkstra [21], que se basa en encontrar el camino más corto entre dos puntos teniendo en cuenta condiciones de peso, es decir, obstáculos, en planificación local, el algoritmo DWA el robot planifica la mejor ruta posible entre puntos y ajusta su velocidad para controlarse de manera dinámica con respecto a la nueva información proveniente del entorno, dando características más reales como evitar obstáculos [21].

1.5.5 Sensor LiDAR

Lidar o por sus siglas en inglés, light detection and ranging, es un sensor que emite rayos láser infrarojo a su alrededor, estos láseres rebotan en objetos cercanos y son nuevamente receptados por el sensor, el sensor considera los tiempos de viaje de cada pulso para calcular la distancia recorrida teniendo en cuenta variables físicas como la Velocidad de la luz [22].

1.5.6 Cámara de profundidad

Las cámaras de profundidad tienen la capacidad de proporcionar información acerca de la distancia de los objetos en una foto a partir de patrones de luz, son comúnmente usadas en tareas de reconstrucción de estructuras, modelados 3D, y estimación de localización de obstáculos [23], actualmente existen productos comerciales como las cámaras Kinect de Microsoft que cuentan con esta tecnología.

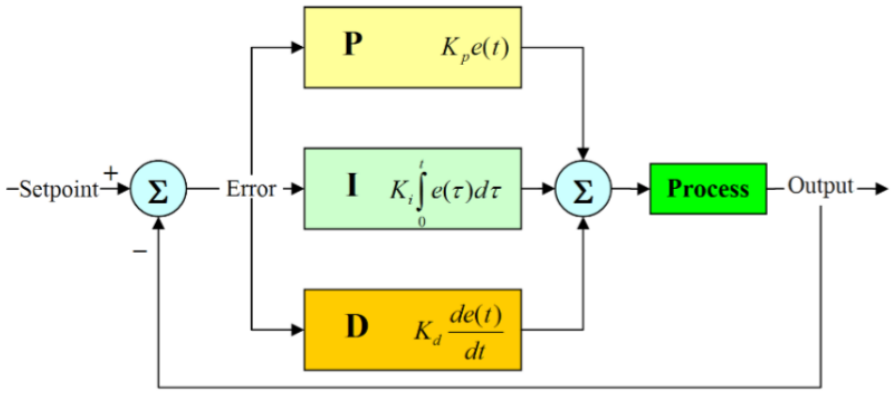
1.5.7 Sistema de Control

Un sistema de control es el encargado de regular el comportamiento de los subsistemas de un proceso, este sistema es el que permite el movimiento y funcionamiento de los actuadores o distintas partes de un robot, además, un sistema de control puede o no puede tener retroalimentación, dando lugar a los sistemas de lazo cerrado y lazo abierto respectivamente.

Existen diversos tipos de Sistemas de Control ambientado a la robótica, el controlador PID, es uno de los más usados debido a su versatilidad y capacidad de controlar distintos actuadores, su lazo cerrado brinda una retroalimentación que pasa por un ajuste Proporcional (P), Integral (I) y Derivativo (D), disminuyendo así el error entre la variable de entrada del sistema y su valor de salida dependiendo de la calibración de sus parámetros [24].

Figura 1.8

Diagrama de bloques de sistema de control PID



Nota. Modelo de bloque obtenido en [25]

Capítulo 2

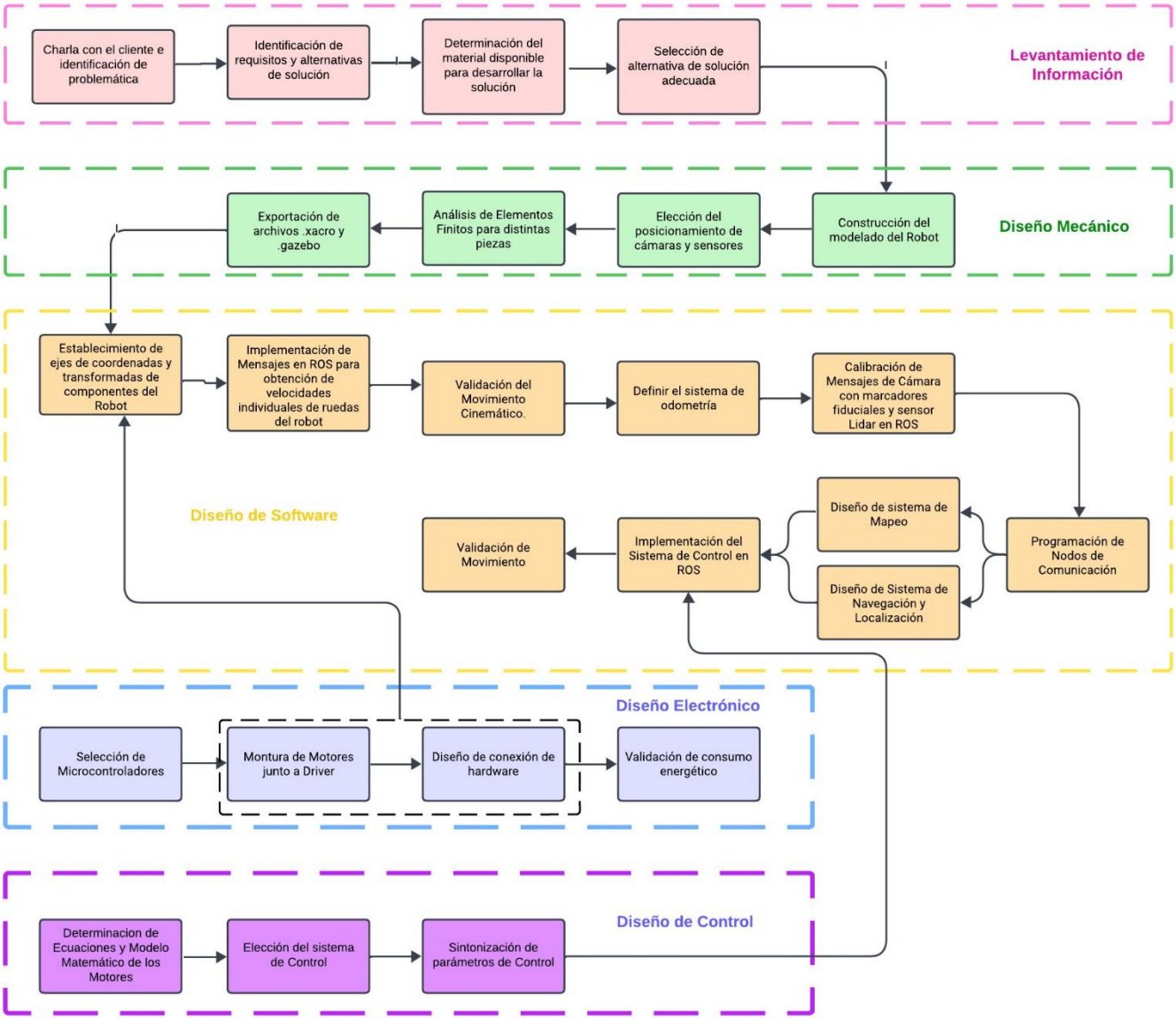
2 Metodología

2.1 Mapa de metodología

Este es el mapa de metodología de nuestro proyecto, sigue una serie de diseños individuales que convergen en un diseño funcional.

Figura 2.1

Mapa de Metodología de la solución



2.2 Estructura y diseño inicial

Se elaboró nuestra solución a partir de una plataforma ya existente la cual de nuestro cliente, es plataforma es la “ARLO Complete Robot System” de PARALLAX.

2.2.1 Plataforma robótica de ARLO

El kit DE PARALLAX está compuesto por los siguientes componentes:

Tabla 2.1

Tabla de componentes de Arlo

Componentes del kit
ARLO Base Kit (#28960)
Caster Wheel Kit, Rev B (#28961)
Motor Mount and Wheel Kit – Aluminum (#28962)
ARLO Top Deck Kit (#28965)
ARLO Power Distribution Board (#28996)
Propeller Activity Board WX (#32912)
USB A to Mini B Cable (#805-00006)
Hardware Pack ARLO Full Kit (#570-28966)
ARLO Battery Charger – 2.5mm (#700-00240) – or 2.1mm (#700-00245) *
Dual H-Bridge Motor Controller (#28231) with Prop Plug (#32201)
Ultrasonic Distance Sensor+ Protector Stand (#910-28015B)
12 Volt, 7.2Ah, Sealed Lead Acid Battery (#752-00007)
DVM810 Multimeter (#700-10026)
3M Safety Eyewear (#700-10003)

Nota. Información obtenida de la página de PARRALLAX [26]

Figura 2.2

Estructura del sistema robótico completo de ARLO

Nota. Imagen del robot obtenida en [26]

El sistema robótico que se puede observar en la **Figura 2.2**, es una opción perfecta que permitió implementar un robot autónomo, debido a que el kit provee robustez y software avanzado, con él se completó tareas complejas de navegación autónoma y técnicas SLAM.

Tabla 2.2

Características de Plataforma ARLO de Parallax

Características de la plataforma robótica ARLO	
Material de pisos/placas	Polietileno de alta densidad
Materiales de motores	Aluminio
Actuadores	Motores DC 12V
Tipo de encoders	Encoders de Cuadratura de 36 posiciones proporcionan hasta 144 pulsos/revolución
Controlador de motores	Placa electrónica DHB-10
Distribución de energía	Placa de distribución de potencia para varias salidas de voltaje
Voltaje de entrada	12V
Consumo de corriente	1.5 – 3 Amperios
Dimensiones (diámetro)	18 pulgadas
Peso ensamble	20 libras
Temperatura de operación	0 a 70 °C

Nota. Esta información se obtuvo en la página oficial de Parallax [26]

2.2.1.1 Motores DC

El sistema robótico ARLO de Parallax incluye dos motores DC de 12V y 96W se utilizaron para nuestro prototipo final.

Figura 2.3

Motores DC de 12V del sistema ARLO de Parallax

Nota. En la imagen se puede observar los motores junto al encoder y su bloque de ensamble, imagen obtenida de la página de Parallax [27]

Se eligió este sistema de motores porque cumplió con el torque necesario y capacidad de carga que necesitaron los requerimientos de nuestra problemática.

Tabla 2.3

Características de los motores DC del sistema ARLO

Motores DC	
Voltaje	12V
Capacidad de carga	60lb
Velocidad aproximada	95 RPM
Torque aproximado	9.6Nm
Peso	2.8lb/motor
Temperatura de operación	0 a 49 °C
Corriente	2 – 8 Amperios dependiendo del terreno
Diámetro de llantas	6 pulgadas

Nota. Información obtenida de la página oficial de Parallax [27]

2.2.1.2 Encoders de cuadratura

Los encoders son ópticos de cuadratura con 36 posiciones el cual proporcionó una retroalimentación rotacional para las ruedas del robot.

Figura 2.4

Encoder de cuadratura con 36 posiciones de Parallax

Nota. Esta imagen se obtuvo de la página oficial de Parallax [28]

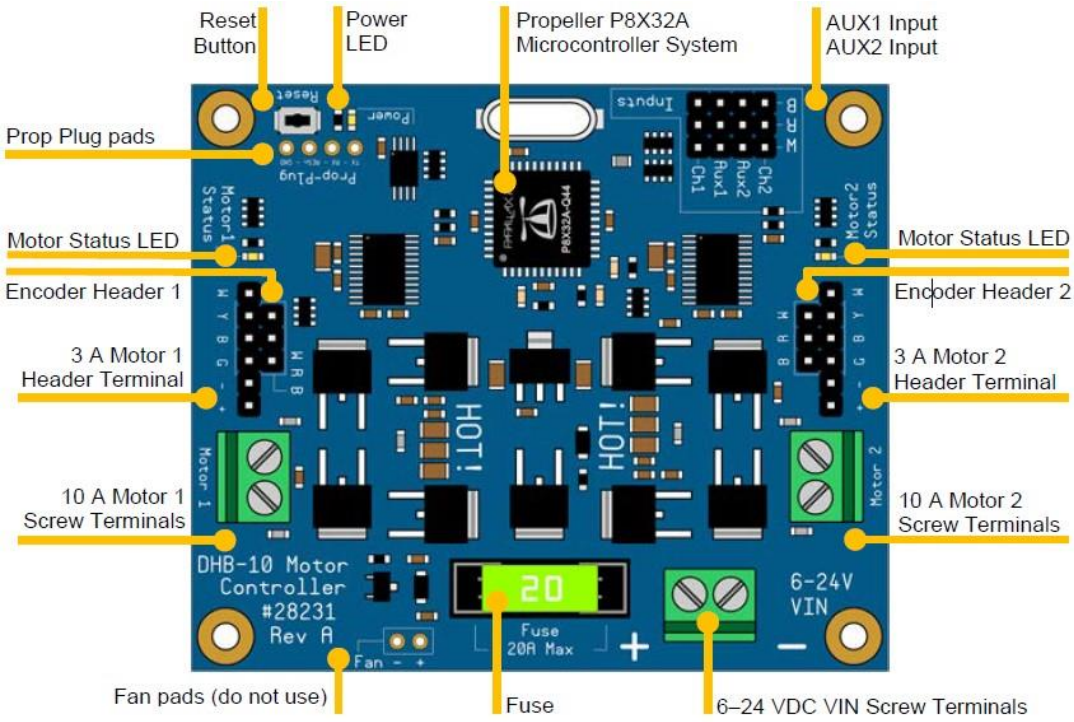
Estos encoders ofrecieron 144 posiciones en la salida del sensor, se utilizó esta información en nuestro controlador PID de mejor manera y también se utilizó para obtener la odometría de ruedas del robot.

2.2.1.3 Controlador de motores DC DHB-10

El principal controlador de motores DC que se utilizó fue el DHB-10 el cual es fue proporcionado por Arlo.

Figura 2.5

Controlador DHB-10 de Parallax



Nota. Esta imagen muestra las entradas y salidas del controlador DHB-10 de Parallax y se obtuvo de la página oficial de Parallax [29]

Este controlador junto a la comunicación serial con el ESP32 y la señal de los encoders conectó los motores con ROS, moviendo finalmente el robot utilizando ROS.

Tabla 2.4

Características del controlador de motores DC DHB-10 de Parallax

Características de DHB-10	
Microcontrolador	Propeller P8X32A
Núcleos	8
EEPROM	64kB
Cristal	5MHz
Voltaje	6 – 24 VDC
Corriente máxima por motor	12 amperios
Baudrate	19200 – 115200
Temperatura de operación	32 – 158 °C

Nota. Toda esta información se obtuvo de la página oficial de Parallax [29]

2.2.1.4 Baterías utilizadas en el robot

Las baterías que alimentaron todo nuestro sistema fueron dos baterías de ácido-plomo de 12 voltios y 7.2 amperios-hora, ambas fueron conectadas en paralelo para obtener mayor corriente, y por el tanto mayor duración de funcionamiento.

2.2.2 RPLIDAR A1

Es un escáner laser 2D de 360 grados, utiliza tecnología de luz láser para medir distancias y crear un escaneo tridimensional del entorno.

Figura 2.6

RPLIDAR A1 de Slamtec

Nota. Esta imagen se obtuvo de la página oficial de Slamtec [30]

El RPLIDAR A1 tiene compatibilidad con ROS, esto permitió integrar este escáner a nuestros algoritmos de SLAM y también para elaborar mapas.

Tabla 2.5

Características principales del RPLIDAR A1

Características del RPLIDAR A1	
Dimensiones	98.5mm x 70mm x 60mm
Peso G. W	170g
Batería	excluida
Rango de distancia	0.15 - 6m, objetos blancos
Rango angular	0-360 grados
Resolución de distancia	inferior a 0.5mm
Resolución Angular	1 grado
Duración de la muestra	0.5 ms
Frecuencia de muestra	2000 ~ 2010Hz
Velocidad de escaneo	1 ~ 10Hz, 5.5Hz típico,

Nota. Toda la información de esta tabla se obtuvo directamente de la página del fabricante [30]

2.2.3 Cargador de baterías de 12V

Se recargó las dos baterías con un cargador de 12V, este cargador es universal para diferentes tipos de baterías, su nombre es imax B6 mini y su fabricante es SKYRC.

Figura 2.7

Cargador universal de baterías imax B6 mini

Nota. Esta imagen fue obtenida de la página oficial del fabricante [31]

2.2.4 Placa de desarrollo ESP32

El módulo ESP32 es un microcontrolador con diferentes características que se observan en la **Tabla 2.6**, con este componente se controló el movimiento del robot, el ESP32 se conectó directamente al controlador DHB-10 utilizando los pulsos que recibe el pin ‘AUX1’ y ‘AUX2’ del controlador que se puede observar en la **Figura 2.5**.

Figura 2.8

ESP32 DevKit de ESPRESSIF

Nota. Esta imagen de obtuvo de la página oficial de ESPRESSIF [32]

Tabla 2.6

Tabla de características del ESP32 DevKit de ESPRESSIF

Especificaciones de ESP32	
Característica	Especificación
MCU	Xtensa Dual Core 32-bit LX6
Conexión inalámbrica	WiFi y BLE
Frecuencia	240MHz
SRAM	512 kBytes
Flash	4MB
GPIO	36
Hardware/Software	PWM 1/16 Channel
SPI/I2C/I2S/UART	4/2/2/2
ADC	12bit
CAN	1
Ethernet Mac Interface	1/16 Channel
Temperatura de trabajo	-40 C - 125 C

Nota. Esta información se obtuvo de la página oficial de ESPRESSIF [32]

2.3 Requerimientos y análisis de soluciones

Se planteó los requerimientos que necesitaba nuestro prototipo final:

Tabla 2.7

Requerimientos generales de nuestro proyecto

	Requerimientos de la solución
1.	Que el robot sea configure una sola vez y que luego de eso funcione autónomamente ubicando las estaciones y realizando funciones de servicio.
2.	Que el robot funcione correctamente al momento de realizar un trabajo de servicio hacia el usuario.
3.	Que el robot tenga la batería suficiente para trabajar autónomamente sin detenerse al menos por una jornada de 4 horas.
4.	Que el chasis del robot sea capaz de contener o soportar todos los componentes internos del robot.
5.	Que sea capaz de ubicar obstáculos y evadirlos de la mejor manera sin provocar inconvenientes al usuario.
6.	Que tenga una velocidad óptima al momento de realizar sus tareas de servicio.
7.	Que sea fácil realizar el mantenimiento del robot, así evitando que este se pueda dañar o que al ser muy complejo no se pueda realizar su mantenimiento.

2.3.1 Soluciones propuestas

Teniendo estos requerimientos se planteó tres soluciones para resolver nuestra problemática:

Tabla 2.8

Soluciones propuestas a partir de los requerimientos

	Soluciones propuestas
1.	Diseño de un robot móvil omnidireccional de servicio con planeación, mapeado y posicionamiento basado en odometría laser con controlador NVIDIA para tareas de navegación de entorno basado en mapas y navegación social para evasión de obstáculos.
2.	Diseño de un robot móvil diferencial de servicio, con planeación, mapeado y posicionamiento basado en odometría visual e inteligencia artificial con computadora como controlador para realizar tareas de navegación de entorno basado en mapas y sistemas de lógica difusa para evasión de obstáculos.
3.	Diseño de un robot móvil diferencial de servicio con planeación, mapeado y posicionamiento basado en odometría visual y de ruedas con controlador NVIDIA para realizar tareas SLAM, reconocimiento de marcadores de posicionamiento y navegación social para evasión de obstáculos.

2.3.2 Análisis de soluciones

Teniendo estas tres soluciones posibles, se realizó varias matrices de decisión para elegir la solución óptima.

Primero se estableció las características más importantes y su jerarquía de importancia, estas características son las siguientes:

Tabla 2.9

Características importantes de nuestra solución

Características de nuestra solución
Escalabilidad
Eficiencia energética
Carga computacional
Replicabilidad
Adaptabilidad
Facilidad de mantenimiento

Para definir la ponderación de las características se analizó a partir de la **Tabla 2.10**.

Tabla 2.10

Ponderación de criterios

Criterio	Carga computacional	Eficiencia energética	Facilidad de mantenimiento	replicabilidad	adaptabilidad	escalabilidad	Sum +1	Ponderación
Carga computacional	X	0.5	1	1	1	1	5.5	0.26
Eficiencia energética	0.5	X	1	1	1	1	5.5	0.26
Facilidad de mantenimiento	0	0	X	1	1	1	4	0.19
replicabilidad	0	0	0	X	0.5	1	2.5	0.11
adaptabilidad	0	0	0	0.5	X	1	2.5	0.11
escalabilidad	0	0	0	0	0	X	1	0..05
						Total	21	

Nota. Esta tabla se obtuvo en base al orden de jerarquía de las características y los requerimientos

Obteniendo así el siguiente orden de Jerarquía:

$$Carga\ Computacional = Eficiencia\ Energética > Facilidad\ de\ Mantenimiento > Replicabilidad = Adaptabilidad > Escalabilidad$$

Luego, se crearon tablas de ponderación específicas de cada criterio, donde se analizó la viabilidad de las soluciones de manera individuales para finalmente comparar las alternativas de solución a partir de sus valores ponderados.

Tabla 2.11

Tabla de ponderación de la Carga Computacional

Carga Computacional	Alternativa 1	Alternativa 2	Alternativa 3	SUM + 1	Ponderación
Alternativa 1	X	1	1	3	0.5
Alternativa 2	0	X	0	1	0.167
Alternativa 3	0	1	X	2	0.33
Total				6	

Nota. $Alternativa\ 1 > Alternativa\ 3 > Alternativa\ 2$

Tabla 2.12

Tabla de ponderación de la Eficiencia Energética

Eficiencia Energética	Alternativa 1	Alternativa 2	Alternativa 3	SUM + 1	Ponderación
Alternativa 1	X	1	0	2	0.33
Alternativa 2	0	X	0	1	0.167
Alternativa 3	1	1	X	3	0.5
Total				6	

Nota. $Alternativa\ 3 > Alternativa\ 1 > Alternativa\ 2$

Tabla 2.13

Tabla de ponderación de la Facilidad de Mantenimiento

Facilidad de Mantenimiento	Alternativa 1	Alternativa 2	Alternativa 3	SUM + 1	Ponderación
Alternativa 1	X	1	0	3	0.33
Alternativa 2	0	X	0	1	0.167
Alternativa 3	1	1	X	2	0.5
Total				6	

Nota. $Alternativa\ 3 > Alternativa\ 1 > Alternativa\ 2$

Tabla 2.14

Tabla de ponderación de la Replicabilidad

Replicabilidad	Alternativa 1	Alternativa 2	Alternativa 3	SUM + 1	Ponderación
Alternativa 1	X	0.5	0	3	0.25
Alternativa 2	0.5	X	0	1	0.25
Alternativa 3	1	1	X	2	0.5
Total				6	

Nota. $Alternativa\ 3 > Alternativa\ 1 = Alternativa\ 2$

Tabla 2.15

Tabla de ponderación de la Adaptabilidad

Adaptabilidad	Alternativa 1	Alternativa 2	Alternativa 3	SUM + 1	Ponderación
Alternativa 1	X	0	0	3	0.167
Alternativa 2	1	X	1	1	0.5
Alternativa 3	1	0	X	2	0.33
Total				6	

Nota. $Alternativa\ 2 > Alternativa\ 3 > Alternativa\ 1$

Tabla 2.16

Tabla de ponderación de la Escalabilidad

Escalabilidad	Alternativa 1	Alternativa 2	Alternativa 3	SUM + 1	Ponderación
Alternativa 1	X	1	0.5	3	0.417
Alternativa 2	0	X	0	1	0.167
Alternativa 3	0.5	1	X	2	0.417
Total				6	

Nota. $Alternativa\ 3 = Alternativa\ 1 > Alternativa\ 2$

Tabla 2.17

Tabla de Conclusiones

Conclusiones	Carga computacional	Eficiencia energética	Facilidad de mantenimiento	replicabilidad	adaptabilidad	escalabilidad	Sum +1	Ponderación
Alternativa 1	0.13	0.09	0.06	0.03	0.02	0.02	0.35	2
Alternativa 2	0.04	0.04	0.03	0.03	0.06	0.01	0.22	3
Alternativa 3	0.09	0.13	0.1	0.06	0.04	0.02	0.43	1

Nota. La solución tres fue la opción óptima y por ende la alternativa elegida

Finalmente, con esta alternativa 3 que fue la mejor ponderada, se procedió a elegir los diferentes componentes de hardware que se aplicó en nuestro prototipo final.

2.4 Escenario de funcionamiento

Los escenarios principales en los cuales se desenvolvió el robot fueron espacios cerrados sin ningún desnivel y en un ámbito social. Ambos escenarios serán descritos en las siguientes secciones

2.4.1 Escenario de identificación de espacio desconocido

Este escenario es una etapa inicial que nuestro robot tuvo que cumplir, el robot utilizó su sensor Lidar para mapear un sitio desconocido, las características principales de este escenario son las siguientes:

- El robot deberá utilizar un algoritmo de SLAM para mapear un ambiente desconocido.
- Se tendrá que guardar el mapa realizado por el Lidar utilizando librerías de ROS.

- Se tendrá que identificar los obstáculos no dinámicos como las paredes, puertas o mesas.

2.4.2 Escenario de navegación en un espacio recurrente

En este escenario se utiliza un mapa precargado desde ROS donde el robot navegó autónomamente y a su vez cumplió las tareas de servicio, las características iniciales de este escenario fueron las siguientes:

- Pueden existir obstáculos dinámicos.
- Se debe obtener la odometría del robot para poder utilizarla en la navegación del robot.
- Se debe identificar las estaciones de trabajo para que el robot pueda cumplir sus tareas de servicio de manera eficiente.
- Debe utilizar un solo mapa de todo el lugar el cual funcionará como plano de trabajo del robot.

2.5 Hardware adicional

En esta sección se especifica el hardware adicional que se consideró en base a los requerimientos previamente establecidos.

2.5.1 Cámara de odometría Intel RealSense T265

La cámara de odometría que se utilizó en nuestro robot para obtener su posición en el mapa y así se completó la navegación autónoma.

Figura 2.9

Cámara de rastreo Intel RealSense T265

Nota. Esta imagen fue obtenida de la página oficial de Intel [33]

2.5.2 Cámara de profundidad Intel RealSense D435i

Para completar la tarea de visión por computadora, el robot de servicio utilizó la cámara de Intel RealSense D435i debido a que tiene RGB y detecta profundidad. Las detecciones de personas se realizaron con ayuda de esta cámara y el paquete Darknet ROS [34]

Figura 2.10

Cámara de profundidad Intel RealSense D435i

Nota. Esta imagen fue obtenida de la página oficial de Intel [33]

Tabla 2.18

Características de la cámara D435i

Características de cámara D435	
Resolución de profundidad estéreo	1280x720
Resolución RGB	1920x1080
Campo de visión	90 grados
Cuadros por segundo	90
Rango de medición de distancia	0.3 – 10 metros

Nota. esta información se obtuvo de la página principal de Intel [33]

2.5.3 Placa de desarrollo Jetson Nano

La Jetson Nano de NVIDIA fue el centro de cómputo principal de nuestro robot, la ventaja principal por la cual se eligió la Jetson Nano fue el ahorro de energía y su adaptabilidad con la inteligencia artificial.

Figura 2.11

NVIDIA Jetson Nano

Nota. Esta imagen se obtuvo de la página oficial de NVIDIA [35]

2.6 Software

2.6.1 Técnicas SLAM

Existe un gran repertorio de paquetería para realizar tareas SLAM, en ROS se pueden utilizar distintos paquetes, entre los más destacables se encuentran, Cartographer, Hector-SLAM, Gmapping y RTAB.

Se utilizó los criterios de evaluación especificados en la **Tabla 2.19**, de esta forma se obtuvo la técnica SLAM a utilizar como se muestra en la **Tabla 2.20**.

Tabla 2.19

Criterios de Evaluación para Técnica SLAM

Técnica SLAM			
Calidad de Mapa Generado	**Capacidad de Corrección de Mapa en Tiempo Real**	**Eficiencia computacional**	**Soporte Técnico de Paquetería**
35%	25%	20%	20%

Tabla 2.20

Matriz de decisión para elección de Técnica SLAM a utilizar

Técnica SLAM					
	Calidad de Mapa Generado	**Capacidad de Corrección de Mapa en Tiempo Real**	**Eficiencia computacional**	**Soporte Técnico de Paquetería**	**Total**
Cartographer	4.5	5	4	4	4.425
Hector-SLAM	3	4	4	4	3.65
Gmapping	4	4	4.5	4	4.1
RTAB	4.5	2	3.5	3	3.375

Para las tareas SLAM, se utilizó Cartographer [36] para realizar mapas en 2D o 3D de ambientes desconocidos a partir de información de Sensor Lidar y Odometría del sistema robótico.

2.6.2 Algoritmos de Navegación y Localización

Debido a la necesidad de optimizar recursos y dado a que ya no se necesita mapear el entorno de nuevo, se decidió utilizar otro algoritmo para el segundo escenario, en ROS hay 3 algoritmos de Navegación y Localización destacable los cuáles son: AMCL (Adaptive Monte Carlo Localization), Cartographer Navigation, Sbpl Lattice Planner

Se utilizó los criterios de evaluación especificados en la **Tabla 2.21**, de esta forma se obtuvo el algoritmo de Navegación y localización a utilizar como se muestra en la **Tabla 2.22**

Tabla 2.21

Criterios de Evaluación para Algoritmo de Navegación y Localización

Algoritmo de Navegación			
Exactitud de Pose estimada	**Capacidad de evasión de obstáculos**	**Eficiencia computacional**	**Soporte Técnico de Paquetería**
35%	35%	15%	15%

Tabla 2.22

Matriz de decisión para elección de algoritmo de Navegación y Localización a utilizar

	Algoritmo de Navegación				
	Exactitud de Pose estimada	**Capacidad de evasión de obstáculos**	**Eficiencia computacional**	**Soporte Técnico de Paquetería**	**Total**
AMCL	4.5	4	4	4	4.175
Cartographer Navigation	3.5	4	3	4	3.675
Sbpl Lattice Planner	3.5	4.5	3	2	3.55

Para la Navegación y localización se optó por usar el algoritmo AMCL [37], el cuál es un paquete que permite brindar costos a mapas tanto locales como globales, lo que permitió brindar personalización en la capacidad de evadir obstáculos dinámicos.

2.6.3 Comunicación entre componentes.

Para poder enlazar componentes no nativos de ROS, como lo es la ESP32, se necesitó un protocolo para enlazar el Hardware y empezar una comunicación basada en mensajes a través de distintos topics, para poder comunicar el código de Micropython de la ESP32 con los sistemas de mensajes personalizados de ROS, se utilizó el paquete rosserial, más específicamente, rosserial_python [38], de esta manera se estableció un canal de comunicación que permitió intercambiar mensajes serializados entre los periféricos.

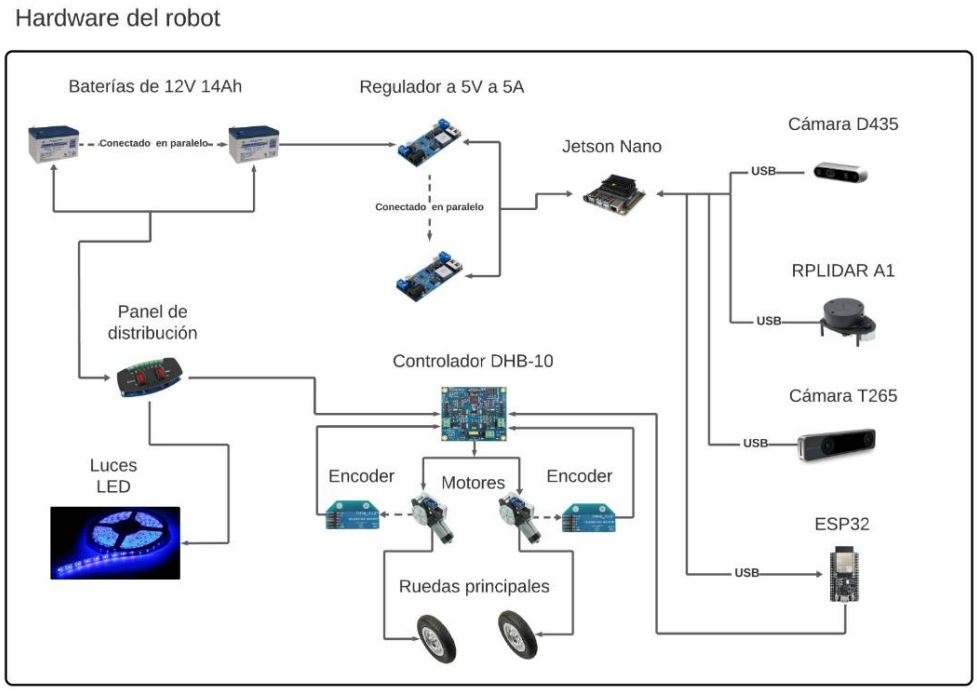
2.7 Arquitectura del sistema del robot

2.7.1 Conexión Hardware del Robot

Se realizó un diagrama de conexiones de hardware donde se halló la relación que existe en cada uno de estos componentes.

Figura 2.12

Diagrama básico de cómo se conectan nuestros diferentes componentes de hardware



2.7.2 Lógica de Comunicación en funcionamiento

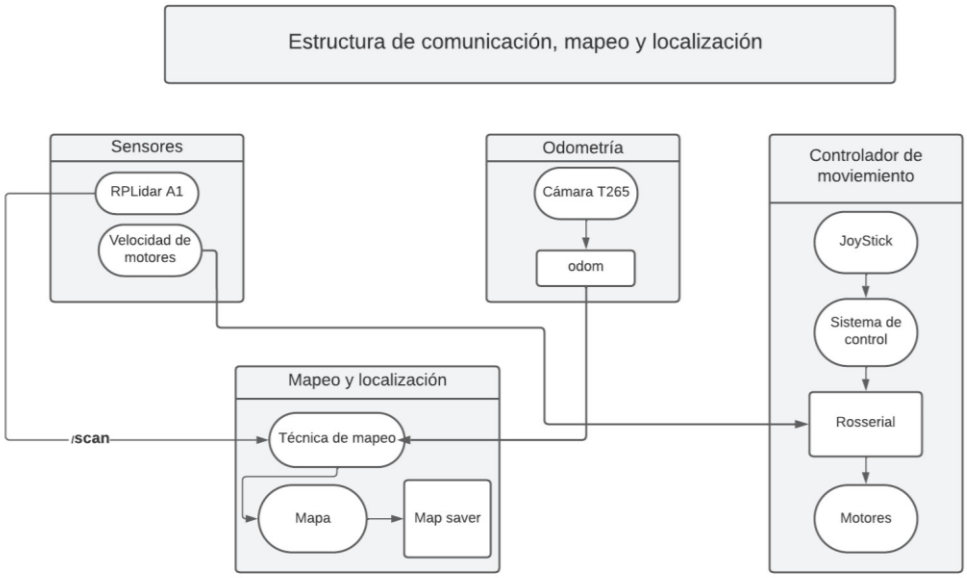
El Robot realizó tareas diferentes según el escenario de funcionamiento en el que se encuentre, por ende, su lógica es diferente en cada uno de estos.

2.7.2.1 Estructura de Comunicación para Mapeo y Localización

Se realizó un diagrama de la arquitectura de comunicación donde se enlazan los nodos para cumplir las tareas de mapeo y localización.

Figura 2.13

Estructura de comunicación, mapeo y localización del robot

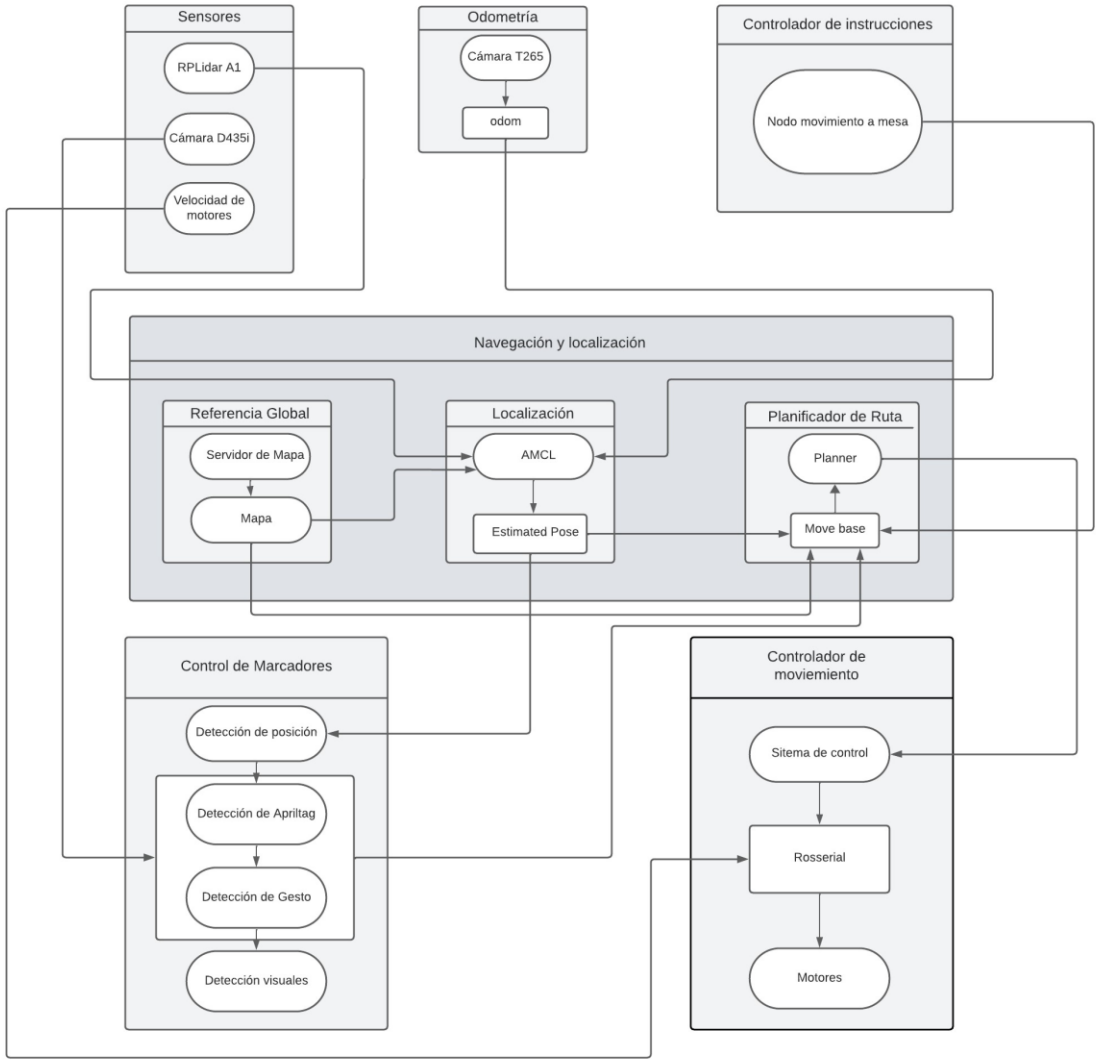


2.7.2.2 Estructura de Comunicación para Navegación y Localización

Para el segundo escenario, el Robot fue capaz de realizar todas sus funcionalidades de trayectoria, movimiento autónomo y planeación de rutas, se planteó el siguiente diagrama de comunicación.

Figura 2.14

Estructura de Comunicación para Navegación y Localización

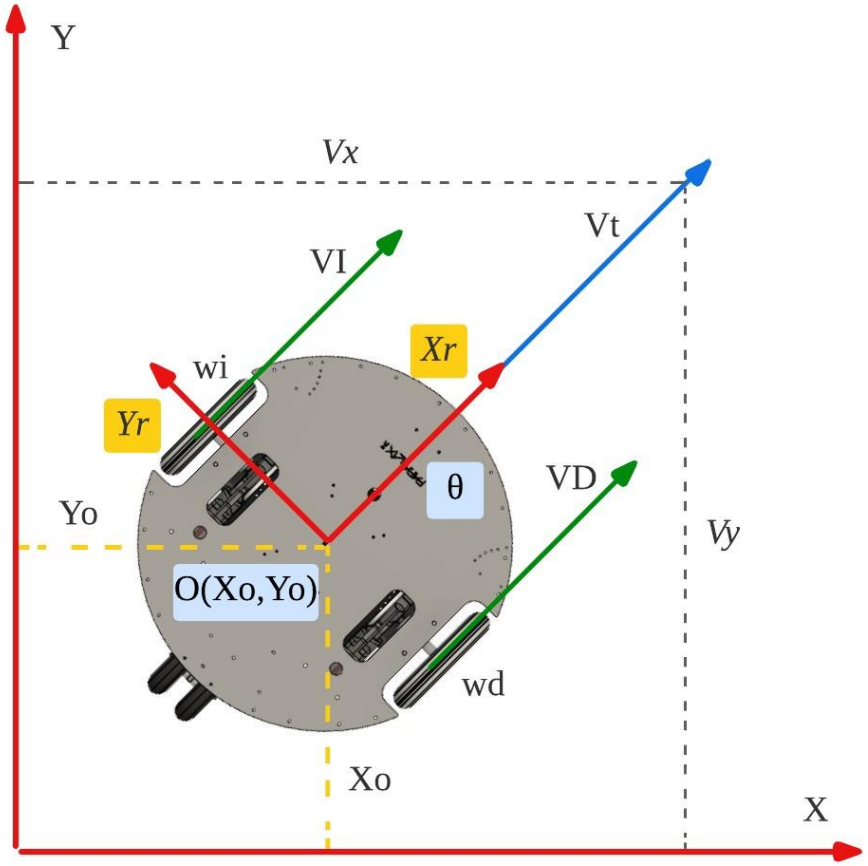


2.8 Cinemática del robot diferencial

Se planteó las siguientes ecuaciones de nuestro sistema que luego se incorporaron a un paquete de ROS llamado *'differential_drive'*, el cual sirvió para el control del robot.

Figura 2.15

Diagrama de la cinemática de nuestro robot diferencial



En la **Figura 2.15** existen dos cuadros referenciales con los cuales se obtuvo nuestras ecuaciones cinemáticas y dinámicas, el plano inercial y el plano del robot.

Tabla 2.23

Variables de nuestro sistema

Variables de nuestro sistema	
V_t	Velocidad total del robot
V_x	Velocidad en el eje X del robot en el plano inercial
V_{xr}	Velocidad en el eje X del robot en el plano del robot
V_y	Velocidad en el eje Y del robot en el plano inercial
V_{yr}	Velocidad en el eje Y del robot en el plano del robot
V_D	Velocidad de la rueda derecha
V_I	Velocidad de la rueda izquierda
w_d	velocidad angular de la rueda derecha
w_i	velocidad angular de la rueda izquierda
θ	ángulo de rotación entre el plano del robot y el plano inercial
R	el radio de las ruedas
wr	Velocidad angular del robot en el plano del robot
w	Velocidad angular del robot en el plano inercial
L	Distancia entre las ruedas principales
B	Distancia entre el robot y el eje de coordenadas

Basándo en la **Figura 2.15** se obtuvo el siguiente análisis de vectores que representan su cinemática a través de ecuaciones matemáticas.

Análisis para hallar las expresiones de las velocidades lineales:

$$VD = wr * R \tag{2.1}$$

$$VI = wl * R \tag{2.2}$$

$$Vx = Vt * cos\theta \tag{2.3}$$

$$Vy = Vt * sen\theta \tag{2.4}$$

Si en el robot solo una de las dos llantas trabaja entonces el robot girará y las velocidades de V_{xr} y V_{yr} en el punto O será igual a:

$$V_{xr} = \frac{R(wd)}{2} \tag{2.5}$$

$$V_{yr} = \frac{R(wi)}{2} \tag{2.6}$$

Se obtuvo la velocidad del robot con respecto a la velocidad angular de las ruedas y también con respecto a la velocidad lineal de las mismas.

$$Vt = \frac{R(wd + wi)}{2} \tag{2.7}$$

$$Vt = \frac{VD + VI}{2} \tag{2.8}$$

$$Vx = \frac{VD + VI}{2} * cos\theta \tag{2.9}$$

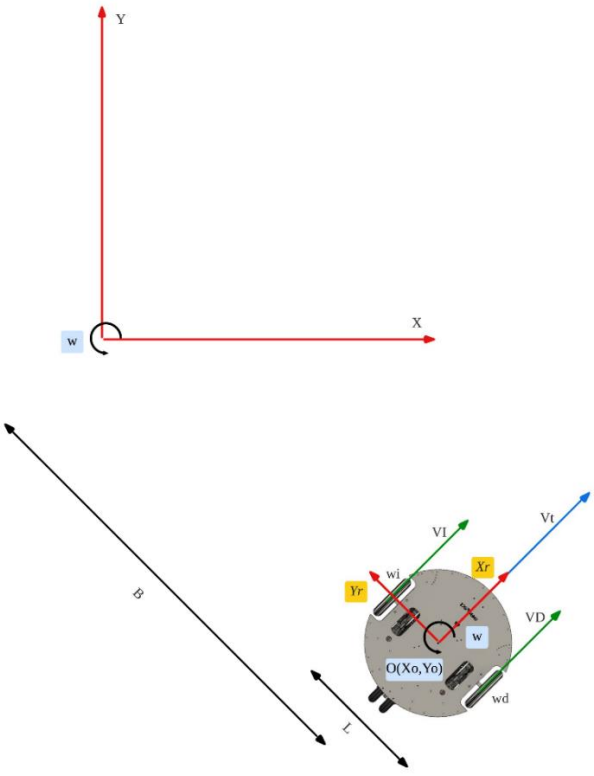
$$Vy = \frac{VD + VI}{2} * sen\theta \tag{2.10}$$

De las ecuaciones se obtuvo las siguientes características del robot:

1. Si la velocidad de cada rueda es de igual magnitud, pero diferente sentido, entonces el robot rotará sobre su propio eje.
2. Si las velocidades de las ruedas son iguales y tienen la misma dirección, entonces el robot se moverá a lo largo de su eje X.
3. Para nuestro modelo se asumió que no existe deslizamiento lateral y como nuestro robot tienes restricciones holónomicas, este no se moverá a lo largo del eje y con el marco de referencia dado.

Figura 2.16

Análisis de la velocidad de nuestro robot diferencial en un plano 2D



Como se puede observar en la **Figura 2.16** los componentes añadidos al modelo cinemático del robot son los siguientes:

Análisis para hallar la expresión matemática de la velocidad angular:

$$v = wr \tag{2.11}$$

$$V_I = w * (B - \frac{L}{2}) \tag{2.12}$$

$$V_D = w * (B + \frac{L}{2}) \tag{2.13}$$

$$B = \frac{(V_I + \frac{w * L}{2})}{w} \tag{2.14}$$

$$V_D = w * (\frac{V_I + \frac{w * L}{2}}{w} + \frac{L}{2}) \tag{2.15}$$

$$V_I = w * (\frac{V_I + \frac{w * L}{2}}{w} - \frac{L}{2}) \tag{2.16}$$

finalmente se obtuvo la ecuación para representar la velocidad angular del robot.

$$wr = \frac{V_D - V_I}{L} \tag{2.17}$$

Analizándolo desde otro punto de vista también se pudo describir la ecuación de velocidad angular de otra manera.

$$wr = \frac{R(wd - wi)}{L} \tag{2.18}$$

Luego de que se obtuvo estas ecuaciones se procedió a representar el sistema cinemático del robot en su forma matricial con respecto al marco de referencia del robot.

$$\dot{q^r} = \begin{bmatrix} \dot{x_o^r} \\ \dot{y_o^r} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ 0 & 0 \\ \frac{R}{L} & \frac{-R}{2} \end{bmatrix} * \begin{bmatrix} wd \\ wi \end{bmatrix} \tag{2.19}$$

Se obtuvo el modelo cinemático del robot en su forma matricial y con respecto al marco de referencia inercial.

$$\dot{q^I} = \begin{bmatrix} \dot{x_o^I} \\ \dot{y_o^I} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}cos\theta & \frac{R}{2}cos\theta \\ \frac{R}{2}sen\theta & \frac{R}{2}sen\theta \\ \frac{R}{L} & -\frac{R}{2} \end{bmatrix} * \begin{bmatrix} wd \\ wi \end{bmatrix} \tag{2.20}$$

Se asumió que existe una única velocidad lineal y velocidad angular que desplaza al robot siendo estas las entradas del sistema, las velocidades individuales de cada llanta serán las salidas del sistema, estos valores se utilizaron para el posterior control del prototipo final.

Teniendo la ecuación (2.8) y (2.17) se desarrolló algebraicamente y se obtuvo las ecuaciones finales de nuestro sistema que representan de manera matemática el comportamiento cinemático del robot móvil diferencial.

$$V_I = V - \frac{w * L}{2} \tag{2.21}$$

$$V_D = V + \frac{w * L}{2} \tag{2.22}$$

2.9 Diseño del controlador de motores DC

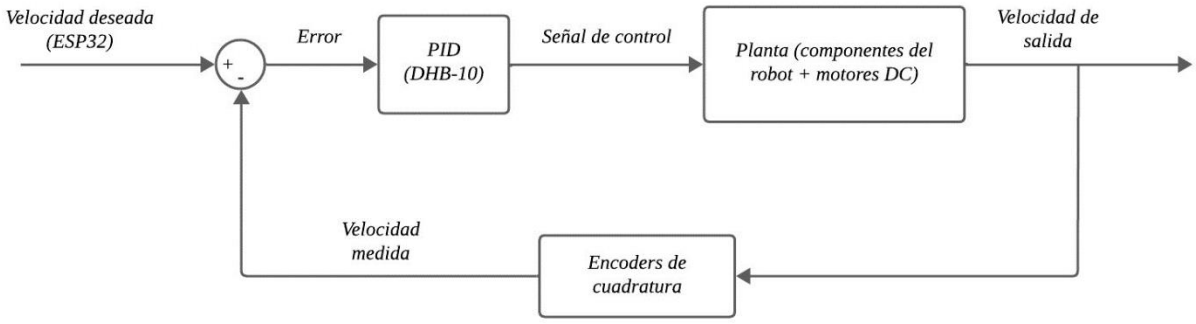
Se analizó las diferentes opciones de control de motores DC y se obtuvo las siguientes opciones:

- utilizar el modelo cinemático del robot para ajustar la velocidad de los motores DC
- utilizar las características de los motores DC para realizar un control por corriente.

Debido a la complejidad que presentó la obtención de los parámetros físicos de los motores, se optó por utilizar el primer tipo de control el cual utiliza internamente las ecuaciones (2.21) y (2.22) para manejar las velocidades de cada rueda, el modelo de control sería el siguiente:

Figura 2.17

Esquema general del control PID



Las ecuaciones que estarán internamente en nuestro PID serán las siguientes:

$$P = k_p * e(t) \tag{2.23}$$

$$I = k_i * \int_0^t e(\tau)\, d\tau \tag{2.24}$$

$$D = k_d * \frac{de}{dt}(t) \tag{2.25}$$

Finalmente se obtuvo la ecuación final de nuestro controlador:

$$PID = k_p * e(t) + k_i * \int_0^t e(\tau)\, d\tau + k_d * \frac{de}{dt}(t) \tag{2.26}$$

Se debe tener en cuenta que la empresa Parallax transformó las variables físicas de los motores y utilizó estos valores en funciones de programación integradas directamente en el firmware del controlador DHB-10, se utilizó estas funciones de control por PID que modificó el movimiento de los motores, por lo que se optó realizar un control cinemático debido a las funciones preestablecidas por Parallax y también por la configuración y posicionamiento de las ruedas.

El firmware del controlador que otorgó las funciones de movimiento, control y estatus de los motores fue realizada por la empresa Parallax y se encuentra en su página oficial [39] y se encuentra construido en idioma de programación C++, para nuestro robot el estudiante Steven Silva transformó estas funciones a micropython que finalmente es lo que utilizó la ESP32 para los comandos de movimiento.

2.10 Esquema de funcionamiento del robot

Figura 2.18

Diagrama de flujo del funcionamiento del robot

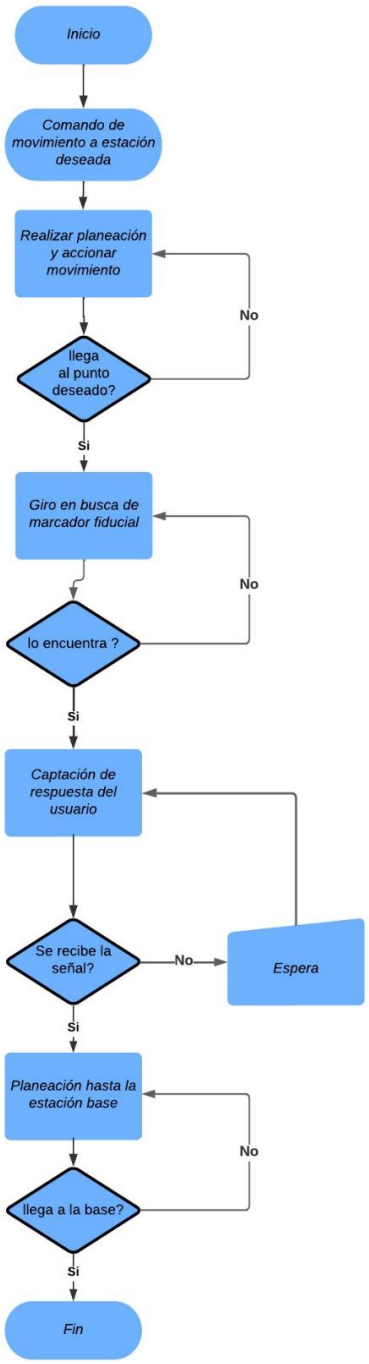
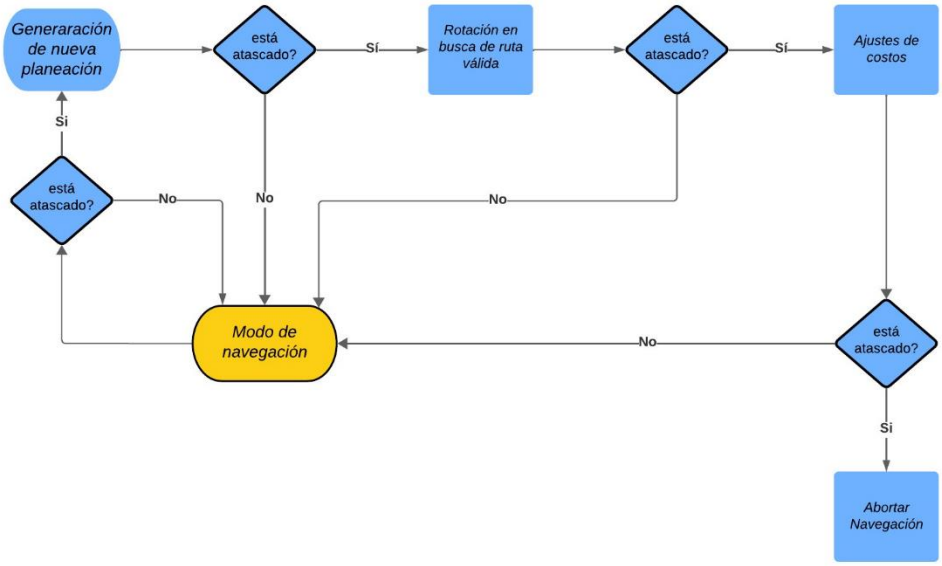


Figura 2.19

Diagrama de flujo de comportamiento ante atascamientos



2.11 Diseño mecánico del robot

Se diseñó el cuerpo del robot y se realizó un análisis de elementos finitos para confirmar que el cuerpo del robot soportó el peso a transportar, estos cálculos se mostraron en posteriores capítulos.

2.11.1 Elementos de diseño inicial

Los siguientes componentes mecánicos fueron proporcionados por el diseño inicial de Arlo, la solución se construyó a partir de esta base.

Figura 2.20

Vista explosionada del motor

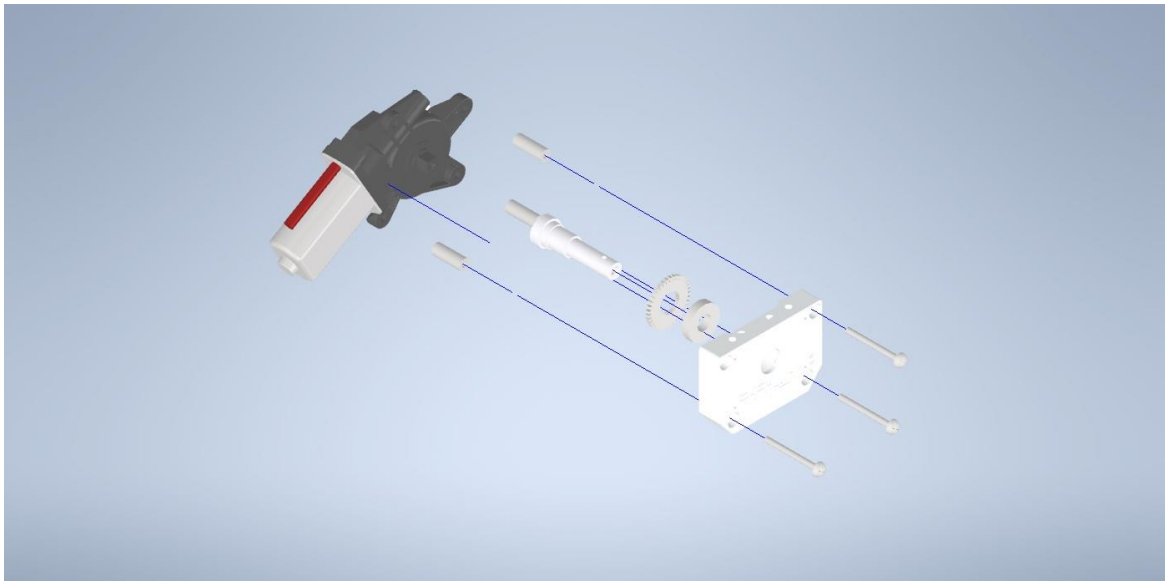


Figura 2.21

Vista explosionada de la rueda

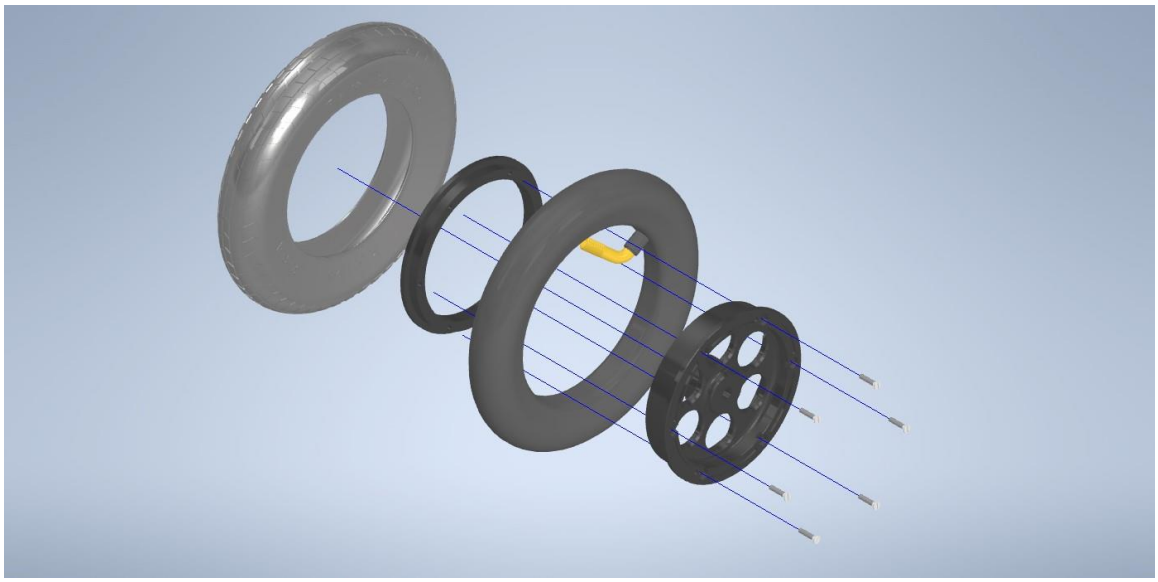


Figura 2.22

Vista explosionada de la Castor Wheel

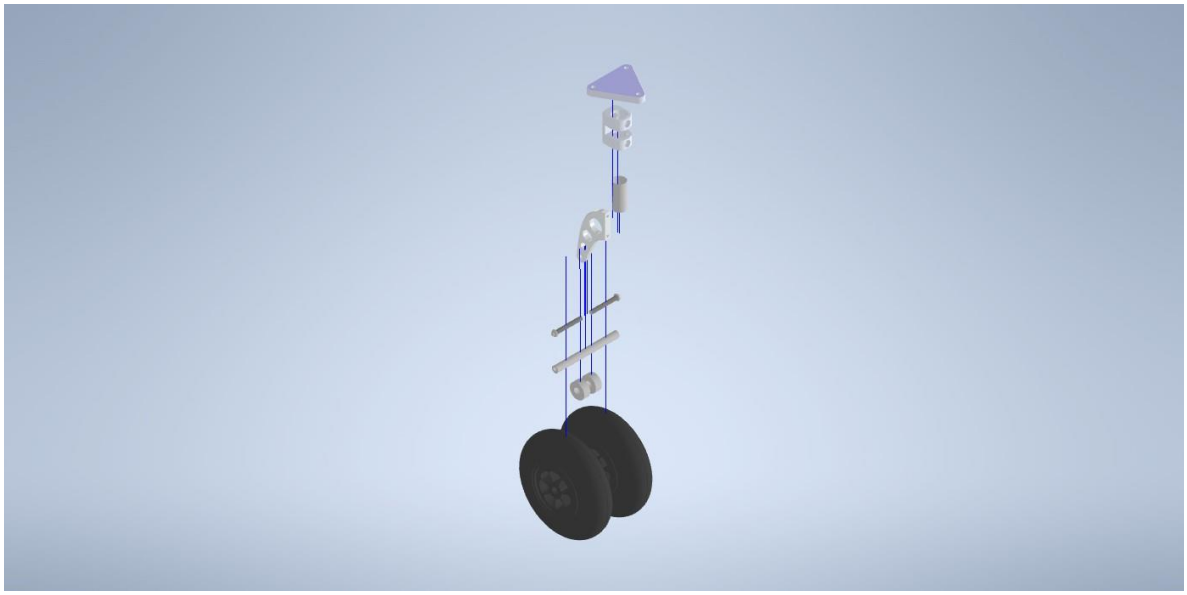
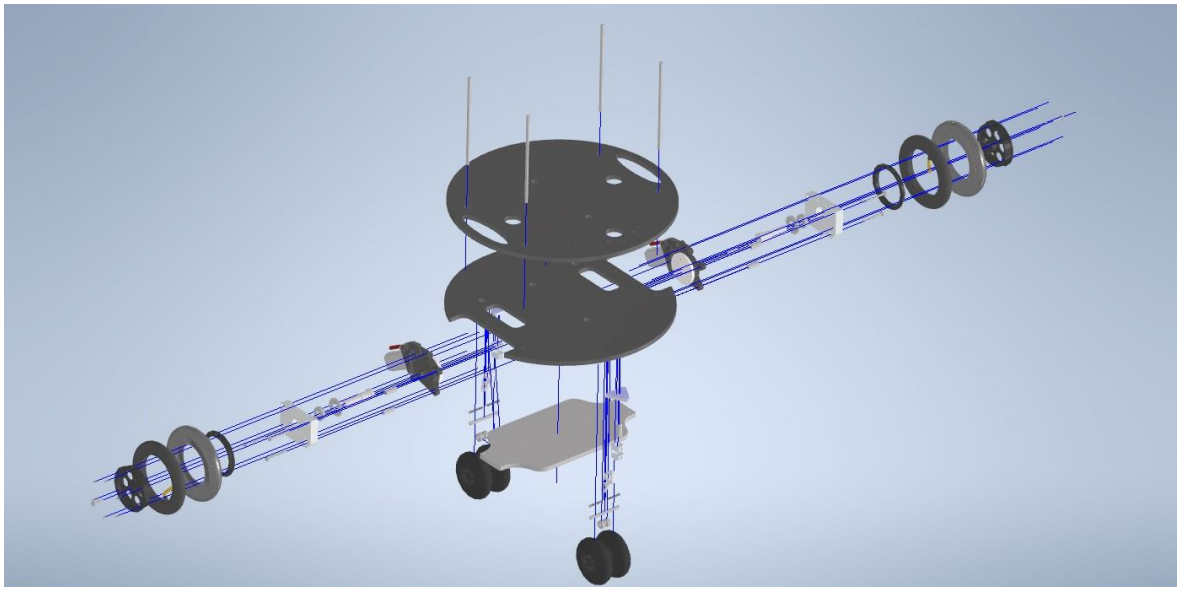


Figura 2.23

Vista explosionada de la base con todos los elementos iniciales



2.11.2 Elementos agrados al diseño inicial

Los siguientes elementos fueron diseñados por nosotros, se utilizaron las siguientes medidas basados en los resultados del análisis de elementos finitos.

Figura 2.24

Vista explosionada de la base de la cámara t265

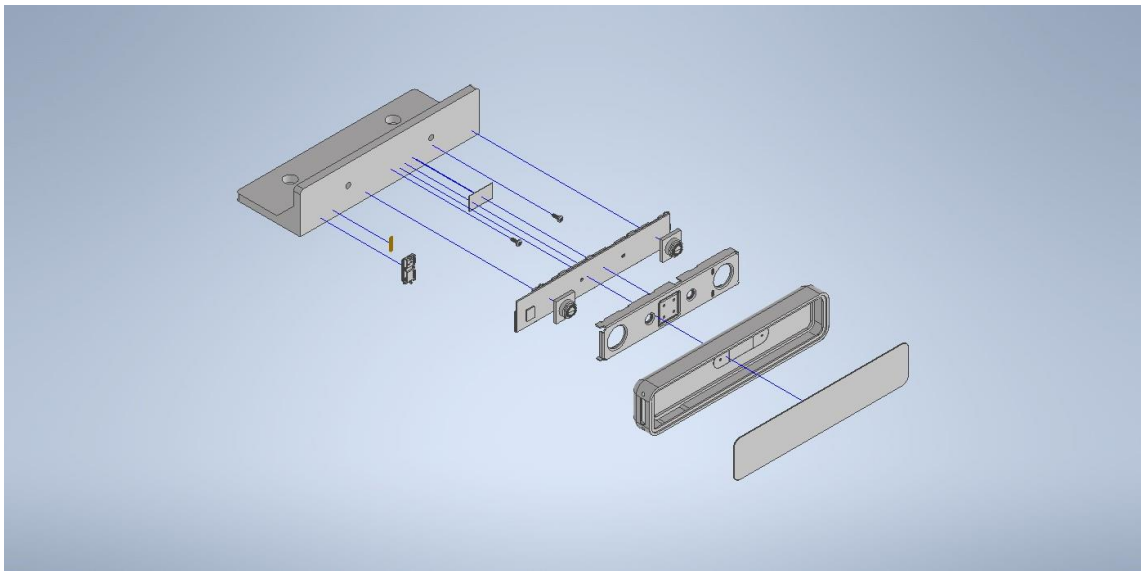
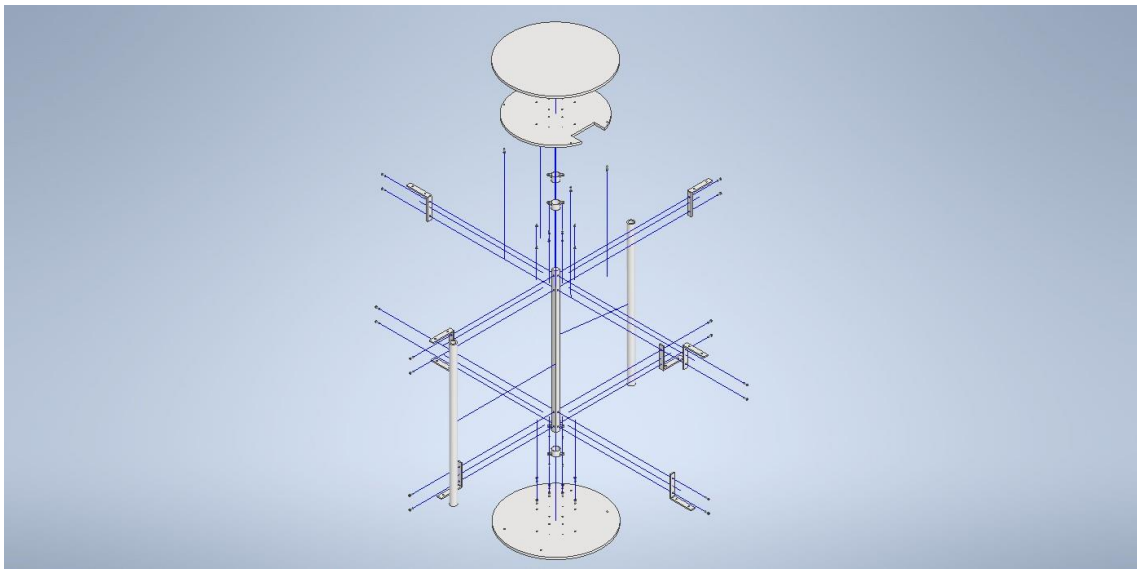


Figura 2.25

Vista explosionada del cuerpo del robot



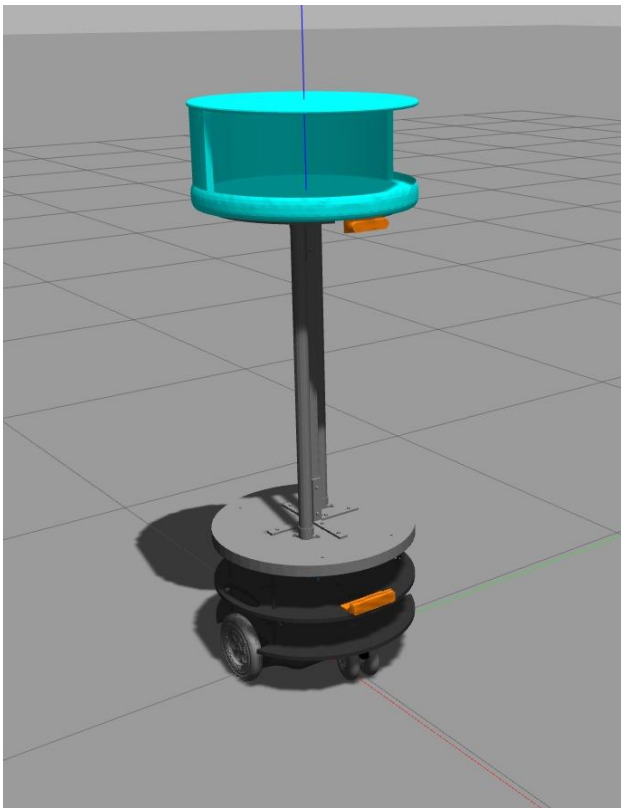
2.12 Simulación en Gazebo

Para la prueba de las distintas técnicas SLAM y técnicas de localización, se utilizó el simulador Gazebo [40] que proporcionó a un entorno de simulación 3D con diferentes *plugins* para asemejarse al prototipo real.

Se obtuvo los archivos de formato stl a formato .urdf, el cuál es el tipo de formato que admite ROS para visualización y control de los distintos *links* y *plugins* del sistema robótico.

Figura 2.26

Visualización del Robot en ROS



Se simuló un ambiente parecido a un escenario real de entrega de paquetería / comida, para esto se propone utilizar el mundo *ecuadorian.world*.

Para poder representar las distintas funcionalidades del robot, fue necesario implementar varios *plugins* [41] que simulan el funcionamiento ya sea del movimiento, como de los distintos sensores del Robot.

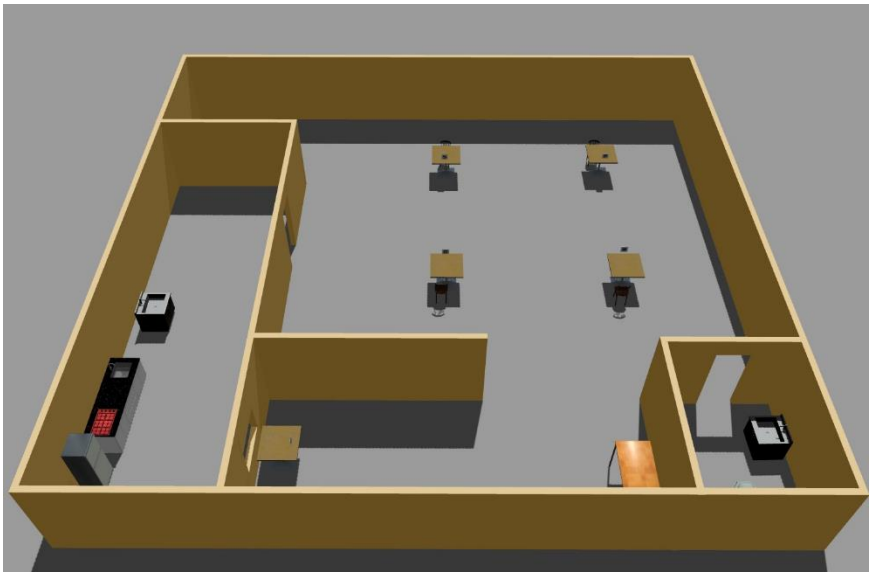
Modelo Cinemático: El *plugin* libgazebo_ros_diff_drive.so permitió simular el comportamiento de un robot diferencial, solo se necesita declarar la separación entre llantas y el diámetro de estas.

Cámara D435i: Para simular una cámara de profundidad, el *plugin* libgazebo_ros_openni_kinect.so permitió la funcionalidad de una cámara de profundidad típica, capta imágenes rgb y mensajes PointCloud [42] para la funcionalidad 3D.

RPLIDAR A1: Para representar la funcionalidad de los escáneres, el *plugin* libgazebo_ros_laser.so permite brindar este tipo de mensaje en ROS.

Figura 2.27

Ambiente de Simulación del Robot

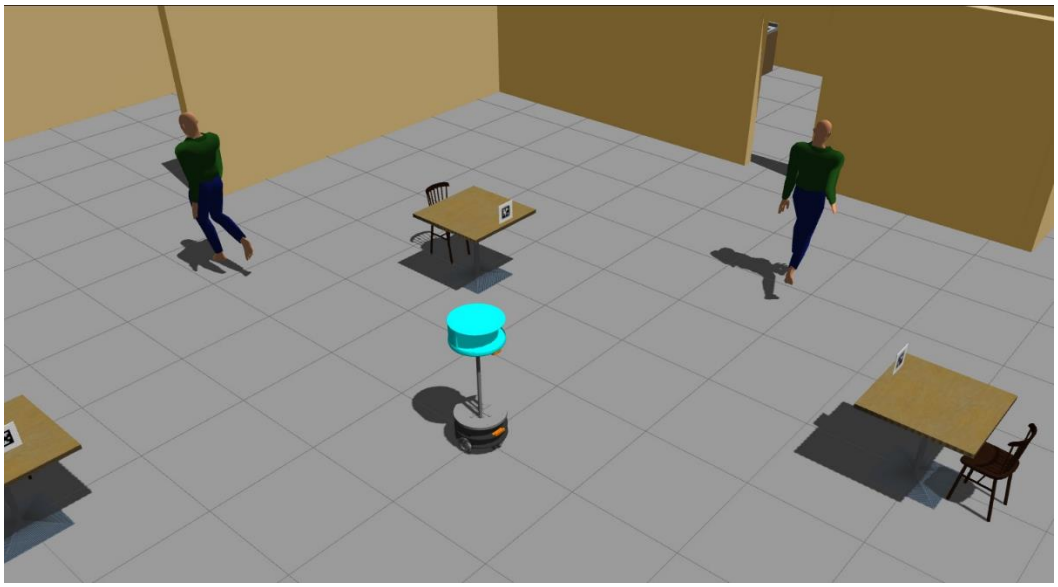


Se simuló el comportamiento del robot en este espacio y se utilizó agentes dinámicos, es decir, personas, además, se debe implementar los marcadores fiduciales.

Para simular los agentes dinámicos se configuró el mapa para agregar estaciones donde las personas van a realizar una ruta, para agregar las estaciones se utilizó paquete ros_maps_to_pedsim [43] y para simular los agentes se utilizó pedsim_ros [44].

Figura 2.28

Simulación del Robot en conjunto con Agentes dinámicos

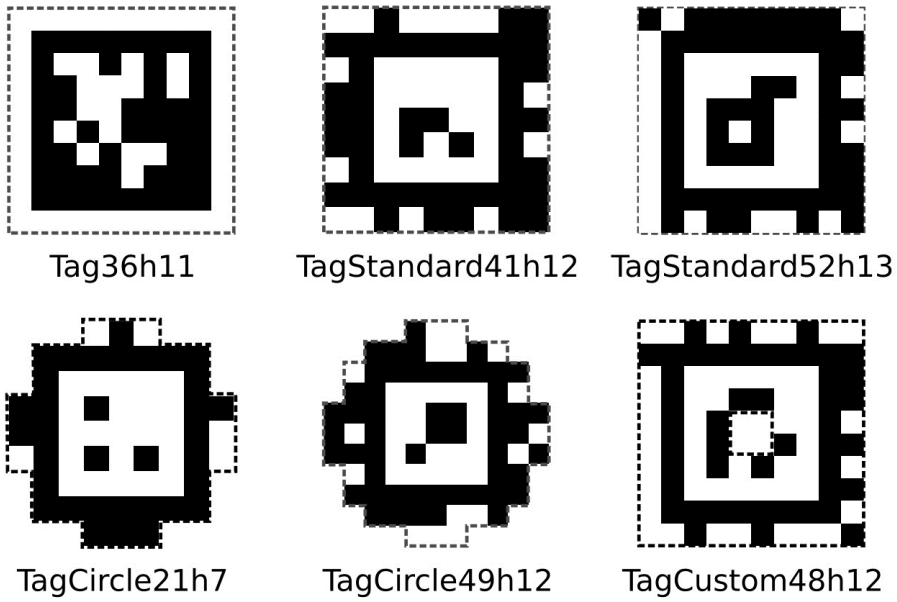


2.13 Diseño e implementación del sistema de reconocimiento de estaciones

Se estableció el reconocimiento de estaciones utilizando la librería de ROS 'AprilTag', se siguió los siguientes pasos para elegir esta opción en nuestra solución final.

Figura 2.29

Ejemplos de los diferentes tipos de marcadores de AprilTag



Nota. Esta imagen se obtuvo de la página oficial de AprilTag [45]

2.13.1 Elección del paquete para el reconocimiento de marcadores

Se utilizó la cámara D435i y un paquete especializado en ROS llamado AprilTag, se utilizó esta opción debido a que, comparado a otros métodos, los AprilTag son más fácil y rápidos de reconocer, se adaptan a diferentes ambientes, tienen mayor distancia de reconocimiento, se puede modificar su tamaño y proporciona robustez a nuestro sistema.

2.13.2 Ajuste del marcador fiducial utilizando AprilTag

Primero se calibró la cámara para que reconociera los marcadores correctamente, luego se eligió que tipo de marcador se utilizó, los diferentes tipos de marcadores que reconoce AprilTag son los siguientes:

Tabla 2.24

Diferentes tipos de marcadores fiduciales de AprilTag

Tipos de marcadores fiduciales proporcionados por AprilTag
Tag36h11
TagStandard41h12
TagStandard52h13
TagCircle21h7
TagCircle49h12

Nota. Esta información se obtuvo de [45]

Para nuestro prototipo final se utilizó el Tag36h11 porque al ser cuadrado fue más fácil medir las distancias y es el más común de generar en internet, se definió una longitud del marcador de 16 cm y se configuró dentro del paquete 'apriltag_ros'.

2.13.3 Creación del archivo Python para guardar las posiciones de las estaciones de trabajo

Con la ayuda de un archivo Python se ubicó cada uno de los marcadores fiduciales en el mapa y se guardó las posiciones de las estaciones, también se utilizó estos marcadores para posicionar al robot cerca de la cada estación y lograr su tarea de servicio.

Capítulo 3

3 Resultados y análisis

Las configuraciones, código y el modelado *.stl* del proyecto se pueden encontrar en el siguiente repositorio: https://github.com/empaguer/Walter_AMR.git y una demo del funcionamiento del robot que se encuentra en el apéndice A.

3.1 Simulación

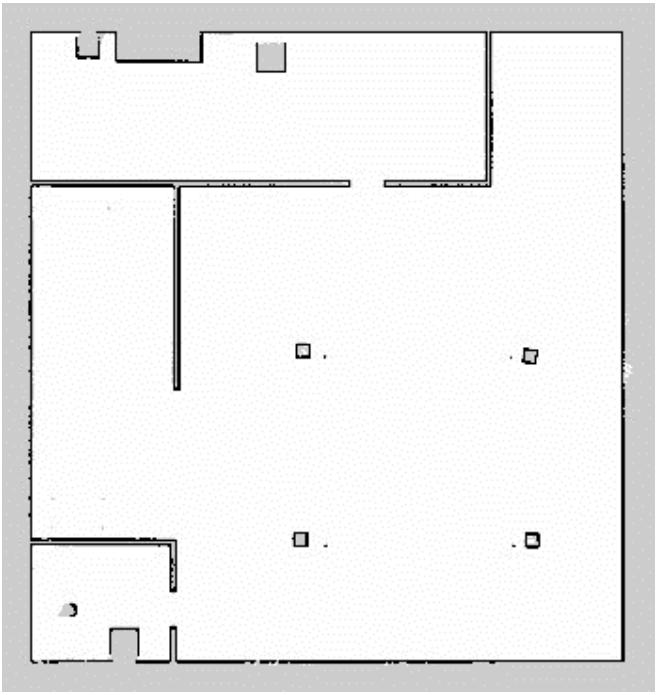
3.1.1 Mapeo en Gazebo

En la **Figura 3.1** se logra observar el mapeo realizado con las configuraciones de Cartographer, para este escenario de funcionamiento, en la implementación se usa el Joystick para efectuar el movimiento, pero en gazebo se utilizó operaciones con el *teleop_twist_keyboard.*

La calidad del mapa generado es bastante eficiente, el mapa se guarda como un archivo. bag.*pbstream* para después ser utilizado en navegación.

Figura 3.1

Mapa de cafetería generado en Gazebo.



3.1.2 Localización en Gazebo

El sistema de localización del robot en el escenario de mapeo es automático y Cartographer da gran precisión, sin embargo, en el escenario dos, para localizar el robot es necesario generar movimientos iniciales hasta que el robot se localice automáticamente.

Figura 3.2

Inicio de sistema de localización, el robot no se encuentra en su ubicación real

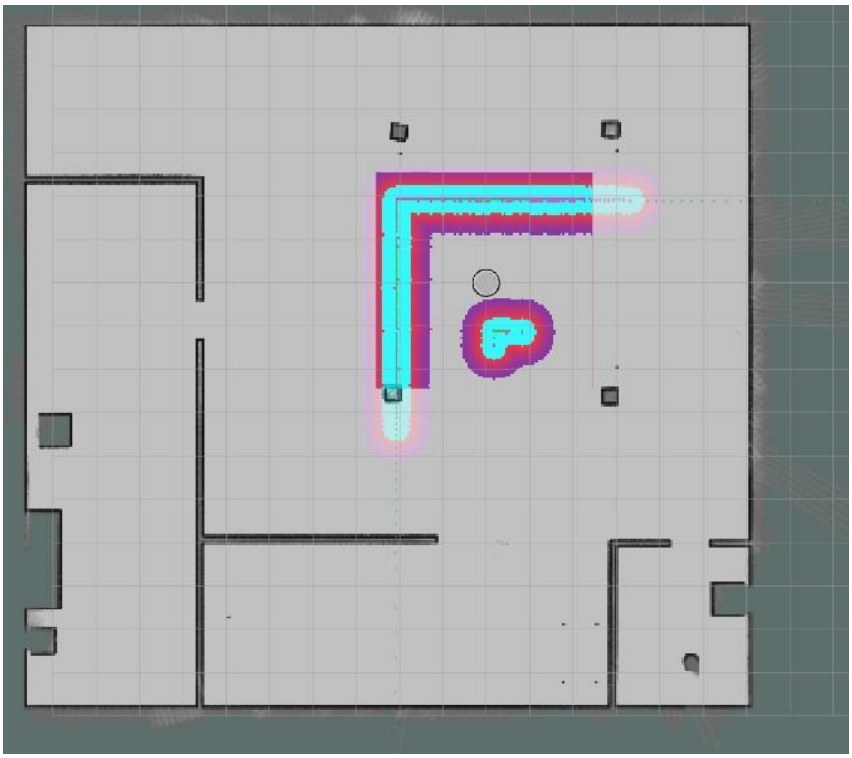
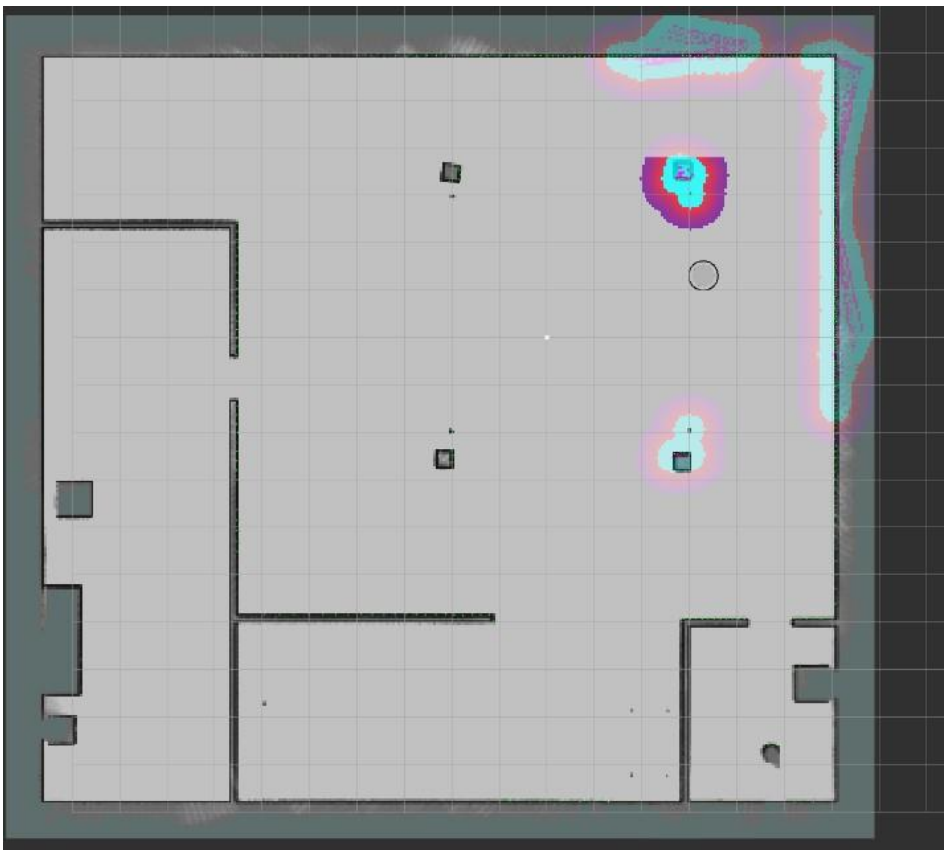


Figura 3.3

Robot Localizado, listo para generar operaciones de navegación.



3.1.3 Navegación autónoma en Gazebo

Apenas el robot se encuentra localizado correctamente, se habilita la navegación entre puntos, se utilizó el enfoque Dynamic Window Approach [46] para planificar el movimiento teniendo en cuenta obstáculos fijos presentes en el mapa global y obstáculos dinámicos presentes en el marco local.

Figura 3.4

Planeación del robot esquivando obstáculos fijos.

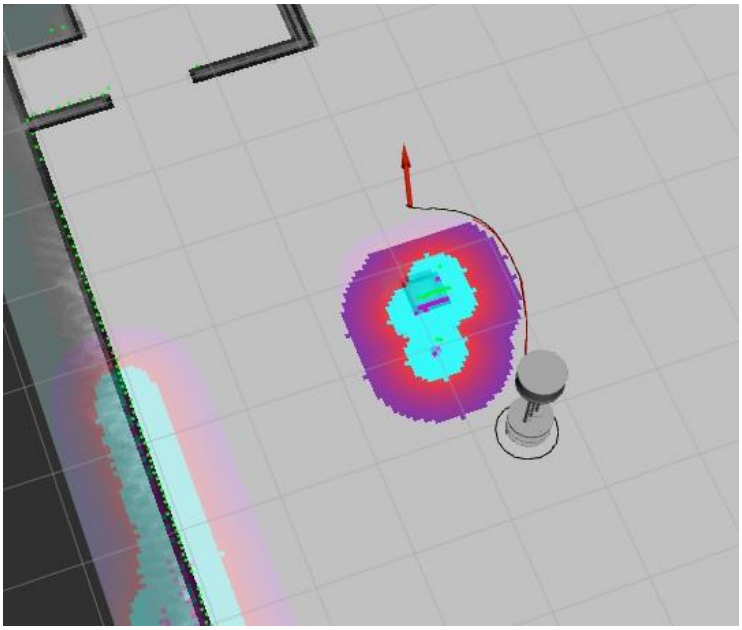


Figura 3.5

Robot esquivando obstáculos fijos en Gazebo.

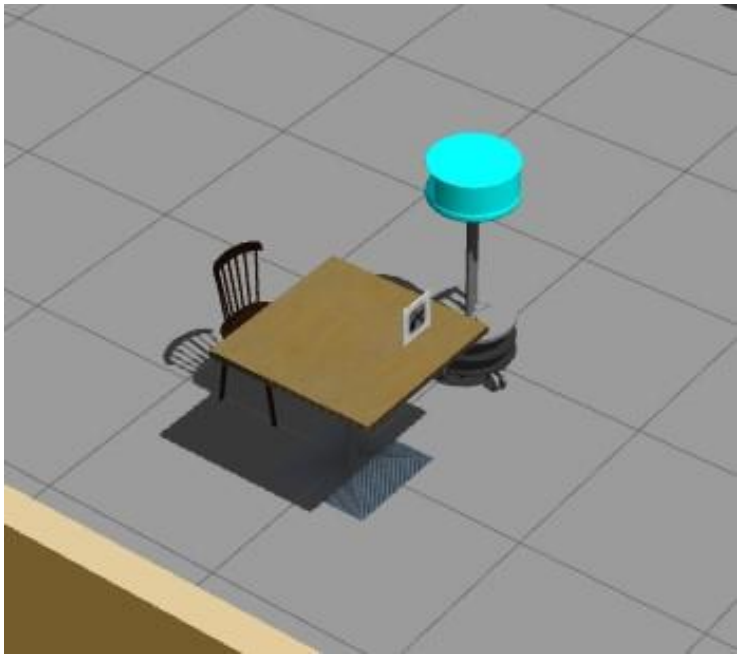
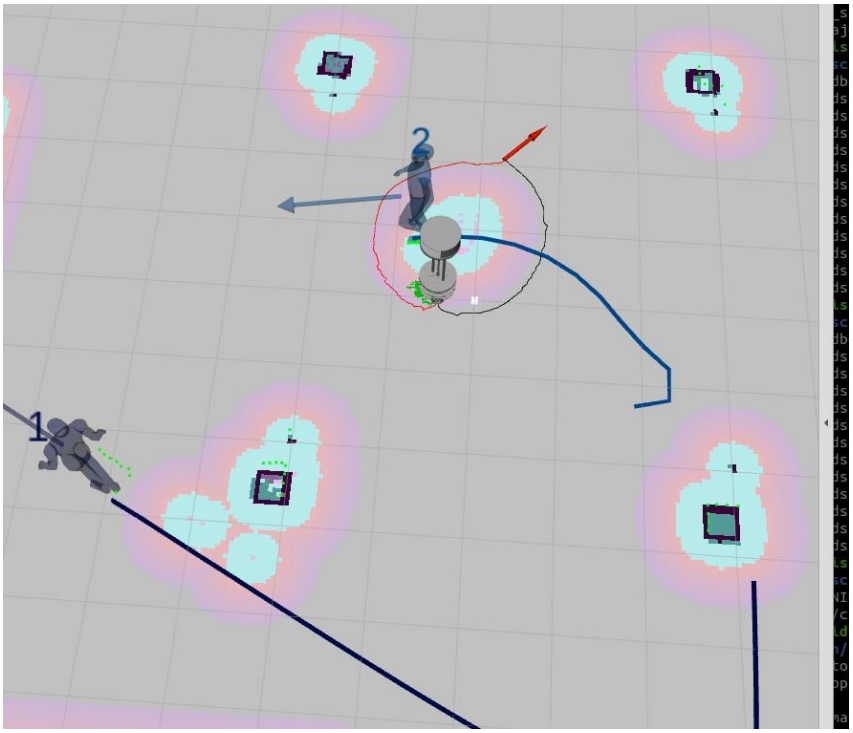


Figura 3.6

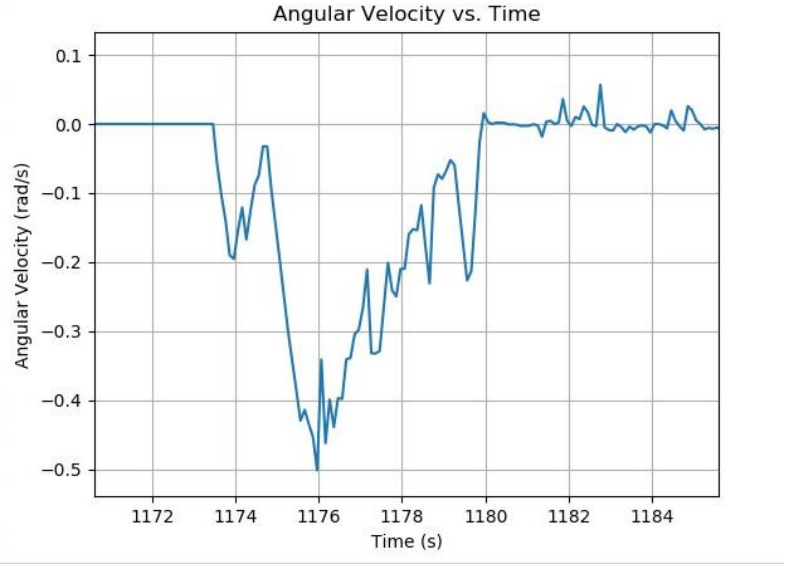
Cambio en la planeación del robot para evitar colisión con obstáculo dinámico.



El robot demostró la capacidad suficiente para evitar no solamente planeaciones con varios obstáculos globales si no también es capaz de evitar obstáculos dinámicos con un tiempo de reacción efectivo y sin movimientos bruscos.

Figura 3.7

Velocidad angular del robot en funcionamiento



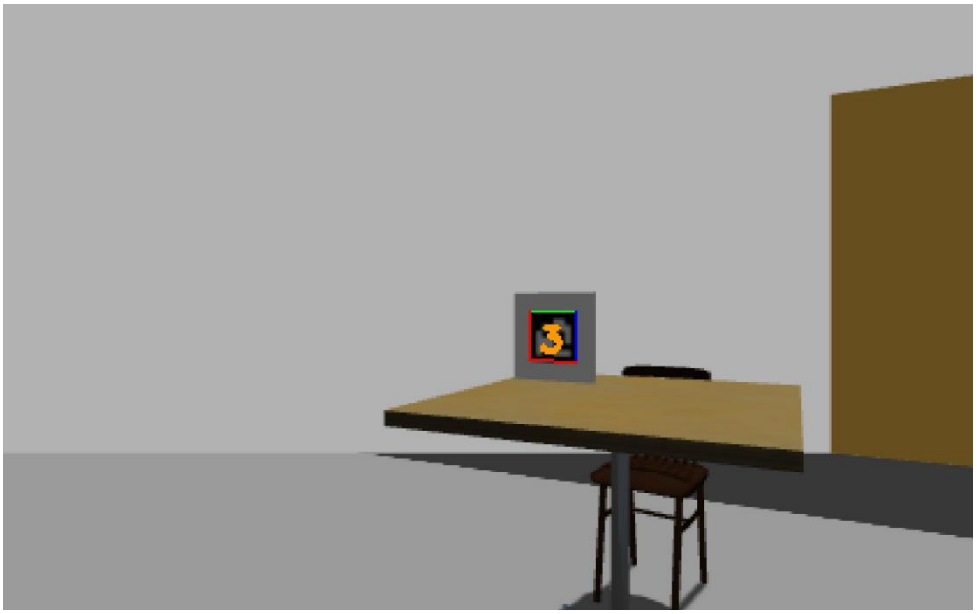
En la **Figura 3.7** se observó como el robot cambió su velocidad angular repentinamente en el segundo 1174 debido a que detectó un obstáculo dinámico representado en el mapa local y su objetivo principal es esquivarlo, en el segundo 1176 de la simulación el robot empezó a reacomodar su velocidad angular para mantener su planeación global debido a que el obstáculo se empezó a alejar.

3.1.4 Reconocimiento de AprilTag en Gazebo

Las distintas estaciones tienen su marcador fiducial único, los marcadores son reconocidos por el nodo apriltag_ros que envía mensajes directamente por ROS [47]

Figura 3.8

Reconocimiento del AprilTag en Gazebo



3.2 Análisis mecánico del cuerpo del robot

Para realizar el análisis mecánico del cuerpo del robot y la base de la cámara T265 primero se obtuvo los materiales con los cuales se diseñó estos componentes y están enlistados en la **Tabla 3.1**.

3.2.1 Materiales de las piezas

Tabla 3.1

Tabla de materiales de la base del robot y la base de la cámara T265

Elementos	Materiales
Base inferior	Plywood
Base media	Plywood
Base superior	Plywood
Contenedor de paquetes	PLA
Perno hexagonal M6x1x8mm	Acero ASTM A36
Perno hexagonal M6x1x15mm	Acero ASTM A36
Perno hexagonal M6x1x30mm	Acero ASTM A36
Tuerca de 6mm	Acero ASTM A36
Eje de cilindro lateral	Aluminio 6061
Eje de soporte medio	Aluminio 6061
Soporte de eje medio	Acero inoxidable AISI 304
Soporte de cilindro lateral	PLA
Tornillo 2mm	Acero ASTM A36
Base de cámara T265	PLA

Se investigó el límite de fluencia y resistencia a la tracción de cada material, estos materiales se encontraron directamente desde el programa de diseño Inventor de Autodesk y fueron los siguientes:

Figura 3.9

Características mecánicas del acero inoxidable AISI 304

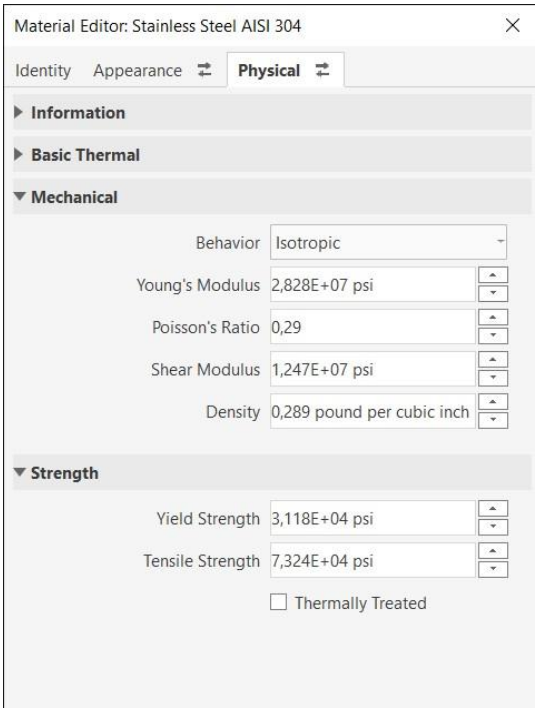


Figura 3.10

Características mecánicas del Acero ASTM A36

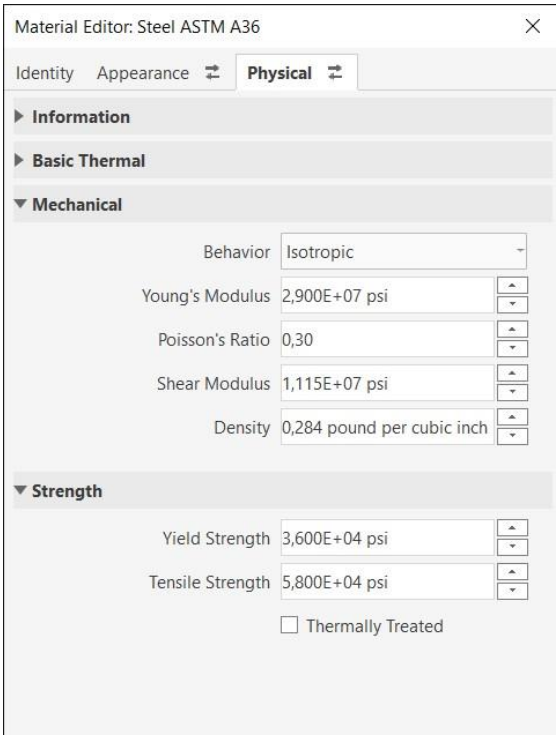
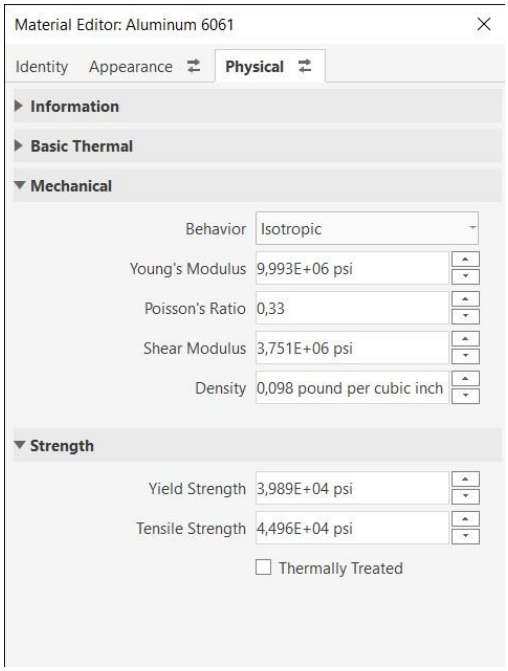


Figura 3.11

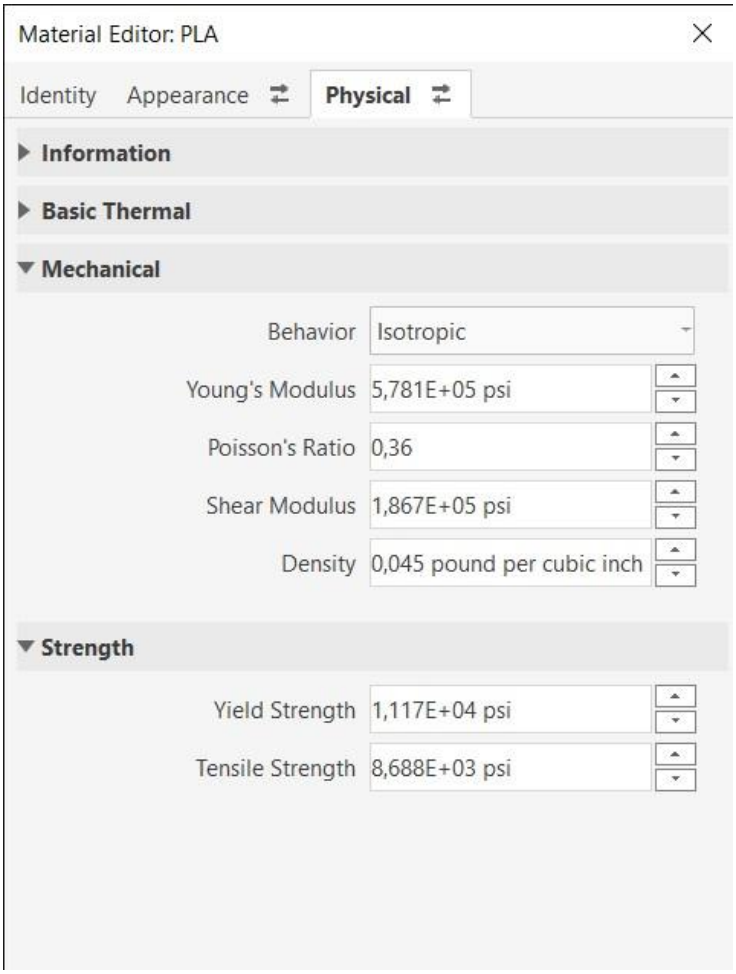
Características mecánicas del Aluminio 6061



Los siguientes materiales fueron agregados de manera manual al programa debido a que se modificaron para el análisis de elementos finitos y fueron los siguientes:

Figura 3.12

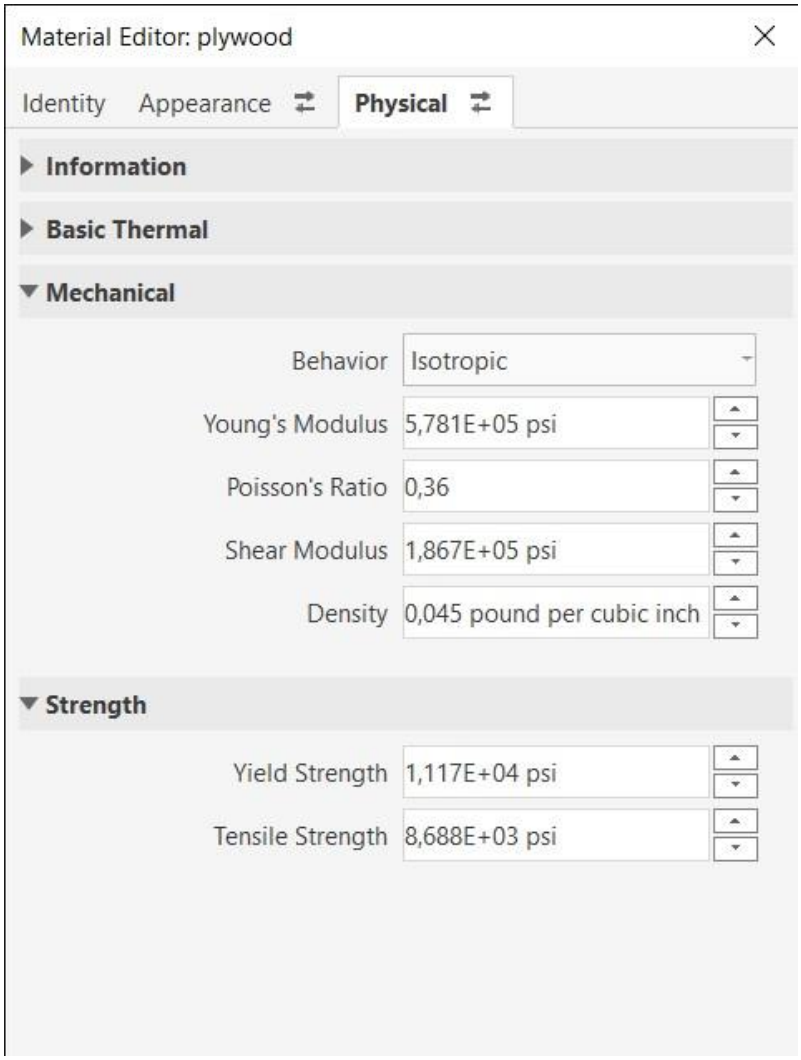
Características mecánicas del PLA



Nota. Esta información se obtuvo en el artículo [48] en el cual se obtuvo a base de pruebas

Figura 3.13

Características mecánicas del Plywood



Nota. Esta información se obtuvo de la siguiente página [49]

Después de establecer los materiales de casa pieza, se diseñó la construcción del cuerpo del robot y la base de la cámara T265, finalmente se obtuvo el análisis de elementos finitos y se dividió en distintos pasos.

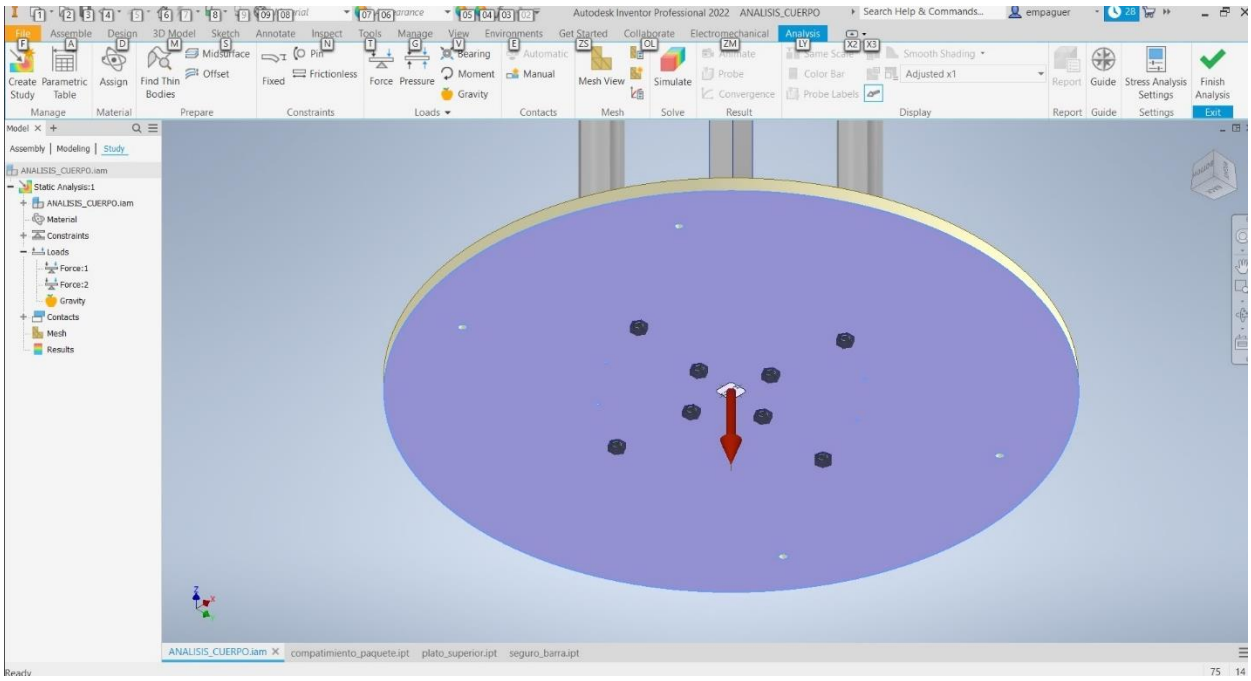
3.2.2 Análisis de elementos finitos

3.2.2.1 Asignación de fuerzas y momentos

Se establecieron dos escenarios en el cual el robot ejerce la mayor fuerza sobre el punto más alto del cuerpo del robot, el primer escenario fue cuando el robot se movilizó en línea recta con una aceleración de 0.9 m/s^2, el segundo escenario fue cuando el robot gira sobre su propio eje con una aceleración de 0.6 rad/s^2, en ambos escenarios se consideró la gravedad y se obtuvo unos resultados más cercanos a la realidad.

Figura 3.14

Gravedad en nuestro diseño

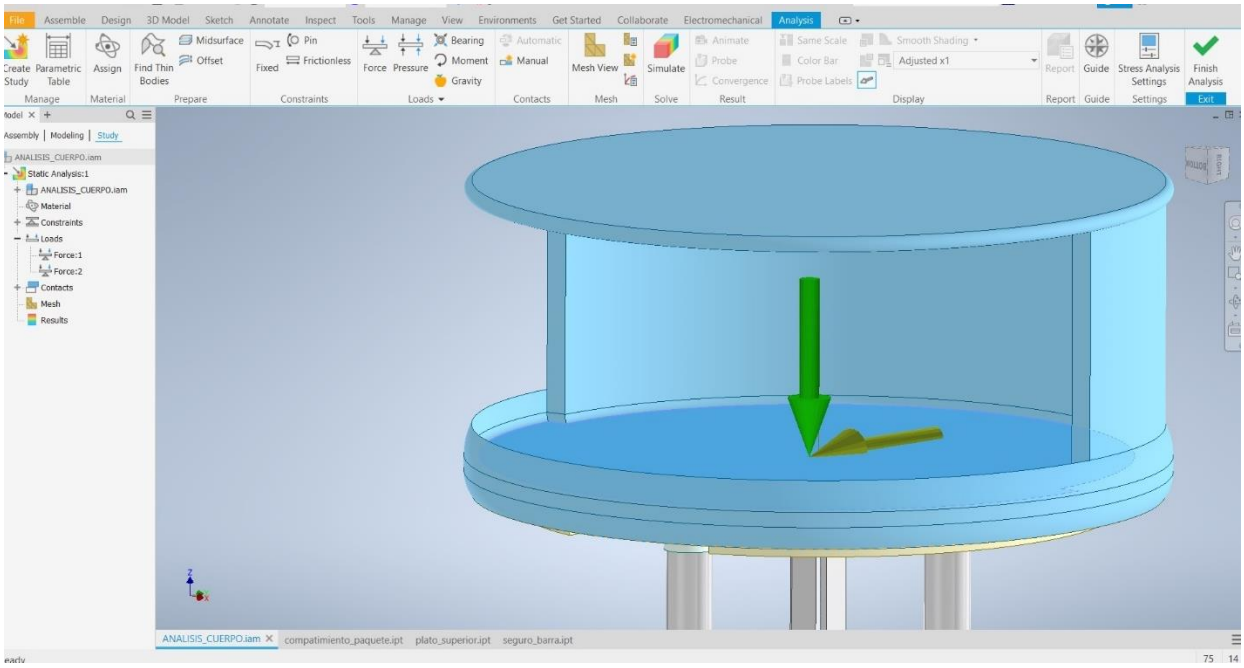


Nota. Se utilizó una gravedad de 9.8 m/s^2, esto ayudó a incorporar el peso al análisis

Para el primer escenario se asignó una fuerza hacia abajo en el eje Z negativo equivalente a la fuerza generada por tres platos de 550 g, la fuerza fue igual a 16.17 N, la fuerza a lo largo del eje Y negativo fue producida por la misma masa, pero utilizando la aceleración horizontal produciendo una fuerza de 1.48 N como se observa en la **Figura 3.15**.

Figura 3.15

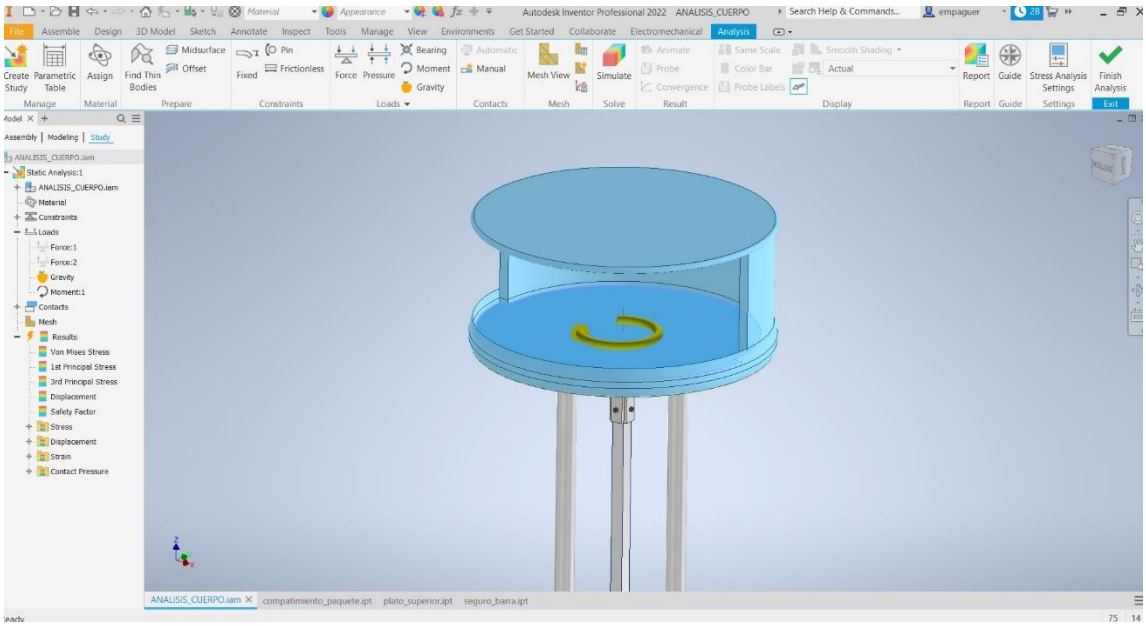
Fuerzas que actúan en la parte superior del cuerpo del robot



Para el segundo escenario se consideró una inercia de un cilindro con radio de 20 cm y masa de 1.65 kg en la parte superior del cuerpo alrededor del eje Z del robot igual a 0.033 kg*m^2 y la aceleración angular de 0.6 rad/s^2.Finalmente, se obtuvo un momento de 0.0198 Nm que se puede observar en **Figura 3.16**.

Figura 3.16

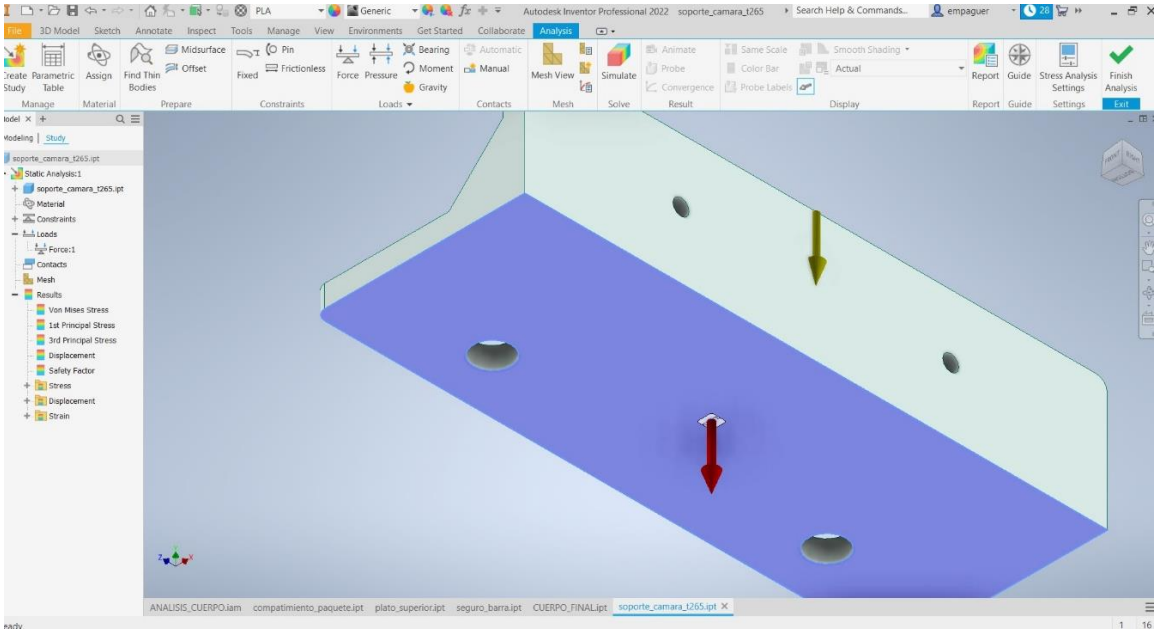
Momento producido en el segundo escenario sobre el cuerpo del robot



Para las fuerzas de la base de la cámara T265 se consideró la misma gravedad de 9.8 m/s^2 y la fuerza producida por la masa de la cámara de 60 g.

Figura 3.17

Fuerzas que interactúan con la base de la cámara T265

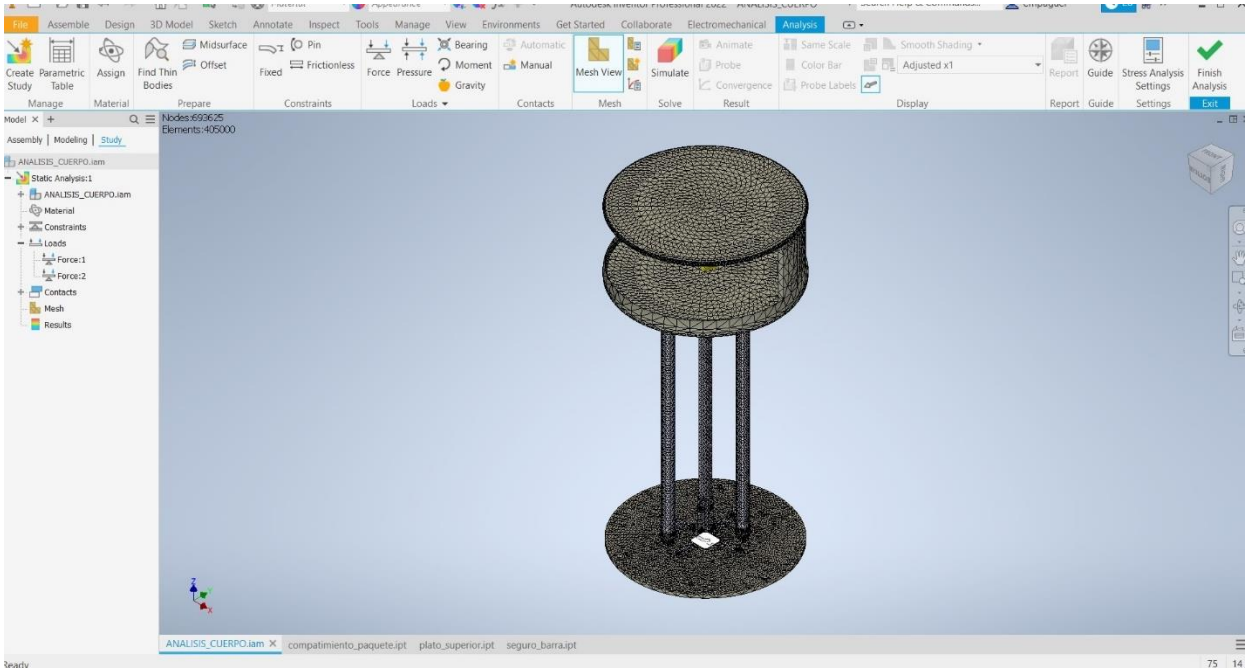


3.2.2.2 Generación de la malla para el análisis

Se generó una malla con un ángulo máximo de giro de 30° para el análisis.

Figura 3.18

Malla del análisis de elementos finitos

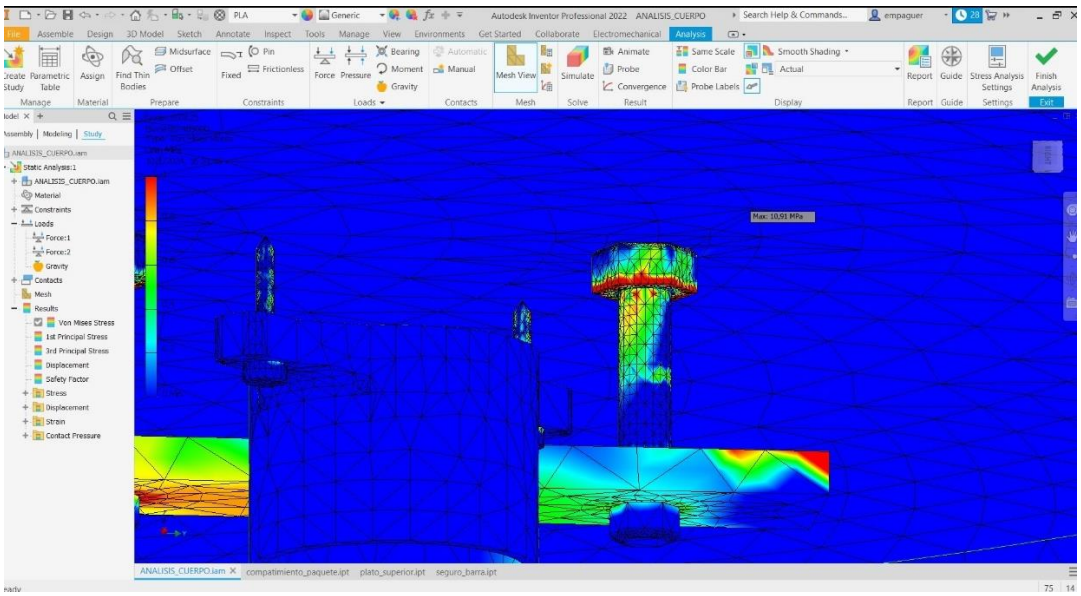


3.2.2.3 Simulación y análisis de elementos finitos

Como se puede observar en la **Figura 3.19** se obtuvo un esfuerzo máximo de Von mises de 10.91MPa principalmente en los pernos, el límite de fluencia del Acero ASTM A36 es de 248.21 MPa. Por lo tanto, la estructura soporta perfectamente los esfuerzos y tiene un factor de seguridad de 15.

Figura 3.19

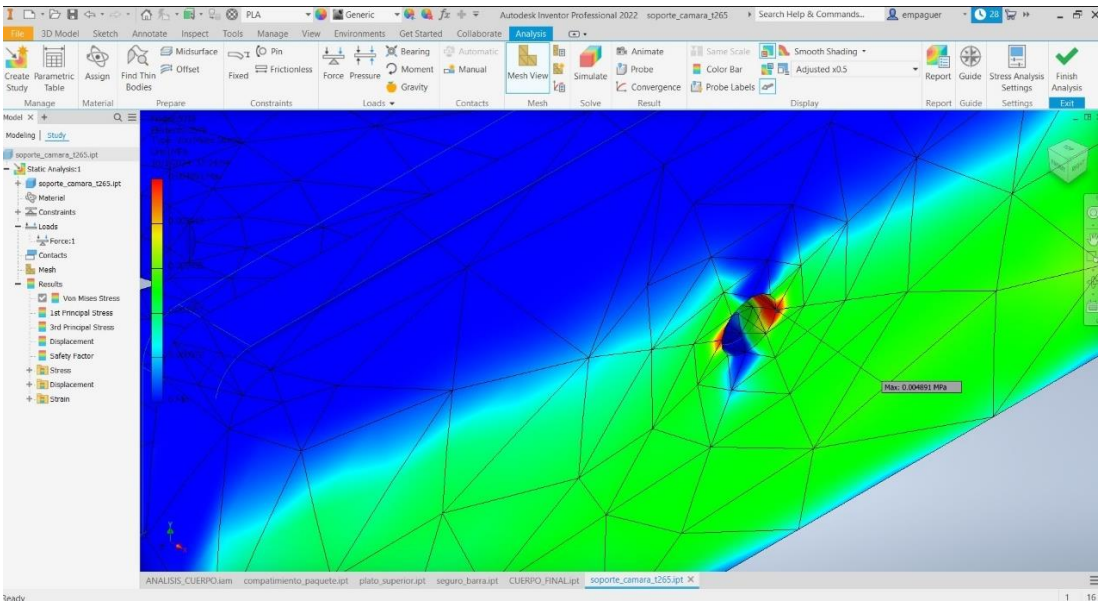
Análisis de elementos finitos del cuerpo del robot



Luego se analizó la base de la cámara T265 que se puede observar en la **Figura 3.20** se obtuvo un esfuerzo máximo de Von mises de 0.004891MPa principalmente en los orificios de sujeción, el límite de fluencia del PLA es de 77 MPa así confirmando que la base soporta perfectamente y tiene un factor de seguridad de 15.

Figura 3.20

Análisis de elementos finitos de la base de la cámara

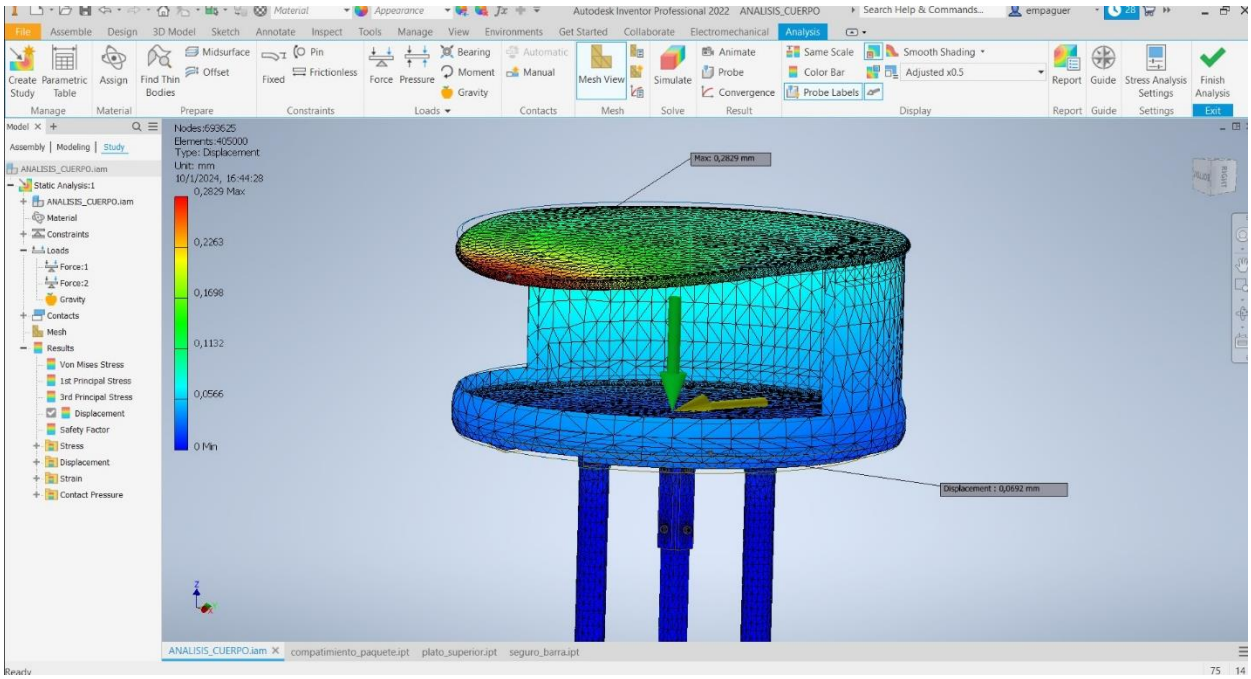


3.2.3 Desplazamiento de los elementos

Para el cuerpo del robot se obtuvo un desplazamiento total de 0.2918 mm con lo cual se concluye que el movimiento horizontal y rotacional no ocasionó que el paquete transportado se caiga, se observa en la **Figura 3.21**.

Figura 3.21

Desplazamiento del cuerpo del robot

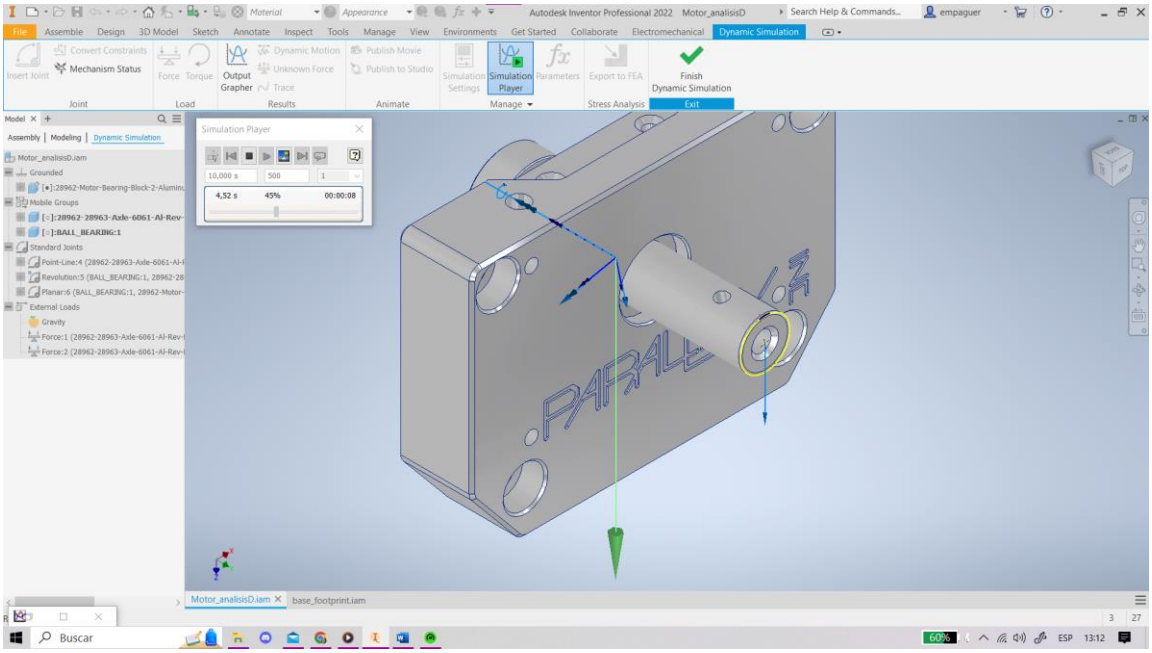


3.2.4 Análisis de torque necesario para el movimiento

Para el análisis de torque de analizó el eje del motor y se utilizó la herramienta de análisis dinámico de inventor para este propósito, se obtuvo dos cargas que actuaron sobre el eje del motor, una igual a 25.48N debido a la rueda y otra de 100.45 N, estas se distribuyen equitativamente para cada motor, se puede observar en la **Figura 3.22** y en el Apéndice 0.

Figura 3.22

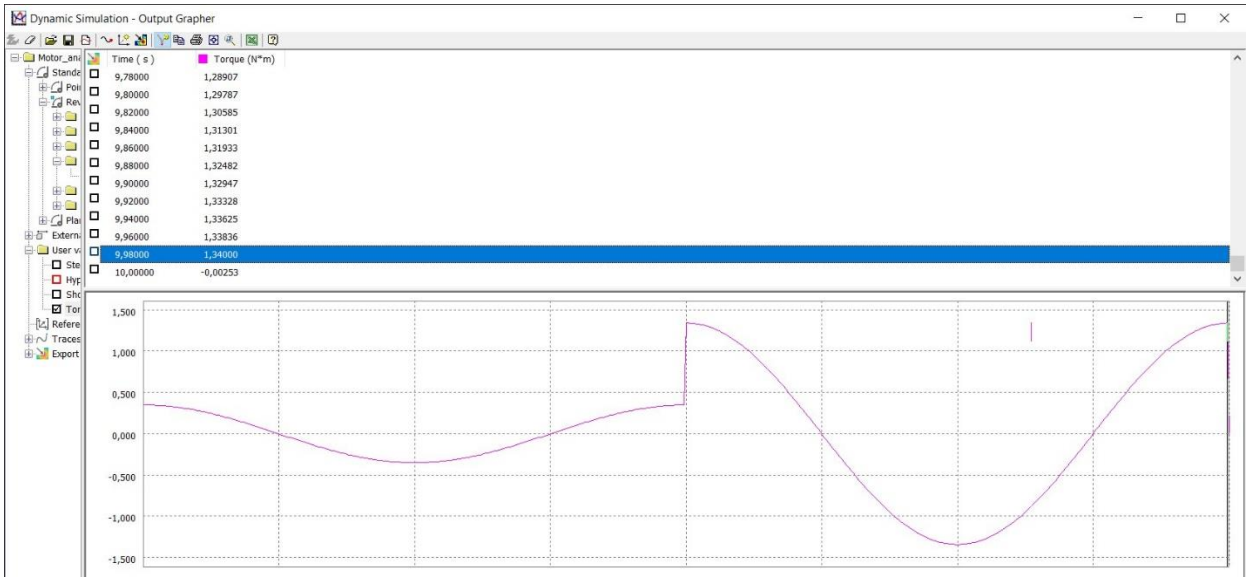
Distribución de cargas en el eje del motor en Inventor



Finalmente, se obtuvo una gráfica de torque con un valor máximo igual a 1.34 N*m utilizando velocidades angulares críticas del robot que se observan en el apéndice E, estos valores son muy cercanos a la simulación en Gazebo de la sección 3.1 y se confirma que el robot no supera el límite de 27.22 kg de carga y 9.60 N*m de torque máximo. Por lo tanto, estos límites de velocidad fueron los óptimos para el buen funcionamiento del robot.

Figura 3.23

Gráfica de Torque para uno de los motores



3.3 Controlador para los motores DC

El controlador se basa directamente en el modelo matemático del robot para el envío de la señal y la calibración (tunning) de las ganancias del PID, inicialmente se pensó hacer un controlador que compare los valores de velocidad de entradas y salidas para sintonizar el PID, sin embargo la existencia de un retraso entre la velocidad actual y la velocidad leída ocasionaba que el cálculo de error no sea confiable, por ende se optó por implementar un controlador mediante el firmware del DHB-10, que proporcionó un código interno en el cual implementa un controlador PI, nuestro controlador se modificó debido a esta limitación y se puede observar su estructura final en la **Figura 3.24**.

Figura 3.24

Esquema de la estructura del controlador del DHB-10

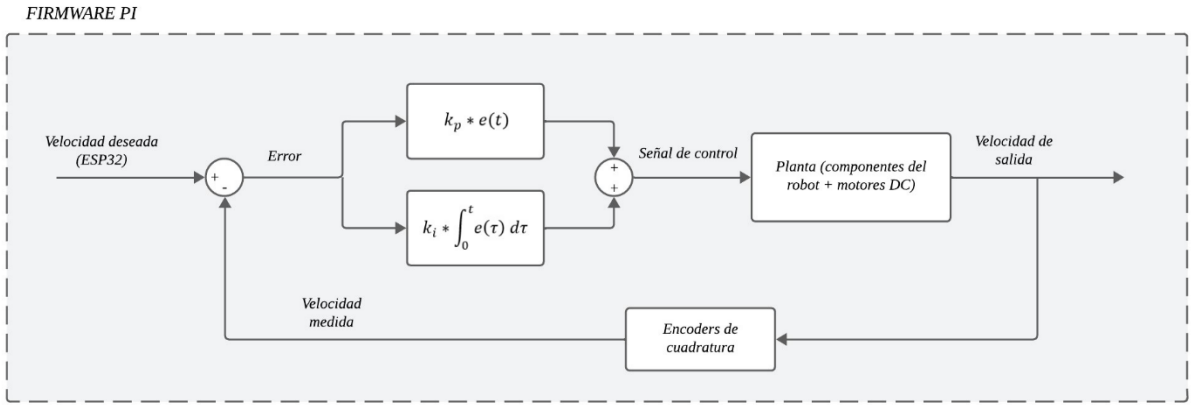
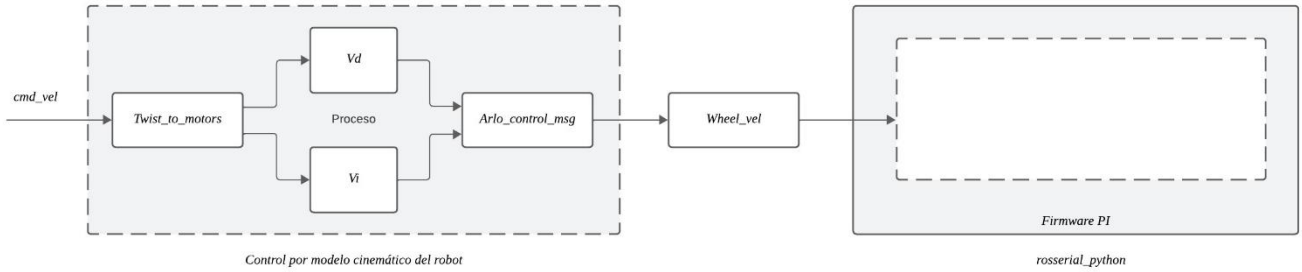


Figura 3.25

Esquema final del control por modelo cinemático

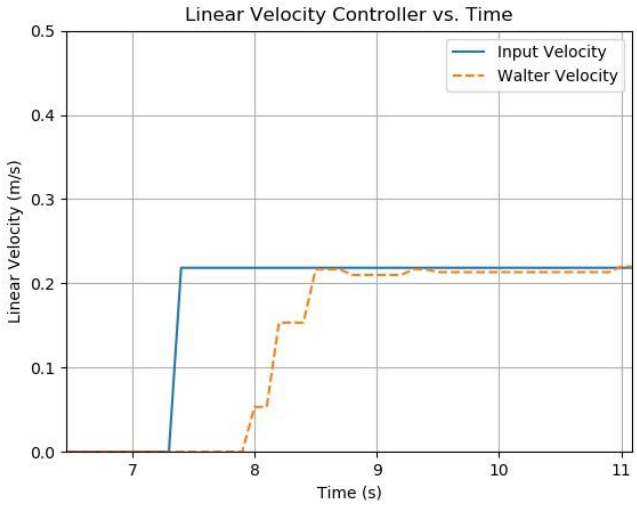


Nota. El firmware PID es conformado por el modelo PID ya explicado en el capítulo 2 sección 2.9

En las siguientes secciones se obtuvo los valores de K_p y K_i con el método heurístico y probándolos en el robot real.

Figura 3.26

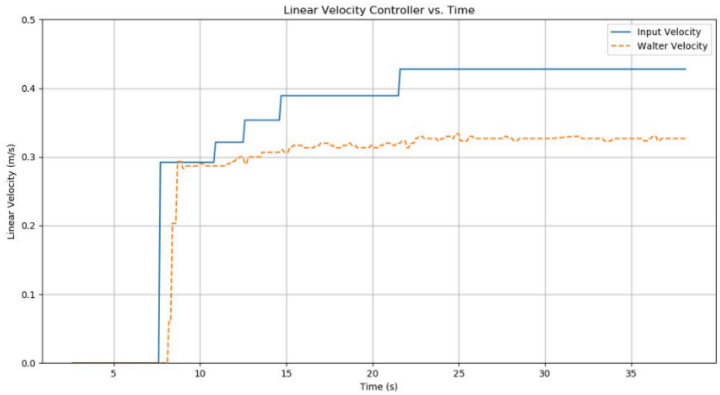
Controlador de velocidad utilizando solo la parte proporcional (Kp)



En la **Figura 3.26** se utilizó un K_P de 100 para obtener la gráfica, en esta gráfica se observó que tiene un tiempo de subida lento, pero logra llegar al punto deseado de velocidad.

Figura 3.27

Gráfica de la velocidad utilizando un controlador proporcional



Nota. En esta gráfica se varió la velocidad para observar el comportamiento del controlador.

En la **Figura 3.27** se observó que el controlador usando solo la ganancia proporcional no manejó los cambios bruscos de velocidad ni logró obtener una velocidad mayor a 0.33 m/s, esto ocasionó un error a medida que se fue variando la velocidad del robot, para esto se agregó la parte integral para eliminar el error en estado estable.

Figura 3.28

Gráfica del controlador PI de velocidad

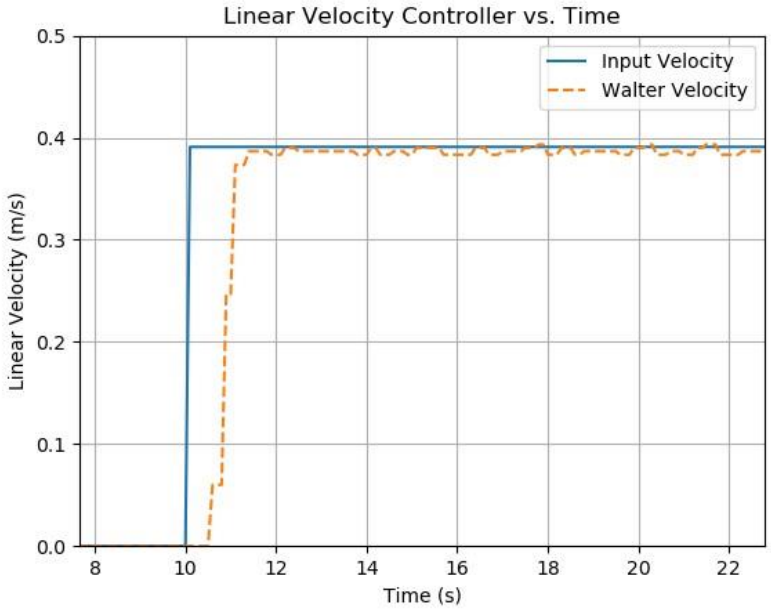
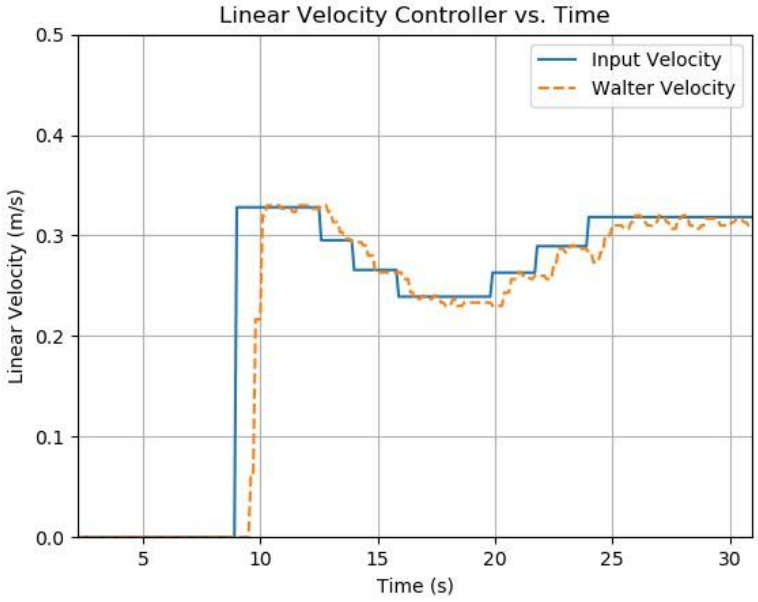


Figura 3.29

Gráfica del controlador PI de velocidad reaccionando a cambios



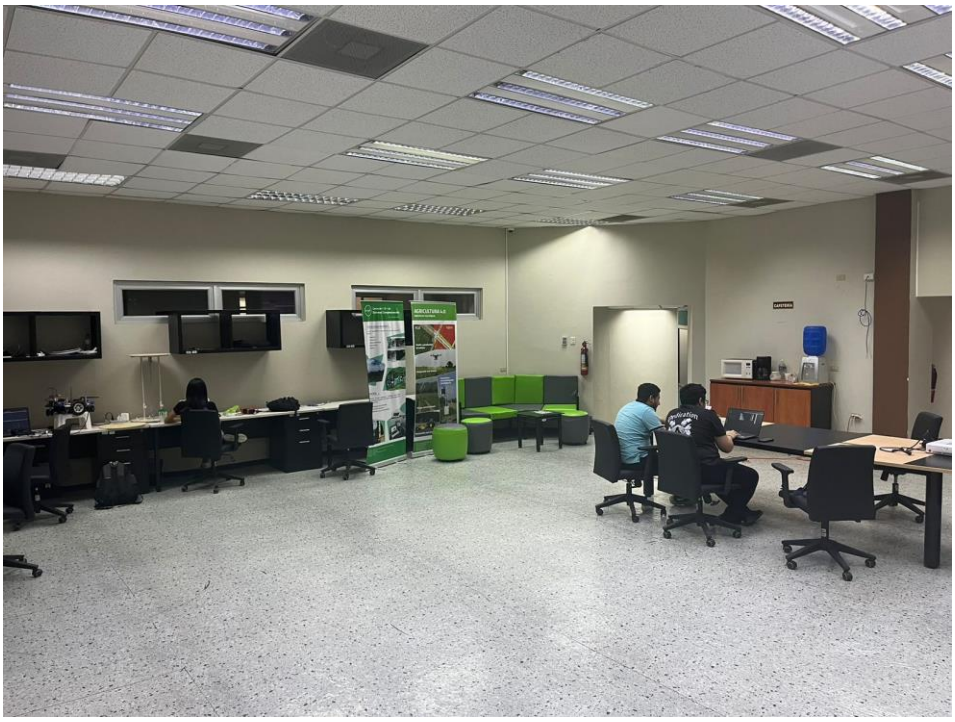
Como se observa en la **Figura 3.28** después de sintonizar nuestra ganancia K_i a un valor de 80 se obtuvo una respuesta óptima de trabajo para la velocidad de nuestro robot, en la **Figura 3.29** se confirmó que nuestra ganancia integral eliminó el error de estado estable y proporcionó un control PI final deseable para nuestro prototipo final con un tiempo muerto de dos segundos.

3.4 Implementación del funcionamiento final del robot

3.4.1 Mapeo de espacio desconocido

Figura 3.30

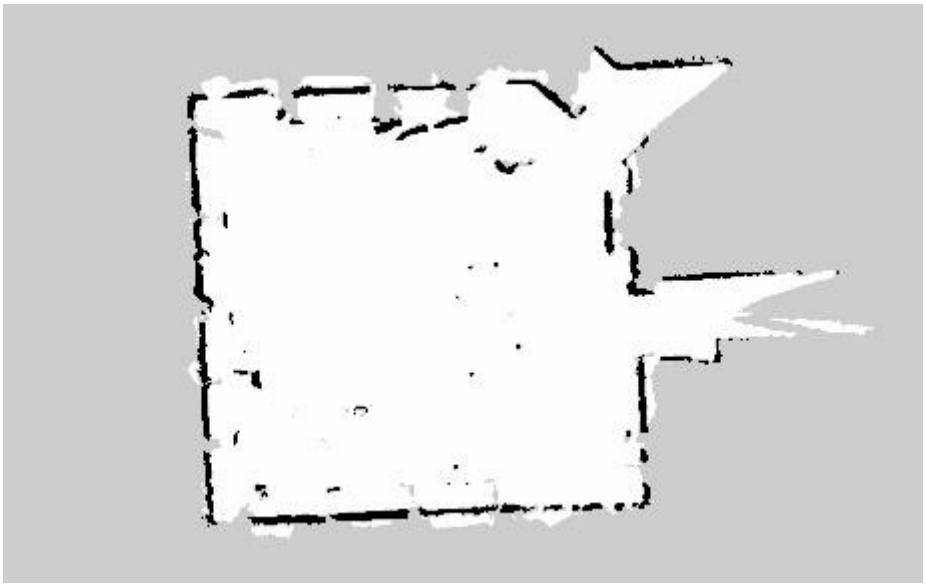
Espacio de CIDIS



La **Figura 3.30** representa el objetivo de mapeo del robot y es donde se realizaron las pruebas de mapeo del robot.

Figura 3.31

Mapa generado con Cartographer



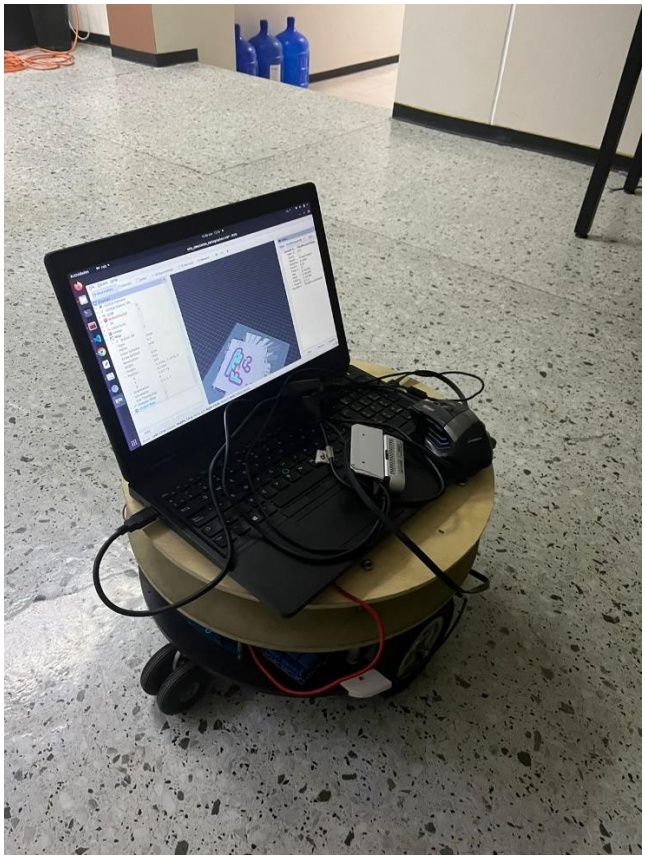
Este mapa se generó con cartographer y se obtuvo correctamente la delimitación de las paredes de la **Figura 3.30**, así como también los puntos en el mapa que representan mesas y sillas, este mapa es estrictamente 2D por lo que el usuario debe asegurarse de que no existan desniveles o escaleras.

La arquitectura de nodos utilizada para el modo de mapeo se puede observar en el apéndice D.

3.4.2 Navegación autónoma del robot

Figura 3.32

Robot navegando en mapa preestablecido



En la **Figura 3.32** el robot navegó a través del mapa escaneado en la sección anterior y con ayuda de las técnicas de SLAM de Amcl y Cartographer, durante la trayectoria existió ruido en el movimiento angular del robot, sin embargo, logró recorrer su ruta de trabajo.

3.4.3 Asignación de estaciones y orientación según marcadores fiduciales

Figura 3.33

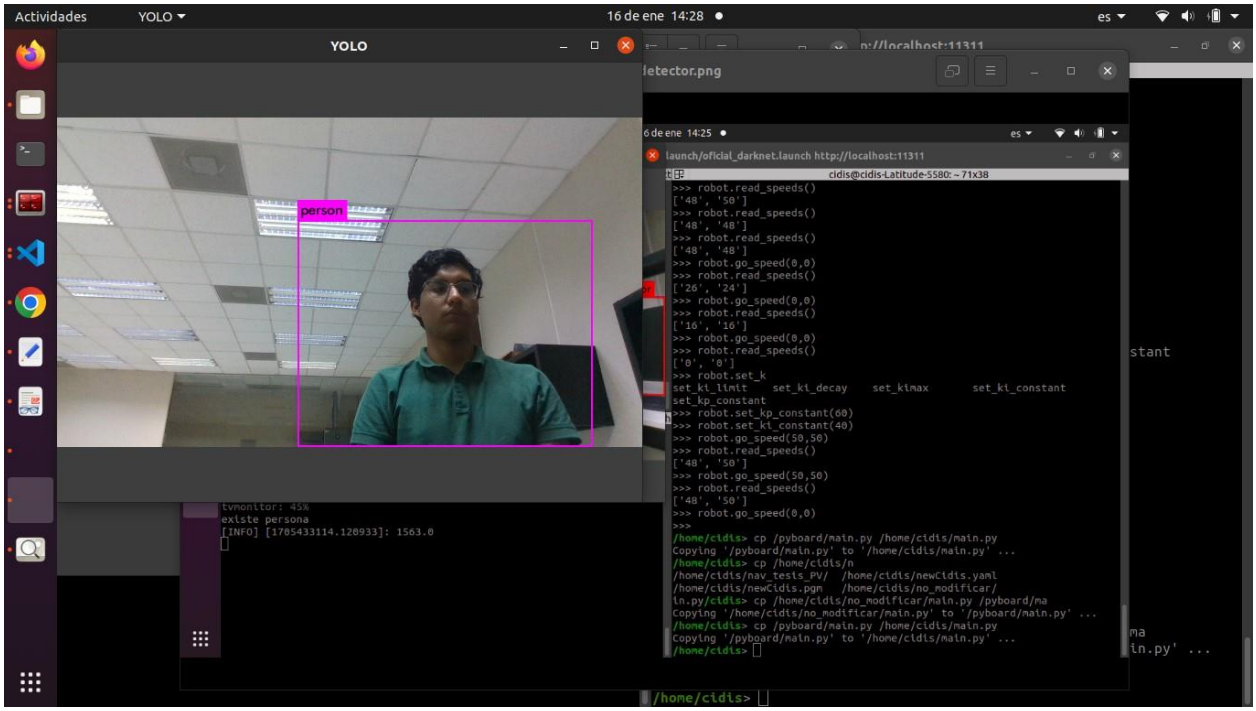
AprilTag asignado a una estación

En la **Figura 3.33** se observa el AprilTag asignado a la estación uno, donde está la base de madera es donde llegará el robot para buscar la orientación correcta.

3.4.4 Reconocimiento de personas utilizando Darknet ROS

Figura 3.34

Reconocimiento de personas utilizando Darknet ROS



Como se observa en la **Figura 3.34** se incorporó la detección de personas utilizando la cámara d435i y el paquete de llamado Darknet ROS, el cual utilizó Yolo V2 para la detección.

Figura 3.35

Detección de diferentes agentes de entrenamiento con Yolo V2

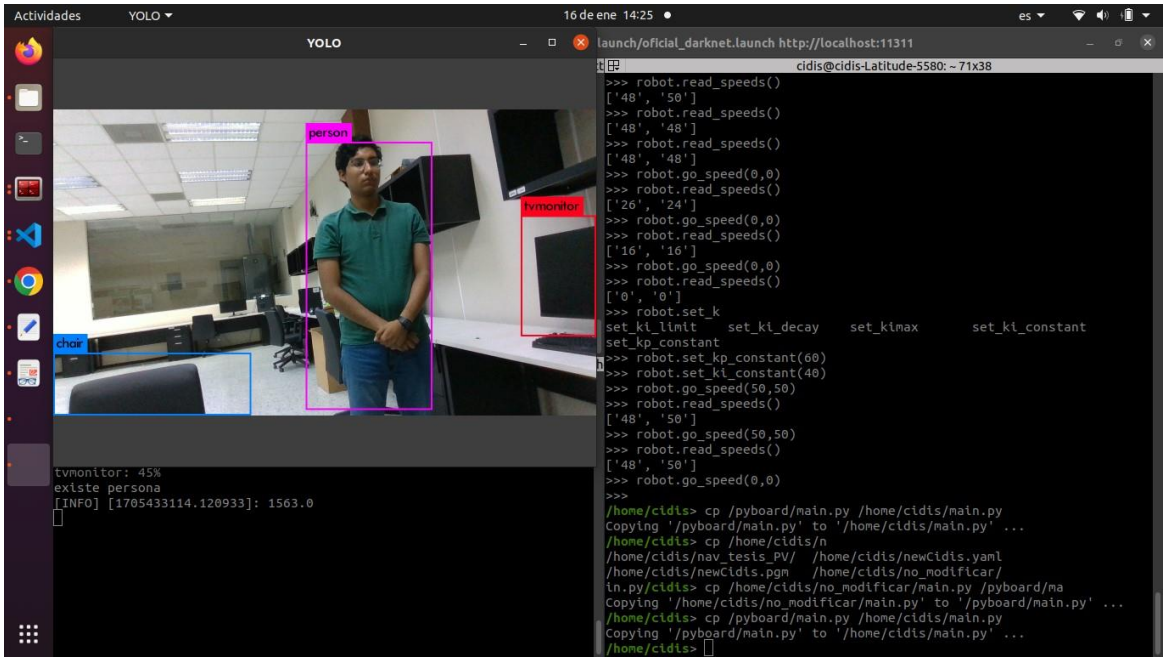
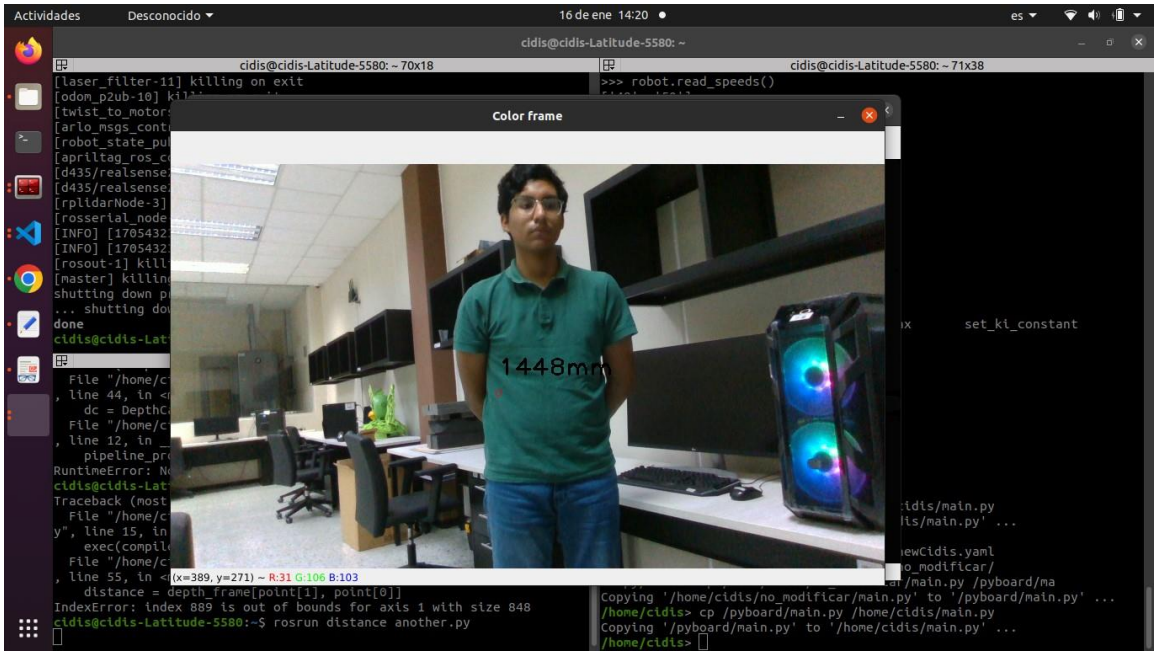


Figura 3.36

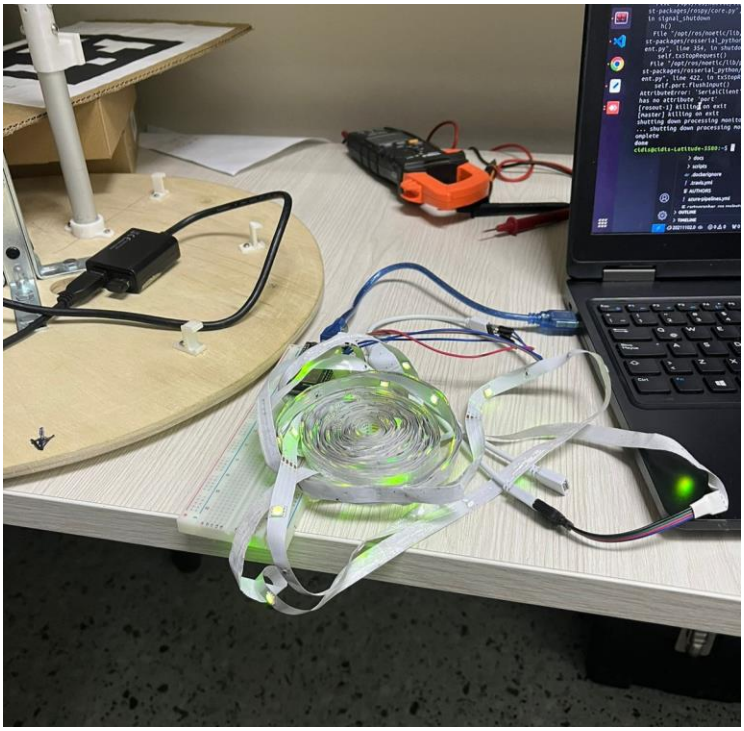
Cálculo de la distancia de una persona con respecto a la cámara



Se calculó la distancia estimada de la persona y se usó esta información en el escenario dos para interactuar con el usuario final utilizando luces LED.

Figura 3.37

Luces LED funcionando con Darknet ROS

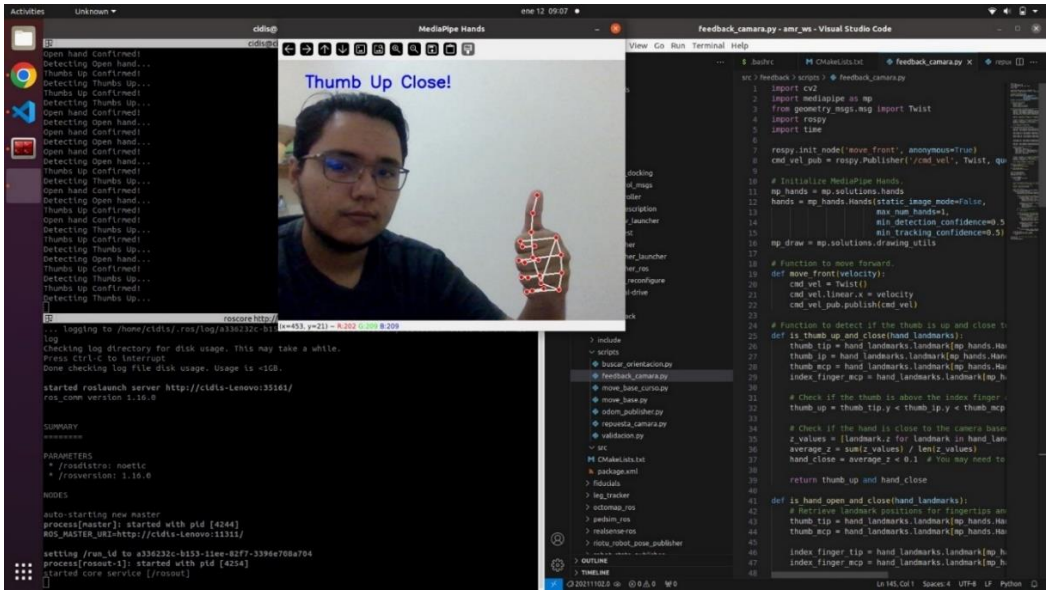


Este nodo de reconocimiento no se utilizó en el producto final debido a que el reconocimiento de personas requería mucho poder computacional y ocasionaba que los mensajes importantes de otros nodos no se transmitieran a una velocidad correcta.

3.4.5 Interacción Humano-Robot mediante gestos

Figura 3.38

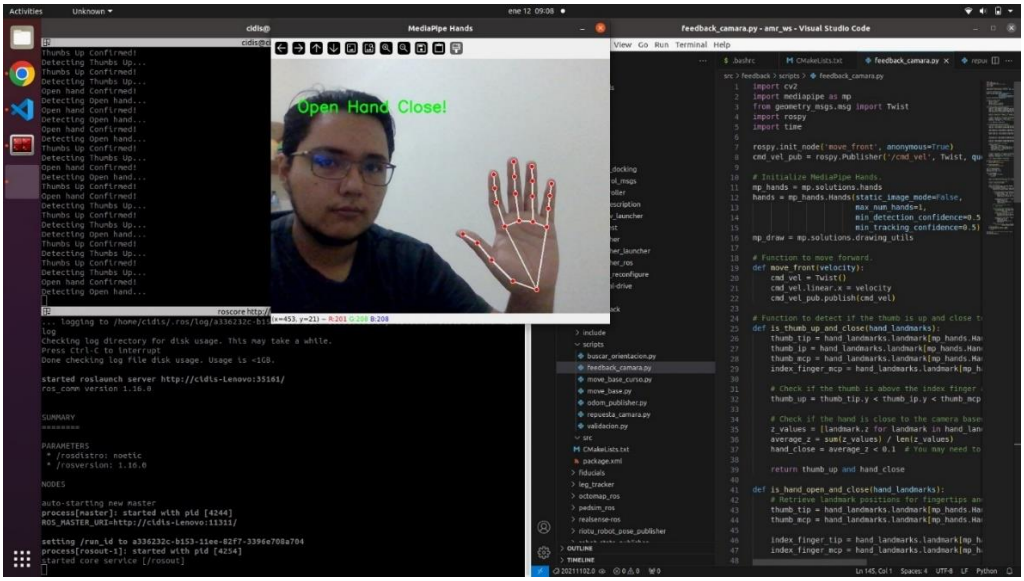
Reconocimiento de gesto de mano con pulgar arriba



Nota. Se puede observar a la izquierda que el usuario debe mantener el gesto cierto tiempo para que el robot acepte la orden

Figura 3.39

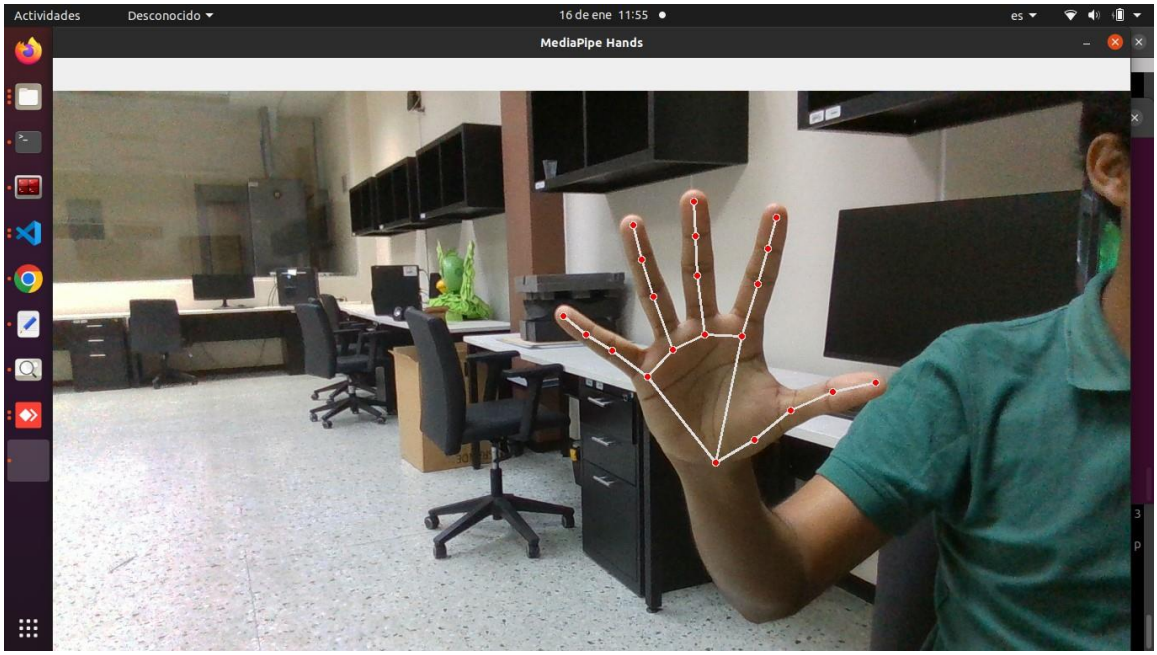
Reconocimiento de gesto de mano abierta



Nota. Se puede observar a la izquierda que el código detectará que tipo de gesto se realiza y luego dará una orden al robot

Figura 3.40

Detección de mano abierta sin realizar alguna acción posterior



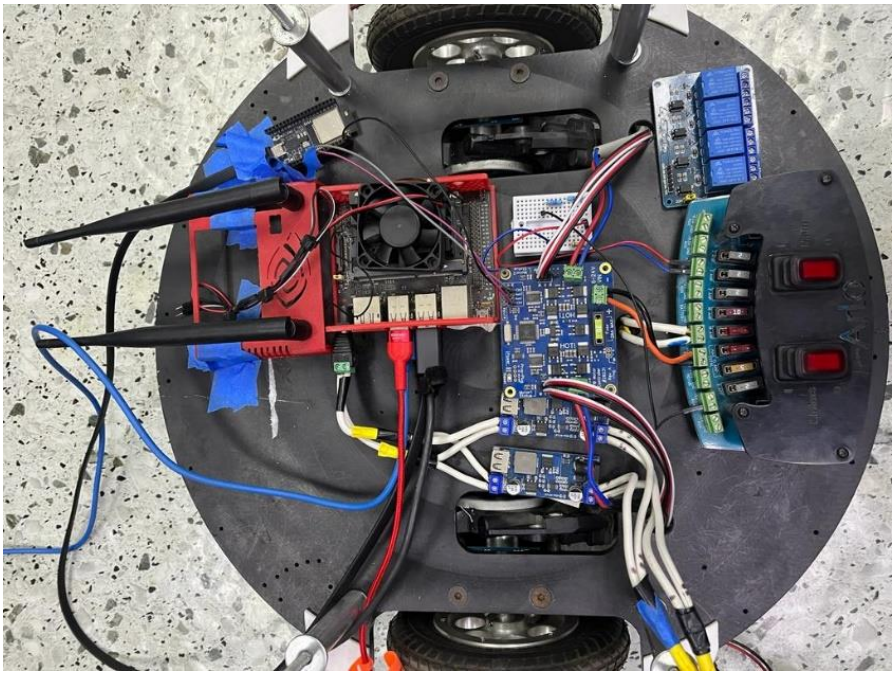
3.5 Funcionamiento del robot en escenarios reales

3.5.1 Construcción del robot

Con los componentes a utilizar definidos, se procedió a construir el robot físicamente, empezando por el primer piso, se realizó las conexiones electrónicas principales de los controladores y componentes PARALLAX.

Figura 3.41

Vista Superior de conexiones eléctricas



En el segundo piso, se ubicó la cámara de Odometría IntelRealsense T265 con su sistema de referencia apuntando a la misma orientación que el Lidar A1 ubicado en el centro del robot.

Figura 3.42

Montura de cámara T265 y Lidar en segundo piso de la plataforma robótica

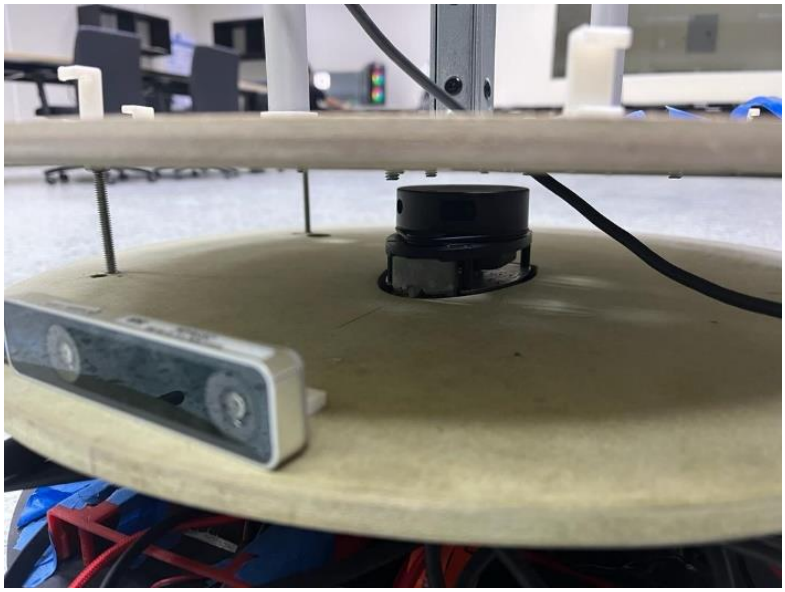
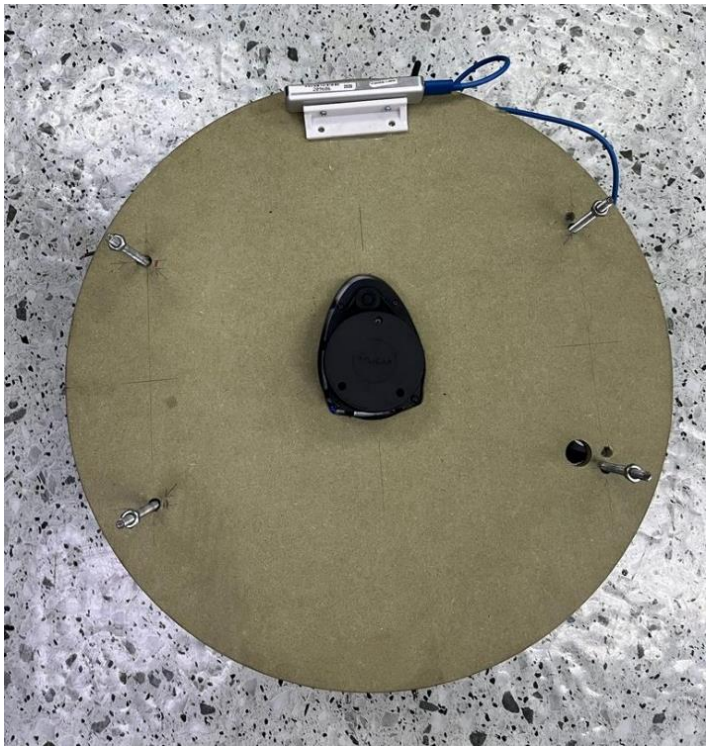


Figura 3.43

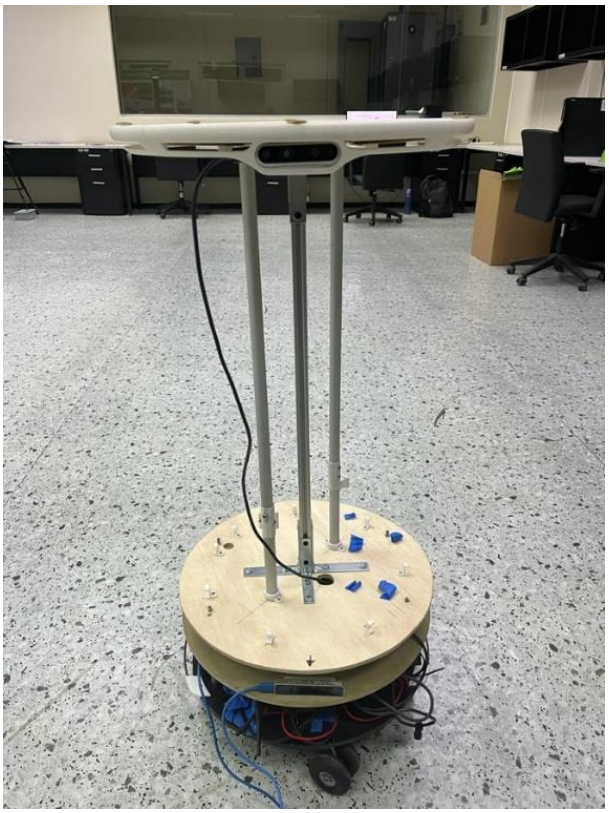
Vista superior del segundo piso de la plataforma



Finalmente se instaló la cámara D435i en el cuerpo superior del Robot en conjunto con su soporte, de igual manera, esta cámara comparte la orientación del Lidar A1 y la cámara T265, todos los componentes se conectaron mediante USB hacia la Jetson Nano.

Figura 3.44

Montura de cámara D435 y construcción interna del robot.



3.5.2 Funcionalidad completa del robot

Para la funcionalidad final del robot, en las siguientes figuras se mostró una rutina del robot en un escenario real, el robot puede ser accionado de manera local o utilizando la aplicación móvil como controlador, esta aplicación móvil fue desarrollada por Joby Farra y Gabriela Ramos como parte de materia integradora de la carrera de computación, las capturas de la aplicación se encuentran en el apéndice B.

Figura 3.45

Robot en la estación base esperando orden

Figura 3.46

Orden recibida y moviéndose a estación uno

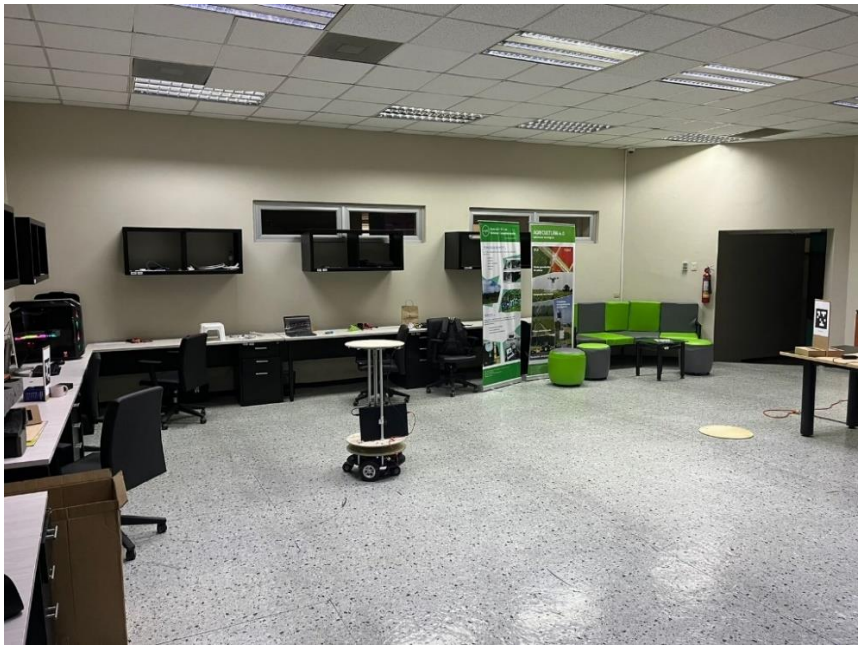


Figura 3.47

Robot llegando a estación uno esperando calibración con AprilTag

Figura 3.48

Robot calibrado con el AprilTag

Figura 3.49

Calibración obtenida vista desde la cámara D435

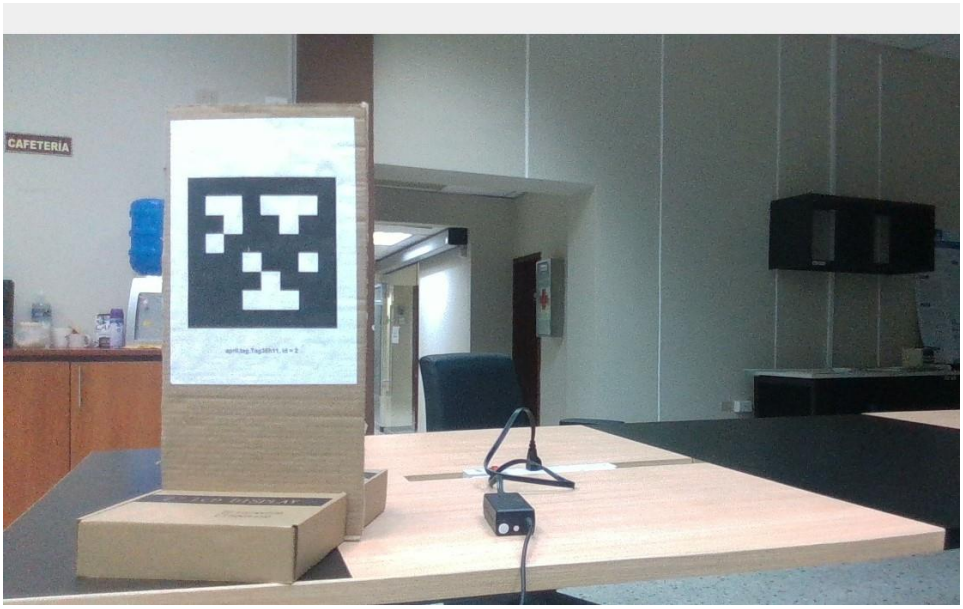


Figura 3.50

Robot recibiendo feedback con gestos para regresar a estación base

Figura 3.51

Robot volviendo a estación base

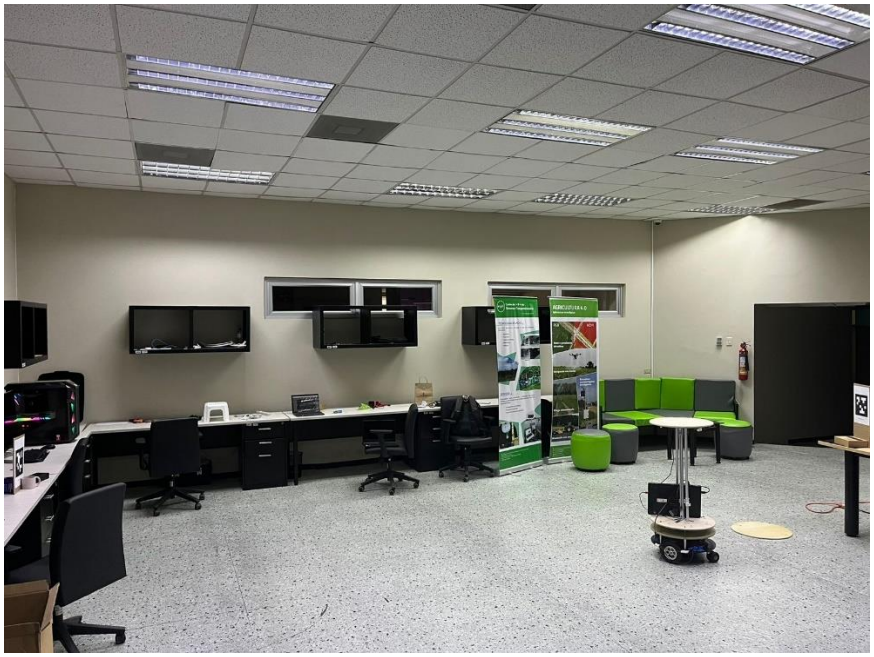


Figura 3.52

Robot en la estación base, pero sin calibrar la orientación

Figura 3.53

Robot calibrado en la estación base

3.6 Análisis de costos

Se realizó un análisis de costos de las diferentes partes de nuestro robot, todos los costos se distribuirán en las siguientes tablas:

Tabla 3.2

Tabla de precios de los elementos del robot

Componentes del robot de servicio

Elementos - Hardware	Unidades	Costo Unitario	Costo Total	Proveedor
Plataforma Arlo	1	$ 995.00	$ 995.00	Parallax
sistema computacional	1	$ 500.00	$ 500.00	N/A
Rplidar A1	1	$ 100.00	$ 100.00	Slam Tech
Cámara T265	1	$ 209.00	$ 209.00	Intel
Cámara D435i	1	$ 189.00	$ 189.00	Intel

ESP32	1	$	12.00	$	12.00	Espressif
Soporte Cámara T265 – Impresión 3D– Impresión 3D	1	$	10.00	$	10.00	CIDIS
Plancha de Plywood 2.40m x 1.20m	1	$	10.00	$	10.00	FERROCOSTA
Perno hexagonal M6x1x8mm	16	$	0.10	$	1.60	La casa del perno
Perno hexagonal M6x1x15mm	8	$	0.10	$	0.80	La casa del perno
Perno hexagonal M6x1x30mm	8	$	0.10	$	0.80	La casa del perno
Tuerca de 6mm	32	$	0.05	$	1.60	La casa del perno
Varilla de soporte cuadrada	1	$	5.35	$	5.35	Acercons
Varilla de soporte cilíndrica	2	$	1.00	$	2.00	Metal Hierro
tornillos	8	$	0.05	$	0.40	N/A
Total				$	2037.55	

Tabla 3.3

Tabla de precios de la mano de obra

Mano de obra			
Área - Actividad	Horas	Costo por hora	Costo Total
Ensamble	15	$ 3.00	$ 45.00
Programación	50	$ 4.00	$ 200.00
Diseño mecánico	20	$ 4.00	$ 80.00
Diseño del controlador	25	$ 4.00	$ 100.00
Simulación en ROS - Gazebo	25	$ 4.50	$ 112.50
Implementación de arquitectura	15	$ 3.50	$ 52.50
Total			$ 590.00

El costo final de construir y diseñar el robot fue de $2627.55 considerando todos los elementos, la mano de obra y sin contar la parte estética, los precios más altos son de la plataforma arlo, el sistema computacional y los sensores, aun así, el mercado oferta robots con el mismo propósito con un valor de $3500 a $6200, por lo tanto, el precio de nuestro robot es accesible de venta al público.

Como modelo de negocio se puede escoger dos opciones, la primera vendiendo directamente nuestro robot a un precio entre $3000 o $4000 y ofreciendo mantenimiento periódico por un precio extra o la segunda opción sería rentar nuestros robots a un precio de $350 basado en el sueldo mínimo que podría recibir un trabajador de $400, en este servicio de renta se ofrece también el mantenimiento, así proporcionando trabajo a los profesionales que manejen estas herramientas.

3.7 Análisis de consumo energético

Se obtuvo los consumos individuales de cada elemento del robot y se enlistó en la siguiente tabla:

Tabla 3.4

Tabla de consumo energético

Consumo energético			
Componente	Cantidad	consumo [W]	Consumo Total
Motor	2	30	60
Encoder	2	0.015	0.03
ESP32	1	0.66	0.66
RPLidar A1	1	5	5
Cámara D435i	1	2.25	2.25
Cámara T265	1	2.25	2.25
Total			70.19

Nota. Toda esta información se obtuvo de los manuales de la sección 2.2.1

Se obtuvo una potencia total de 70.19 W sumando los consumos de todos los elementos, esto se comparó directamente con las baterías en paralelo la cual brinda un voltaje de 12 V, 14 Ah, se obtuvo un valor de 168 Wh totales, haciendo una proporción entre el total consumido y la cantidad máxima de watts/hora capaz de entregar la batería, se obtuvo un tiempo límite de uso igual a 2.40 horas totales teóricas, este número total de horas es considerando que el robot está moviendo sus ruedas todo el tiempo, por lo tanto el número de horas reales será mayor, finalmente se concluye que estas horas fueron suficientes para cumplir nuestra jornada de trabajo propuesta.

Capítulo 4

4 Conclusiones y Recomendaciones

4.1 Conclusiones

- Se diseñó la estructura y funcionamiento de un sistema de entrega de paquetes en espacios cerrados el cual se implementó en un robot de servicio, este sistema permitió el correcto movimiento autónomo entre mesas y la posibilidad de esquivar obstáculos dinámicos con una velocidad de respuesta deunsegundo o menos.
- Se simuló toda la funcionalidad del robot utilizando gazebo y Rviz en ROS, estas simulaciones sirvieron para incorporar todas las funcionalidades en un solo comando, también sirvió para observar el comportamiento de las estructuras considerando inercias, finalmente esto permitió pulir el funcionamiento real del robot evitando averiar el robot y ahorrando tiempos desarrollo de código.
- Se programaron dos situaciones de funcionamiento, el escenario número uno en el cual el robot creó mapas 2D con alta resolución y el escenario dos en el cual se utilizaron estos mapas preestablecidos y técnicas de SLAM para el movimiento autónomo.
- Se diseñó la arquitectura de nodos tanto para el escenario de mapeo como para el escenario de navegación, todas las funcionalidades fueron interconectadas utilizando topics o servicio de ROS.
- Se eligió un centro computacional adecuado para el prototipo final, se había optado por una Jetson Nano desde un principio, pero al observar que la carga computacional sobrepasaba lo permitido por la Jetson, finalmente se cambió por una laptop con más poder computacional la cual permitió implementar todas las funcionalidades del prototipo final.

- Se realizó un análisis de costos el cual determinó que la construcción de este robot es de $2627.55 y entra dentro del rango permisible y rentable comparado con precios de entre $3000 y $6000.

4.2 Recomendaciones

- Como primera recomendación tuvo el tema del tiempo de funcionamiento, se recomienda que para trabajos futuros se cambie las baterías de plomo por una o varias baterías de litio, debido a que las baterías de plomo actuales bajaron su rendimiento general, lo que impacta directamente al tiempo de jornada del robot, las principales ventajas de las baterías de litio son que tienen un mayor número de ciclos de vida, son más livianas y compactas, ofrecen una mayor eficiencia y una menor tasa de autodescarga.
- Como segundo punto se recomienda que al momento de elegir un microcontrolador se considere cuanto poder computacional tomará correr todos los nodos del programa en ROS, estos requerimiento de mayor potencia computacional impactará directamente a que tan rápido responda el robot y cuantas funcionalidades finales posea, en nuestro proyecto se propuso utilizar una Jetson Nano pero debido a la carga computacional requerida, se optó por utilizar una laptop la cual ayudó a mejorar el rendimiento y funcionalidades del robot.
- Como tercer punto, se recomienda que se cambie el controlador principal de los motores, el DHB-10 tiene un bloqueo de ruedas que se activa cada vez que recibe una sobre corriente, este bloqueo es muy sensible a los cambios de corrientes y al momento de mover bruscamente el robot de un punto a otro, las ruedas se bloquean ocasionando que el robot tenga que ser reiniciado, para futuros proyectos se podría cambiar el controlador por uno que no tenga estas limitaciones.

REFERENCIAS

[1] CMV, «El mercurio,» El mercurio, 11 noviembre 2020. [En línea]. Available: https://elmercurio.com.ec/2020/11/11/los-robots-un-nuevo-valor-en-situacion-de-pandemia/. [Último acceso: 3 noviembre 2023].

[2] B. C. d. Ecuador, «Programación Macroeconómica Sector Real 2022-2026,» Banco Central del Ecuador, Quito, 2023.

[3] I. Bear Robotics, «bearrobotics,» Bear Robotics, Inc, 1 enero 2017. [En línea]. Available: https://www.bearrobotics.ai/servi. [Último acceso: 3 noviembre 2023].

[4] I. Bear Robotics, «bearrobotics,» Bear Robotics, Inc, 1 enero 2017. [En línea]. Available: https://www.bearrobotics.ai/specification. [Último acceso: 3 noviembre 2023].

[5] I. Bear Robotics, «bearrobotics,» 1 enero 2017. [En línea]. Available: https://static1.squarespace.com/static/614d1cf7b97a3d6e357abcb3/t/64593b7b8f4ba02152871f2a/1683569531264/Servi+User+Manual+.pdf. [Último acceso: 4 noviembre 2023].

[6] R. Robotics, «relayrobotics,» Relay Robotics, 1 enero 2023. [En línea]. Available: https://www.relayrobotics.com/relay2. [Último acceso: 4 noviembre 2023].

[7] IEEE, «robotsguide,» IEEE, 1 enero 2023. [En línea]. Available: https://robotsguide.com/robots/relay. [Último acceso: 4 noviembre 2023].

[8] I. Relay Robotics, «relayrobotics,» Relay Robotics, Inc, 1 enero 2023. [En línea]. Available: https://www.relayrobotics.com/blog/2019/11/6/5-ways-relay-autonomous-delivery-robots-are-so-cost-effective. [Último acceso: 4 noviembre 2023].

[9] M. Misaros, O.-P. Stan, I.-C. Donca y L.-C. Miclea, «Autonomous Robots for Services-State of the Art, Challenges, and Research Areas,» *Sensors,* vol. 23, nº 4962, pp. 1-31, 2023.

[10] I. Lee, «Service Robots: A Systematic Literature Review,» *Electronics ,* vol. 10, nº 2658, pp. 1-29, 2021.

[11] S.-W. Lee, «kedglobal,» The korean economic daily, 17 agosto 2022 . [En línea]. Available: https://www.kedglobal.com/robotics/newsView/ked202208170019. [Último acceso: 4 noviembre 2023].

[12] S. K. M. &. J. Majumdar, «Kinematics, Localization and Control of Differential Drive Mobile Robot,» *Global Journals Inc,* vol. 14, nº 2249-4596, pp. 1-9, 2014.

[13] J. J. Cuesta, «riunet,» 1 diciembre 2012. [En línea]. Available: https://riunet.upv.es/bitstream/handle/10251/18173/Memoria.pdf?sequence=1. [Último acceso: 4 noviembre 2023].

[14] O. Robotics, «ros,» Open Robotics, 1 enero 2021. [En línea]. Available: https://www.ros.org/. [Último acceso: 4 noviembre 2023].

[15] H. &. X. J. &. X. Z. Deng, «Mobile manipulation task simulation using ROS with MoveIt,» *IEEE International Conference on Real-time Computing and Robotics,* vol. 1, nº 10.1109/RCAR.2017.8311930, pp. 612-616, 2017.

[16] H. D.-W. a. T. Bailey, «Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms,» *IEEE Robotics & Automation,* vol. 13, nº 2, pp. 99-110, 1 enero 2006.

[17] K. M. L. a. F. C. Park, MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL, Cambridge : Cambridge University, 2017.

[18] J. &. S. S. Zhang, «LOAM : Lidar Odometry and Mapping in real-time,» *Robotics: Science and Systems Conference,* vol. 1, nº 1, pp. 109-111, 1 enero 2014.

[19] J. Z. a. S. Singh, «Visual-lidar Odometry and Mapping: Low-drift, Robust, and Fast,» *IEEE International Conference on Robotics and Automation (ICRA),* vol. 1, nº 1, pp. 26-30, 15 mayo 2015.

[20] J. A. S. J. C. H. A. N. I. V. C. L. &. C. J. A. Placed, «A survey on active simultaneous localization and mapping: State of the,» *IEEE Transactions on Robotics,* vol. 1, nº 2207.00254, pp. 1-20, 2023.

[21] J. Ren, T. Wu, X. Zhou, C. Yang, J. Sun, M. Li, H. Jiang y A. Zhang, «SLAM, Path Planning Algorithm and Application Research of an Indoor Substation Wheeled Robot Navigation System,» *Electronics ,* vol. 11, nº 1838, pp. 1-21, 2022.

[22] V. Lidar, «velodynelidar,» Velodyne Lidar, 1 enero 2023. [En línea]. Available: https://velodynelidar.com/what-is-lidar/. [Último acceso: 4 noviebre 2023].

[23] A. Haider y H. Hel-Or, «What Can We Learn from Depth Camera Sensor Noise?,» *Sensors,* vol. 22, nº 5448, pp. 1-16, 2022.

[24] R. R. A. M. S. U. S. R. D. D. Anood Ibrahim, «Control Systems in Robotics: A Review,» *International Journal of Engineering Inventions,* vol. 5, nº 2278-7461, pp. 29-38, 2016.

[25] A. Robotics, «medium,» medium, 2 julio 2019. [En línea]. Available: https://medium.com/autonomous-robotics/pid-control-85596db59f35. [Último acceso: 4 noviembre 2023].

[26] Parallax, «parallax,» Parallax, 1 enero 1990. [En línea]. Available: https://www.parallax.com/product/arlo-complete-robot-system/. [Último acceso: 30 11 2023].

[27] Parallax, «parallax,» Parallax, 1 enero 1990. [En línea]. Available: https://www.parallax.com/product/motor-mount-wheel-kit-aluminum/. [Último acceso: 4 12 2023].

[28] Parallax, «parallax,» 1 enero 1990. [En línea]. Available: https://www.parallax.com/product/36-position-quadrature-encoder-set/. [Último acceso: 4 diciembre 2023].

[29] Parallax, «parallax,» 1 enero 1990. [En línea]. Available: https://www.parallax.com/package/dhb-10-motor-controller-product-guide/. [Último acceso: 4 diciembre 2023].

[30] Slamtec, «slamtec,» Slamtec, 1 enero 2009. [En línea]. Available: https://www.slamtec.com/en/lidar/a1. [Último acceso: 8 diciembre 2023].

[31] S. RC, «skyrc,» SKY RC, 1 enero 2023. [En línea]. Available: https://www.skyrc.com/iMAX_B6mini_Charger. [Último acceso: 8 diciembre 2023].

[32] ESPRESSIF, «espressif,» ESPRESSIF, 1 enero 2008. [En línea]. Available: https://www.espressif.com/en/products/devkits/esp32-devkitc. [Último acceso: 12 diciembre 2023].

[33] Intel, «intelrealsense,» Intel, 1 septiembre 2019. [En línea]. Available: https://dev.intelrealsense.com/docs/depth-and-tracking-cameras-alignment. [Último acceso: 12 diciembre 2023].

[34] R. S. Lab, «github,» Robotic Systems Lab, 1 enero 2017. [En línea]. Available: https://github.com/leggedrobotics/darknet_ros. [Último acceso: 16 enero 2024].

[35] NVIDIA, «nvidia,» NVIDIA, 19 Marzo 2019. [En línea]. Available: https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/. [Último acceso: 12 diciembre 2023].

[36] X. Yáng, «github.com,» Google, 1 enero 2016. [En línea]. Available: https://github.com/cartographer-project/cartographer. [Último acceso: 12 diciembre 2023].

[37] O. Robotics, «wiki.ros.org,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/amcl. [Último acceso: 12 diciembre 2023].

[38] O. Robotics, «wiki.ros.org,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/rosserial_python. [Último acceso: 12 diciembre 2023].

[39] Parallax, «parallax,» Parallax, 1 enero 1990. [En línea]. Available: https://www.parallax.com/package/dhb-10-motor-controller-firmware/. [Último acceso: 11 enero 2024].

[40] O. Robotics, «Gazebosim.org,» Open Robotics, [En línea]. Available: https://gazebosim.org/home.

[41] O. Robotics, «classic.gazebosim.org,» Open Robotics, 1 enero 2014. [En línea]. Available: https://classic.gazebosim.org/tutorials?tut=ros_gzplugins. [Último acceso: 15 enero 2024].

[42] O. Robotics, «docs.ros.org,» Open Roboics, 1 enero 2012. [En línea]. Available: https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud.html. [Último acceso: 15 enero 2024].

[43] fverdoja, «github,» fverdoja, 1 enero 2020. [En línea]. Available: https://github.com/fverdoja/ros_maps_to_pedsim. [Último acceso: 19 enero 2024].

[44] B. Okal y T. Linder, «github,» Social Robotics Lab, 1 enero 2014. [En línea]. Available: https://github.com/srl-freiburg/pedsim_ros. [Último acceso: 19 enero 2024].

[45] A. R. Laboratory, «april.eecs.umich.edu,» APRIL Robotics Laboratory, 1 enero 2010. [En línea]. Available: https://april.eecs.umich.edu/software/apriltag. [Último acceso: 12 diciembre 2023].

[46] D. Foxy, W. Burgar y S. Thrunyz, «The Dynamic Window Approach to Collision Avoidance,» *IEEE Robotics & Automation Magazine,* vol. 4, nº 10.1109/100.580977, pp. 23 - 33, 1997.

[47] A. Robotics, «github,» April Robotics, 1 enero 2010. [En línea]. Available: https://github.com/AprilRobotics/apriltag_ros. [Último acceso: 16 enero 2024].

[48] P. a. Ż. A. a. B. A. Żur, «Finite Elements Analysis of PLA 3D-printed Elements and Shape Optimization,» *European Journal of Engineering Science and Technology,* enero 2019.

[49] Matweb, «matweb,» Matweb, 1 enero 1996. [En línea]. Available: https://www.matweb.com/search/datasheet_print.aspx?matguid=bd6620450973496ea2578c283e9fb807. [Último acceso: 11 enero 2024].

[50] O. Robotics, «wiki.ros,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/Nodes. [Último acceso: 4 noviembre 2023].

[51] O. Robotics, «wiki.ros,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/Topics. [Último acceso: 4 noviembre 2023].

[52] O. Robotics, «wiki.ros,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/Messages. [Último acceso: 4 noviembre 2023].

[53] O. robotics, «docs.ros,» Open robotics, 1 enero 2021. [En línea]. Available: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html. [Último acceso: 1 noviembre 2023].

[54] O. Robotics, «metrics.ros,» Open Robotics, 1 enero 2021. [En línea]. Available: https://metrics.ros.org/packages_rosdistro.html. [Último acceso: 4 noviembre 2023].

[55] O. Robotics, «wiki.ros,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/noetic. [Último acceso: 12 diciembre 2023].

[56] O. Robotics, «wiki.ros.org,» Open Robotics, 1 enero 2021. [En línea]. Available: http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython. [Último acceso: 12 diciembre 2023].

[57] OpenCV, «docs.opencv.org,» OpenCV, 1 diciembre 1999. [En línea]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. [Último acceso: 14 diciembre 2023].

[58] RobertBabuˇska y StefanoStramigioli, «MatlabandSimulinkforModelingandControl,» DelftUniversityofTechnology, Delft, 1999.

[59] O. Robotics, «Gazebo-classic,» Open Robotics, 1 Enero 2020. [En línea]. Available: https://github.com/gazebosim/gazebo-classic/tree/gazebo11/worlds.

APÉNDICES

APÉNDICE A

Video demostrativo del funcionamiento del robot

https://www.youtube.com/watch?v=WiqjhNtHD0s

APÉNDICE B

Aplicación móvil para el manejo del robot

Figura B.1

Ventana de mapeo

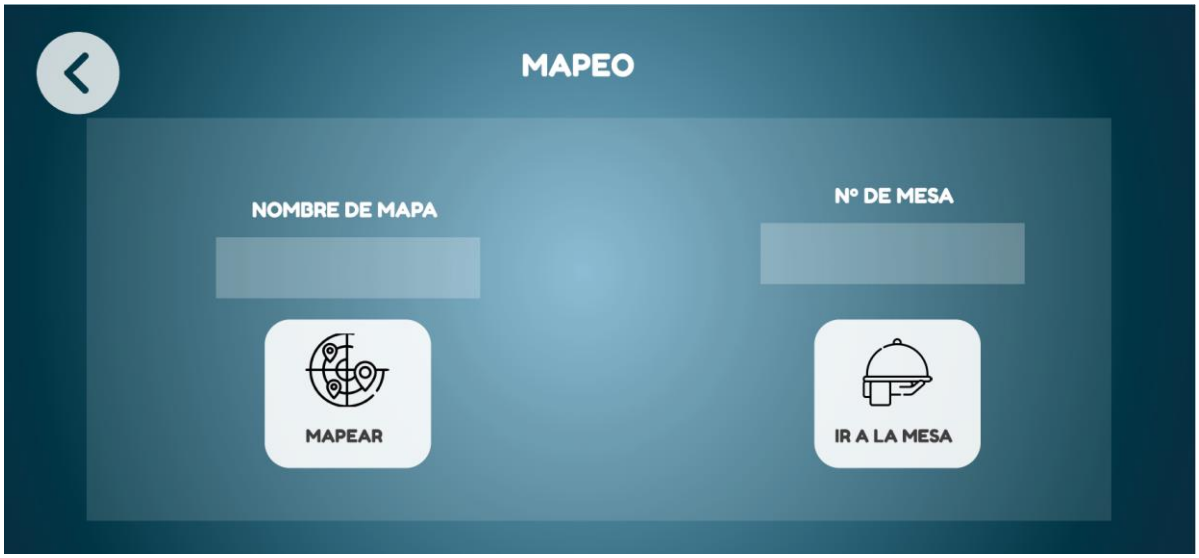
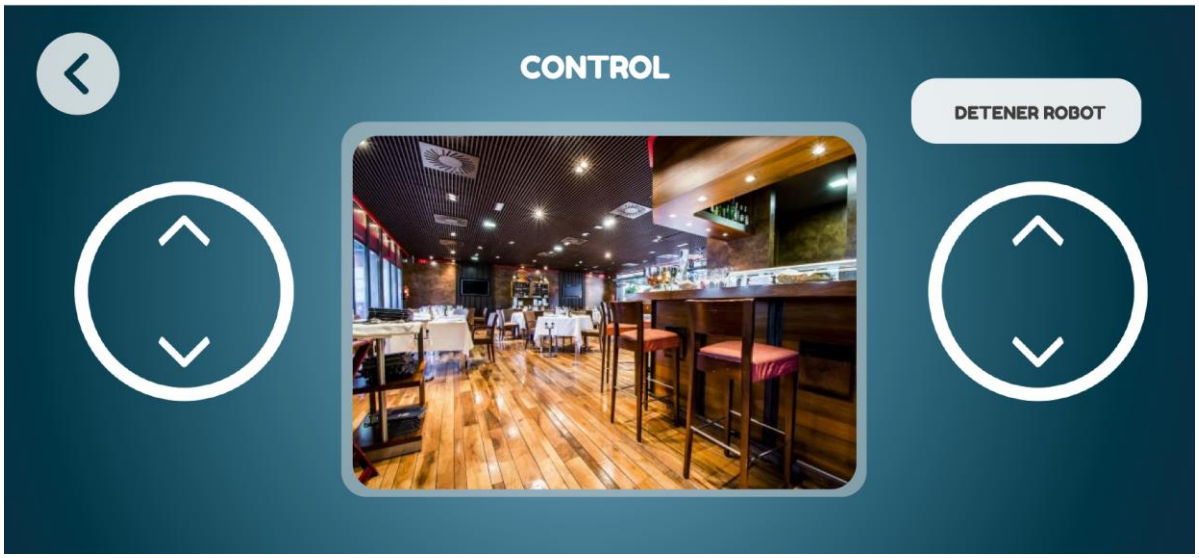


Figura B.2

Ventana de navegación



APÉNDICE C

Código

Figura C.1

Código del controlador primera parte

```python
#codkevin, lineas iniciales:230, ahora:144
import uros
from arlo_control_msgs import WheelsEncoders  # rosserial messages
from arlorobot import ArloRobot
import utime
import gc
from machine import Pin, ADC, PWM
import network
import _thread
from std_msgs import Float32, Bool
import machine

gc.enable()

def do_connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        wlan.connect("covibot", "covibot1")
        while not wlan.isconnected():
            pass

do_connect()

utime.sleep(20)

class ArlorobotMove(object):
    def __init__(self):

        utime.sleep(1)

        # constants
        self.WHEEL_VELOCITY_TOPIC = "/wheels_vel"
        self.ESP_VELOCITY_TOPIC ="/esp_velocity"
        self.PRESENCE_TOPIC ="/presence"
        self.arlobot = ArloRobot(serial_id=2, baudrate=19200, tx=17, rx=16, pace=2)
        self.node = uros.NodeHandle(1, 115200, tx=1, rx=3)
        # , tx=1, rx=3
        self.left_power = 0
        self.right_power = 0

        self.last_left_power = 0
        self.last_right_power = 0

        #LEDS
        self.RED_PIN = 14
        self.GREEN_PIN = 27
```

Figura C.2

Código del controlador segunda parte

```
        self.GREEN_PIN = 27
        self.BLUE_PIN = 12
        self.RED = PWM(Pin(self.RED_PIN), freq=5000)
        self.GREEN = PWM(Pin(self.GREEN_PIN),freq=5000)
        self.BLUE = PWM(Pin(self.BLUE_PIN), freq =5000)
        self.real_left_encoder = 0.0
        self.real_right_encoder = 0.0
        self.real_velocity = Float32()
        self.is_person_near = Bool()
    def turn_on_color(self):
        self.RED.duty(1023)
        self.GREEN.duty(0)
        self.BLUE.duty(0)
    def turn_off_color(self):
        self.RED.duty(0)
        self.GREEN.duty(0)
        self.BLUE.duty(0)
    def presence_callback(self,msg):
        self.is_person_near.data = msg.data
        if(self.is_person_near):
            self.turn_on_color()
        else:
            self.turn_off_color()

    def wheel_velocity_callback(self, msg):
        self.left_power = msg.left_encoder
        self.right_power = msg.right_encoder
        self.left_power = int(self.left_power)
        self.right_power = int(self.right_power)

        if (
            self.last_left_power != self.left_power
            or self.last_right_power != self.right_power
        ):
            self.last_left_power = self.left_power
            self.last_right_power = self.right_power

        if self.left_power > 120:
            self.left_power = 120
        elif self.left_power < -120:
            self.left_power = -120

        if self.right_power > 120:
            self.right_power = 120
        elif self.right_power < -120:
            self.right_power = -120
```

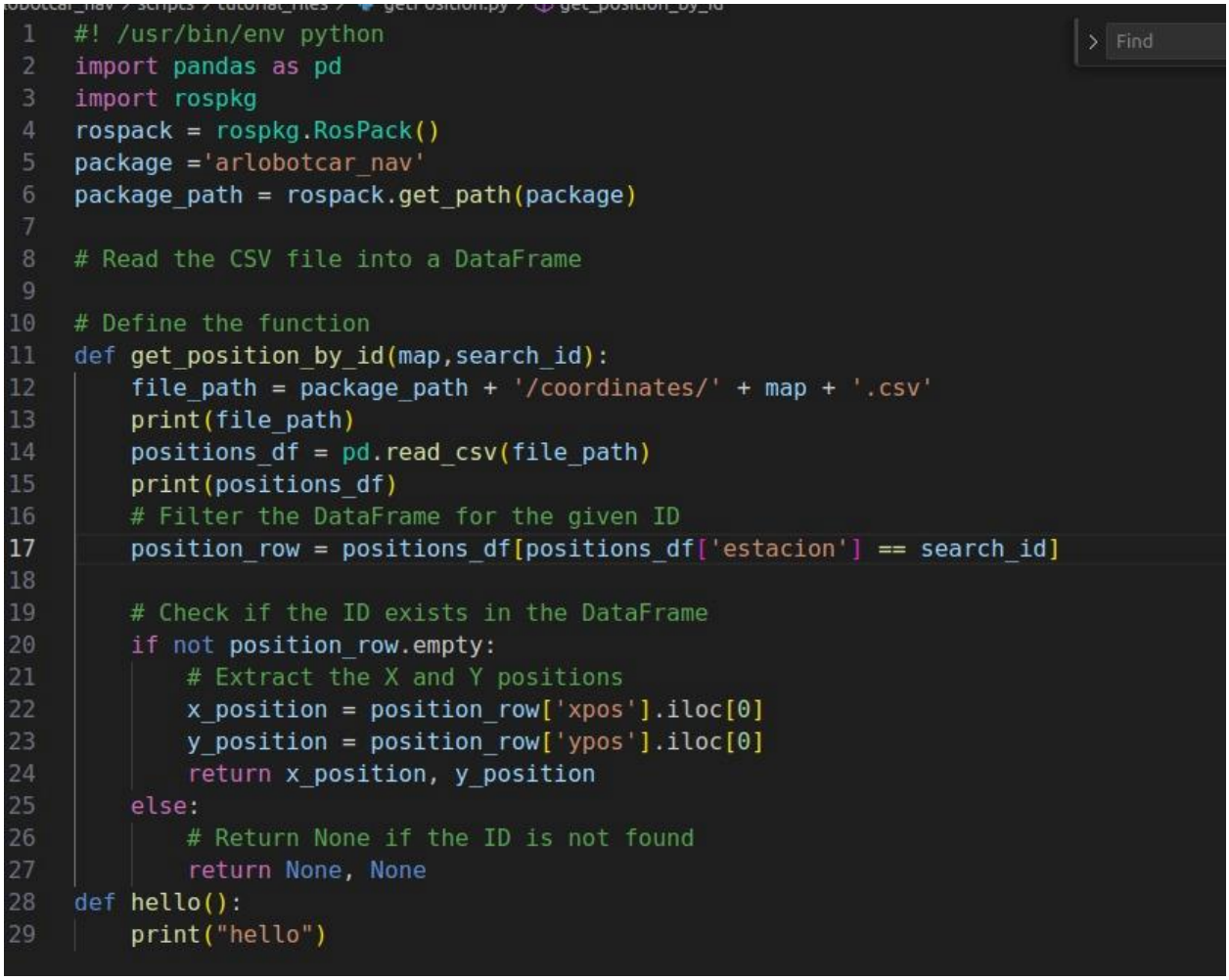
Figura C.3

Código del controlador tercera parte

```
            self.right_power = 120
        elif self.right_power < -120:
            self.right_power = -120

    def process_list(self,input_list):
        if len(input_list) == 2:
            try:
                self.real_left_encoder = float(input_list[0])
                self.real_right_encoder = float(input_list[1])
                #print("Both elements can be converted to float:", input_list)
            except Exception as e:
                self.real_left_encoder = 1.0
                self.real_right_encoder = 1.0
                #print(f"Exception encountered: {e}")
        else:
            self.real_left_encoder = 1.0
            self.real_right_encoder = 1.0

    def start(self):  # threads init and running
        self.arlobot.clear_counts()
        self.node.subscribe(
            self.WHEEL_VELOCITY_TOPIC, WheelsEncoders, self.wheel_velocity_callback
        )
        self.node.subscribe(
            self.PRESENCE_TOPIC, Bool, self.presence_callback
        )
        self.arlobot.set_kp_constant(100)
        self.arlobot.set_ki_constant(80)
        self.arlobot.turn_color()
        #self.arlobot.go_speed(0,0)
        while True:
            #self.left_power, self.right_power

            self.arlobot.go_speed(self.left_power, self.right_power)
            raw_velocity = self.arlobot.read_speeds()
            self.process_list(raw_velocity)
            left_vel = self.real_left_encoder / 300
            right_vel = self.real_right_encoder / 300
            self.real_velocity.data = float((left_vel + right_vel) * 0.5)
            self.node.publish(self.ESP_VELOCITY_TOPIC,self.real_velocity)
            utime.sleep_ms(100)

my_arlobot = ArlorobotMove()

my_arlobot.start()
```

Figura C.4

Código para obtener la posición



```python
#! /usr/bin/env python
import pandas as pd
import rospkg
rospack = rospkg.RosPack()
package ='arlobotcar_nav'
package_path = rospack.get_path(package)

# Read the CSV file into a DataFrame

# Define the function
def get_position_by_id(map,search_id):
    file_path = package_path + '/coordinates/' + map + '.csv'
    print(file_path)
    positions_df = pd.read_csv(file_path)
    print(positions_df)
    # Filter the DataFrame for the given ID
    position_row = positions_df[positions_df['estacion'] == search_id]

    # Check if the ID exists in the DataFrame
    if not position_row.empty:
        # Extract the X and Y positions
        x_position = position_row['xpos'].iloc[0]
        y_position = position_row['ypos'].iloc[0]
        return x_position, y_position
    else:
        # Return None if the ID is not found
        return None, None
def hello():
    print("hello")
```

Figura C.5

Código de la configuración del AprilTag primera parte

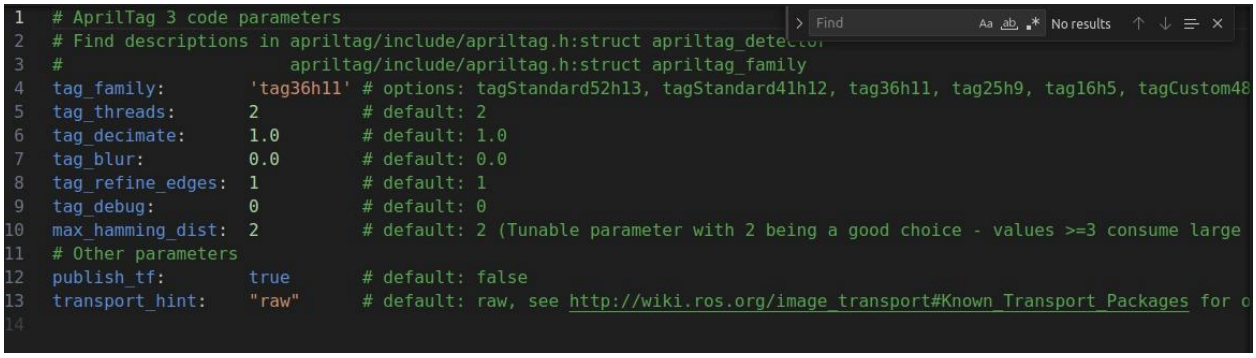
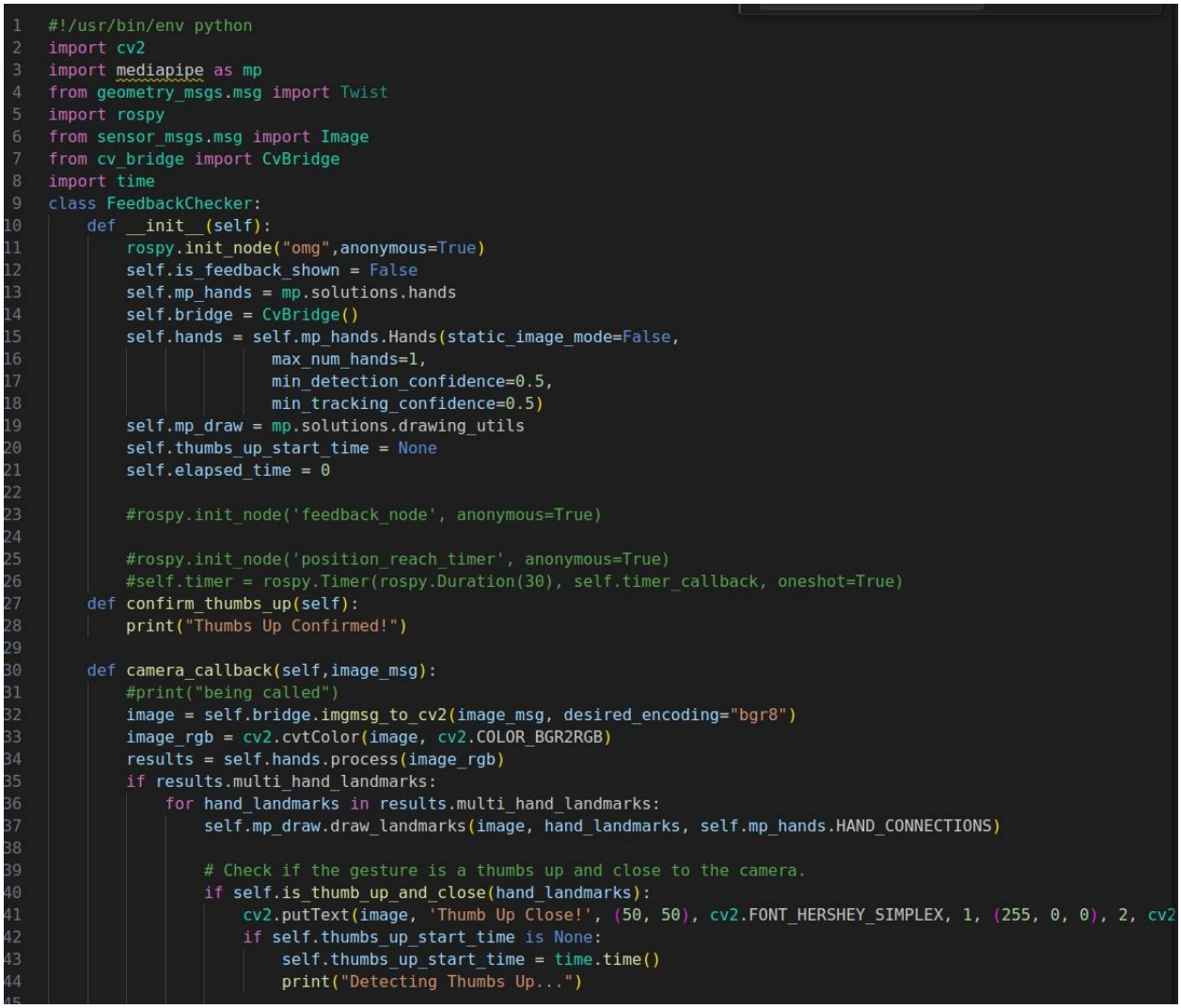


Figura C.6

Código de la configuración del AprilTag segunda parte

Figura C.7

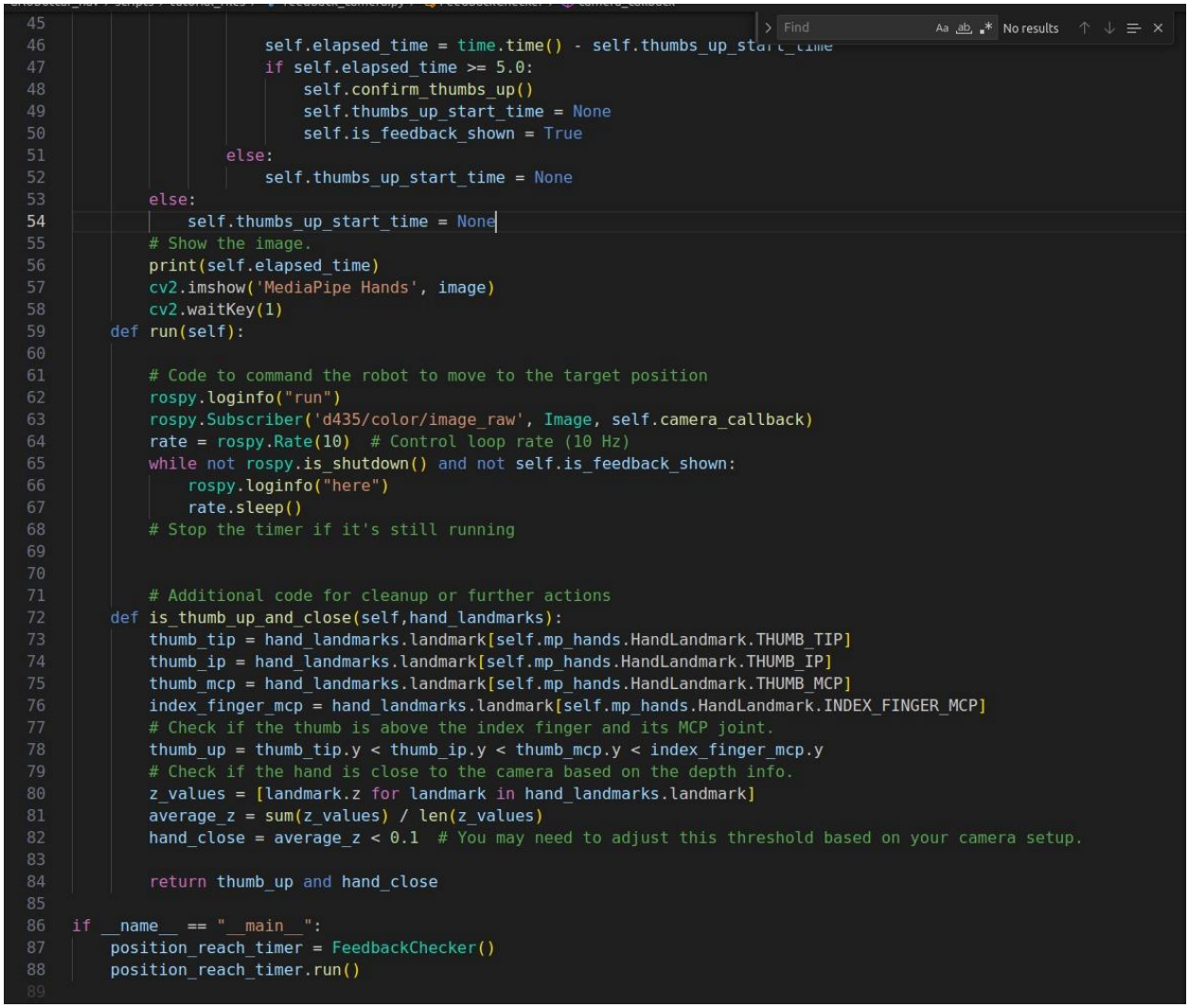
Código del feedback de la cámara primera parte



```python
#!/usr/bin/env python
import cv2
import mediapipe as mp
from geometry_msgs.msg import Twist
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import time
class FeedbackChecker:
    def __init__(self):
        rospy.init_node("omg",anonymous=True)
        self.is_feedback_shown = False
        self.mp_hands = mp.solutions.hands
        self.bridge = CvBridge()
        self.hands = self.mp_hands.Hands(static_image_mode=False,
                    max_num_hands=1,
                    min_detection_confidence=0.5,
                    min_tracking_confidence=0.5)
        self.mp_draw = mp.solutions.drawing_utils
        self.thumbs_up_start_time = None
        self.elapsed_time = 0

        #rospy.init_node('feedback_node', anonymous=True)

        #rospy.init_node('position_reach_timer', anonymous=True)
        #self.timer = rospy.Timer(rospy.Duration(30), self.timer_callback, oneshot=True)
    def confirm_thumbs_up(self):
        print("Thumbs Up Confirmed!")

    def camera_callback(self,image_msg):
        #print("being called")
        image = self.bridge.imgmsg_to_cv2(image_msg, desired_encoding="bgr8")
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = self.hands.process(image_rgb)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                self.mp_draw.draw_landmarks(image, hand_landmarks, self.mp_hands.HAND_CONNECTIONS)

                # Check if the gesture is a thumbs up and close to the camera.
                if self.is_thumb_up_and_close(hand_landmarks):
                    cv2.putText(image, 'Thumb Up Close!', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2
                    if self.thumbs_up_start_time is None:
                        self.thumbs_up_start_time = time.time()
                        print("Detecting Thumbs Up...")
```

Figura C.8

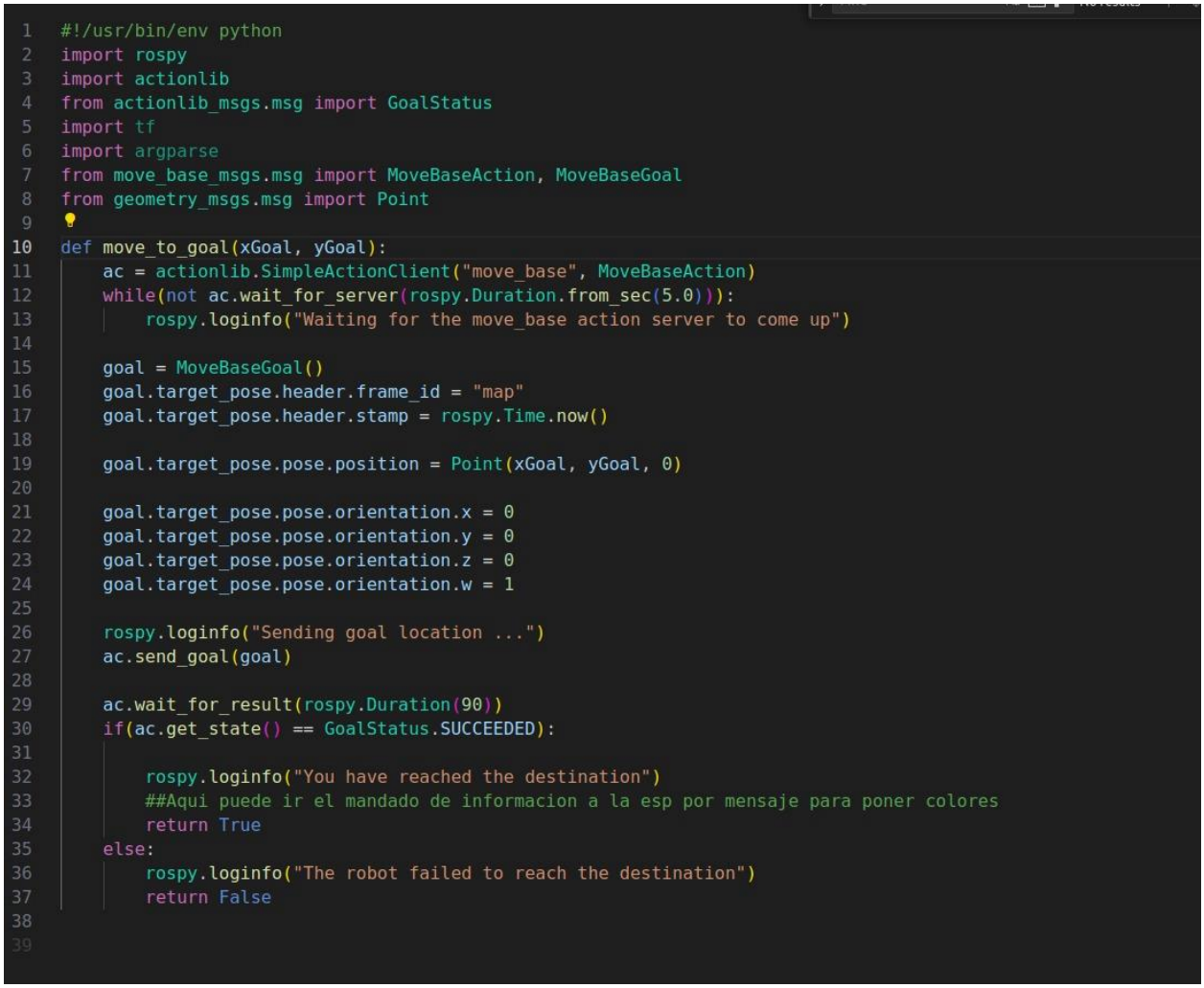
Código del feedback de la cámara segunda parte



```
                    self.elapsed_time = time.time() - self.thumbs_up_start_time
                    if self.elapsed_time >= 5.0:
                        self.confirm_thumbs_up()
                        self.thumbs_up_start_time = None
                        self.is_feedback_shown = True
                else:
                    self.thumbs_up_start_time = None
            else:
                self.thumbs_up_start_time = None
            # Show the image.
            print(self.elapsed_time)
            cv2.imshow('MediaPipe Hands', image)
            cv2.waitKey(1)
        def run(self):

            # Code to command the robot to move to the target position
            rospy.loginfo("run")
            rospy.Subscriber('d435/color/image_raw', Image, self.camera_callback)
            rate = rospy.Rate(10)  # Control loop rate (10 Hz)
            while not rospy.is_shutdown() and not self.is_feedback_shown:
                rospy.loginfo("here")
                rate.sleep()
            # Stop the timer if it's still running


            # Additional code for cleanup or further actions
        def is_thumb_up_and_close(self,hand_landmarks):
            thumb_tip = hand_landmarks.landmark[self.mp_hands.HandLandmark.THUMB_TIP]
            thumb_ip = hand_landmarks.landmark[self.mp_hands.HandLandmark.THUMB_IP]
            thumb_mcp = hand_landmarks.landmark[self.mp_hands.HandLandmark.THUMB_MCP]
            index_finger_mcp = hand_landmarks.landmark[self.mp_hands.HandLandmark.INDEX_FINGER_MCP]
            # Check if the thumb is above the index finger and its MCP joint.
            thumb_up = thumb_tip.y < thumb_ip.y < thumb_mcp.y < index_finger_mcp.y
            # Check if the hand is close to the camera based on the depth info.
            z_values = [landmark.z for landmark in hand_landmarks.landmark]
            average_z = sum(z_values) / len(z_values)
            hand_close = average_z < 0.1  # You may need to adjust this threshold based on your camera setup.

            return thumb_up and hand_close

    if __name__ == "__main__":
        position_reach_timer = FeedbackChecker()
        position_reach_timer.run()
```

Figura C.9

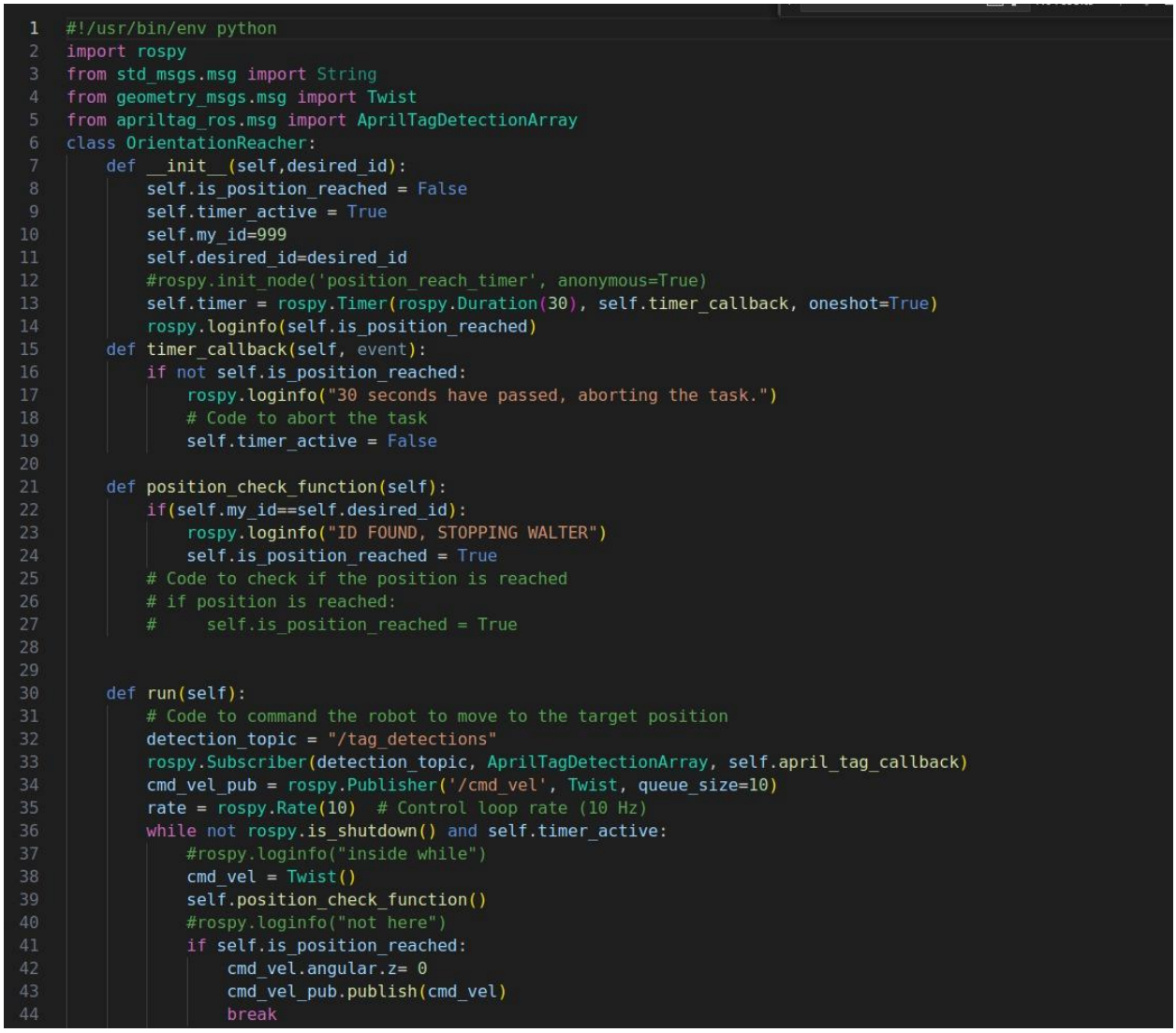
Código para moverse a las mesas



```python
#!/usr/bin/env python
import rospy
import actionlib
from actionlib_msgs.msg import GoalStatus
import tf
import argparse
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from geometry_msgs.msg import Point

def move_to_goal(xGoal, yGoal):
    ac = actionlib.SimpleActionClient("move_base", MoveBaseAction)
    while(not ac.wait_for_server(rospy.Duration.from_sec(5.0))):
        rospy.loginfo("Waiting for the move_base action server to come up")

    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()

    goal.target_pose.pose.position = Point(xGoal, yGoal, 0)

    goal.target_pose.pose.orientation.x = 0
    goal.target_pose.pose.orientation.y = 0
    goal.target_pose.pose.orientation.z = 0
    goal.target_pose.pose.orientation.w = 1

    rospy.loginfo("Sending goal location ...")
    ac.send_goal(goal)

    ac.wait_for_result(rospy.Duration(90))
    if(ac.get_state() == GoalStatus.SUCCEEDED):

        rospy.loginfo("You have reached the destination")
        ##Aqui puede ir el mandado de informacion a la esp por mensaje para poner colores
        return True
    else:
        rospy.loginfo("The robot failed to reach the destination")
        return False
```

Figura C.10

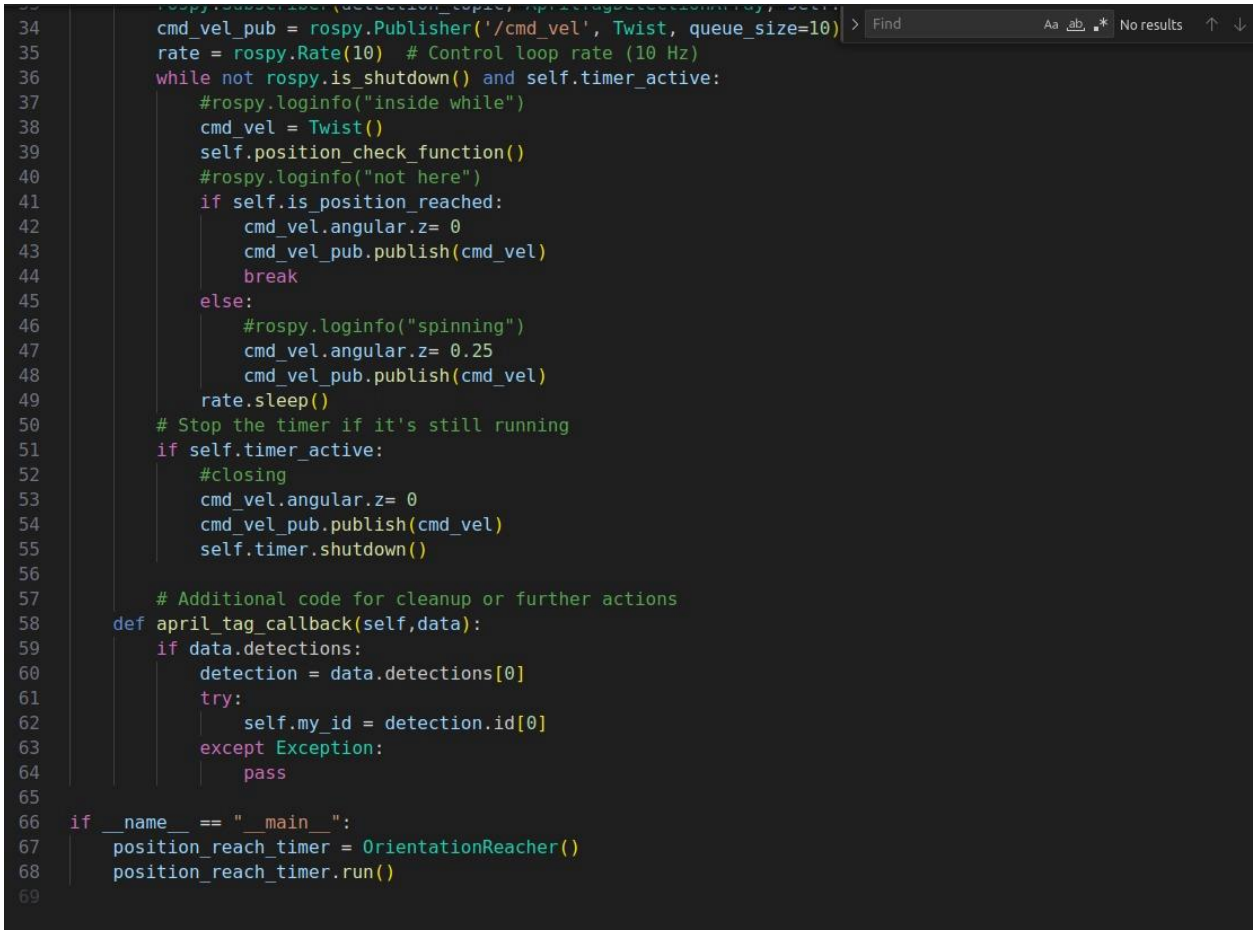
Código para la orientación con el AprilTag primera parte



```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Twist
from apriltag_ros.msg import AprilTagDetectionArray
class OrientationReacher:
    def __init__(self,desired_id):
        self.is_position_reached = False
        self.timer_active = True
        self.my_id=999
        self.desired_id=desired_id
        #rospy.init_node('position_reach_timer', anonymous=True)
        self.timer = rospy.Timer(rospy.Duration(30), self.timer_callback, oneshot=True)
        rospy.loginfo(self.is_position_reached)
    def timer_callback(self, event):
        if not self.is_position_reached:
            rospy.loginfo("30 seconds have passed, aborting the task.")
            # Code to abort the task
            self.timer_active = False

    def position_check_function(self):
        if(self.my_id==self.desired_id):
            rospy.loginfo("ID FOUND, STOPPING WALTER")
            self.is_position_reached = True
        # Code to check if the position is reached
        # if position is reached:
        #     self.is_position_reached = True


    def run(self):
        # Code to command the robot to move to the target position
        detection_topic = "/tag_detections"
        rospy.Subscriber(detection_topic, AprilTagDetectionArray, self.april_tag_callback)
        cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        rate = rospy.Rate(10)  # Control loop rate (10 Hz)
        while not rospy.is_shutdown() and self.timer_active:
            #rospy.loginfo("inside while")
            cmd_vel = Twist()
            self.position_check_function()
            #rospy.loginfo("not here")
            if self.is_position_reached:
                cmd_vel.angular.z= 0
                cmd_vel_pub.publish(cmd_vel)
                break
```

Figura C.11

Código para la orientación con el AprilTag segunda parte



```
        cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        rate = rospy.Rate(10)  # Control loop rate (10 Hz)
        while not rospy.is_shutdown() and self.timer_active:
            #rospy.loginfo("inside while")
            cmd_vel = Twist()
            self.position_check_function()
            #rospy.loginfo("not here")
            if self.is_position_reached:
                cmd_vel.angular.z= 0
                cmd_vel_pub.publish(cmd_vel)
                break
            else:
                #rospy.loginfo("spinning")
                cmd_vel.angular.z= 0.25
                cmd_vel_pub.publish(cmd_vel)
            rate.sleep()
        # Stop the timer if it's still running
        if self.timer_active:
            #closing
            cmd_vel.angular.z= 0
            cmd_vel_pub.publish(cmd_vel)
            self.timer.shutdown()

        # Additional code for cleanup or further actions
    def april_tag_callback(self,data):
        if data.detections:
            detection = data.detections[0]
            try:
                self.my_id = detection.id[0]
            except Exception:
                pass

if __name__ == "__main__":
    position_reach_timer = OrientationReacher()
    position_reach_timer.run()
```

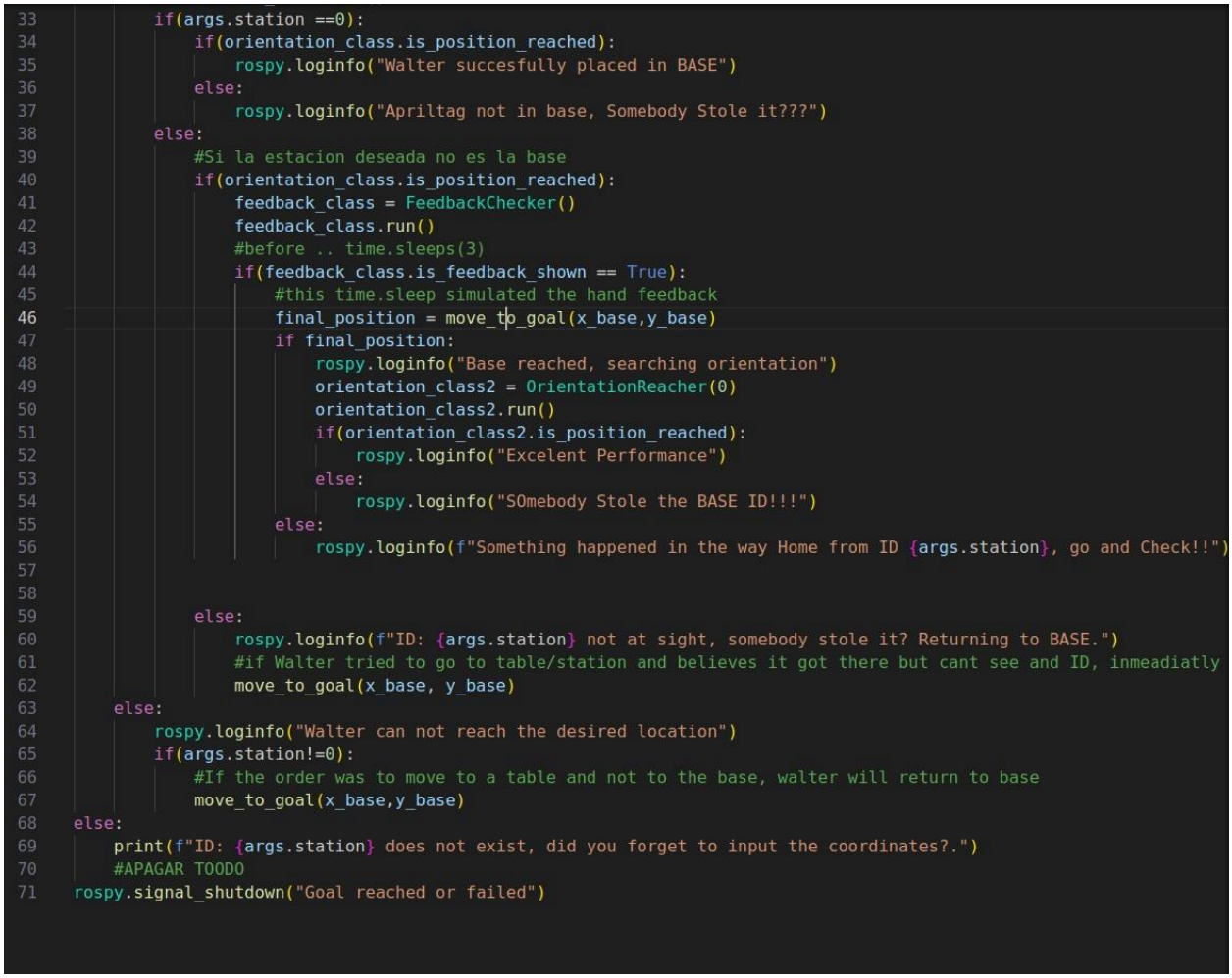
Figura C.12

Código de funcionalidad completa del robot primera parte

```python
#! /usr/bin/env python
import pandas as pd
import rospy
import time
#import buscar_orientacion
#import move_base_test
#import move_base_real
import argparse
from tutorial_files.getPosition import get_position_by_id
from tutorial_files.move_to_table import move_to_goal
from tutorial_files.new_orientation import OrientationReacher
from tutorial_files.feedback_camera import FeedbackChecker
# Example usage

if __name__ == '__main__':
    #hello()
    rospy.init_node('map_navigation', anonymous=False)
    parser = argparse.ArgumentParser(description='Move the robot to a goal (x, y, yaw).')
    parser.add_argument('map', type=str, help='map name for csv')
    parser.add_argument('station', type=float, help='The Station that you want the robot to move')

    args = parser.parse_args()
    x_base , y_base = get_position_by_id(args.map,0)
    x_pos, y_pos = get_position_by_id(args.map, args.station)
    if x_pos is not None and y_pos is not None:
        print(f"ID: {args.station}, X Position: {x_pos}, Y Position: {y_pos}")
        rospy.loginfo("Start go to goal")
        succes_position =move_to_goal(x_pos,y_pos)
        if succes_position:
            rospy.loginfo("Goal reached, searching orientation")
            orientation_class = OrientationReacher(args.station)
            orientation_class.run()
            if(args.station ==0):
                if(orientation_class.is_position_reached):
                    rospy.loginfo("Walter succesfully placed in BASE")
                else:
                    rospy.loginfo("Apriltag not in base, Somebody Stole it???")
            else:
                #Si la estacion deseada no es la base
                if(orientation_class.is_position_reached):
                    feedback_class = FeedbackChecker()
                    feedback_class.run()
                    #before .. time.sleeps(3)
                    if(feedback_class.is_feedback_shown == True):
                        #this time.sleep simulated the hand feedback
```

Figura C.13

Código de funcionalidad completa del robot segunda parte



```
        if(args.station ==0):
            if(orientation_class.is_position_reached):
                rospy.loginfo("Walter succesfully placed in BASE")
            else:
                rospy.loginfo("Apriltag not in base, Somebody Stole it???")
        else:
            #Si la estacion deseada no es la base
            if(orientation_class.is_position_reached):
                feedback_class = FeedbackChecker()
                feedback_class.run()
                #before .. time.sleeps(3)
                if(feedback_class.is_feedback_shown == True):
                    #this time.sleep simulated the hand feedback
                    final_position = move_to_goal(x_base,y_base)
                    if final_position:
                        rospy.loginfo("Base reached, searching orientation")
                        orientation_class2 = OrientationReacher(0)
                        orientation_class2.run()
                        if(orientation_class2.is_position_reached):
                            rospy.loginfo("Excelent Performance")
                        else:
                            rospy.loginfo("SOmebody Stole the BASE ID!!!")
                    else:
                        rospy.loginfo(f"Something happened in the way Home from ID {args.station}, go and Check!!")


            else:
                rospy.loginfo(f"ID: {args.station} not at sight, somebody stole it? Returning to BASE.")
                #if Walter tried to go to table/station and believes it got there but cant see and ID, inmeadiatly
                move_to_goal(x_base, y_base)
    else:
        rospy.loginfo("Walter can not reach the desired location")
        if(args.station!=0):
            #If the order was to move to a table and not to the base, walter will return to base
            move_to_goal(x_base,y_base)
else:
    print(f"ID: {args.station} does not exist, did you forget to input the coordinates?.")
    #APAGAR TOODO
rospy.signal_shutdown("Goal reached or failed")
```

APÉNDICE D

Figura D.1

Arquitectura de nodos para Escenario uno (Mapeo)

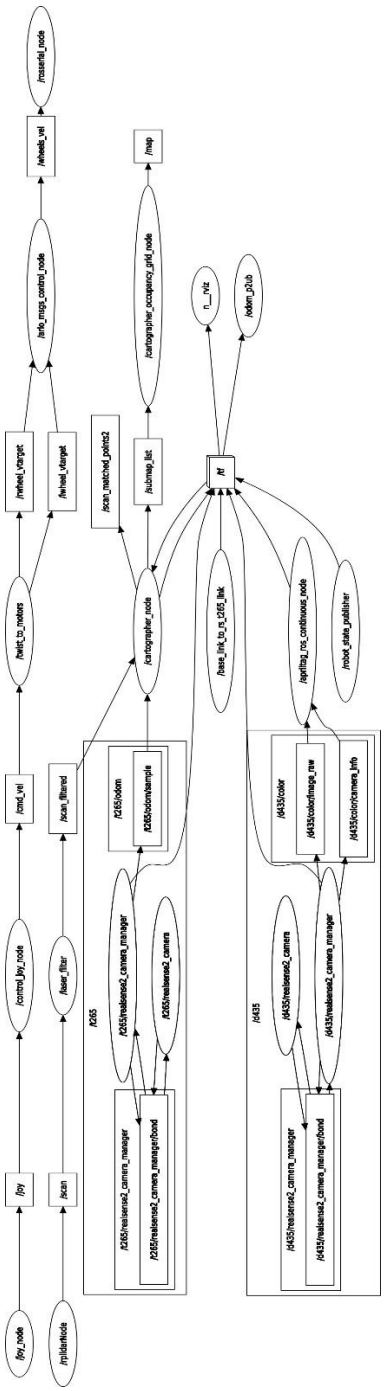


Figura D.2

Arquitectura de nodos para Escenario dos (Navegación y localización sin AMCL)

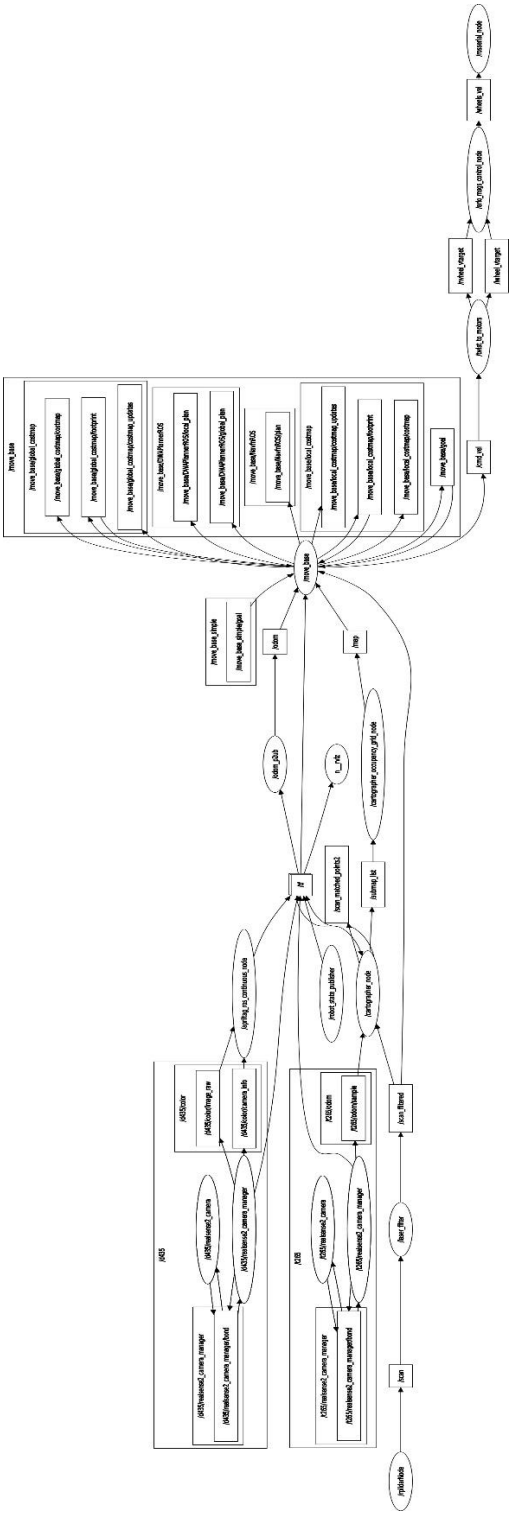


Figura D.3

Arquitectura de nodos para Escenario dos (Navegación y localización con AMCL)

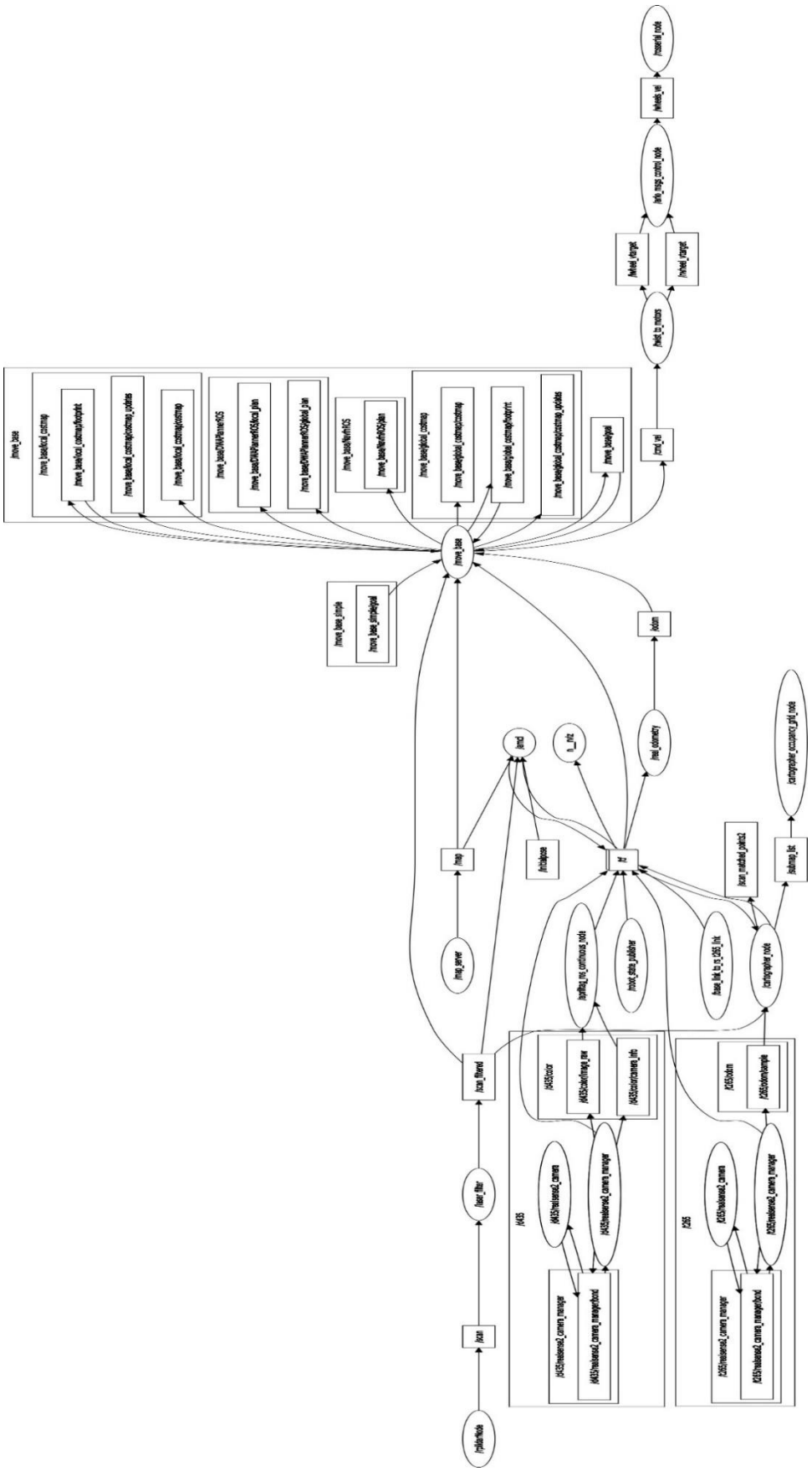


Figura D.4

Árbol de transformadas del robot (inicialización del robot)

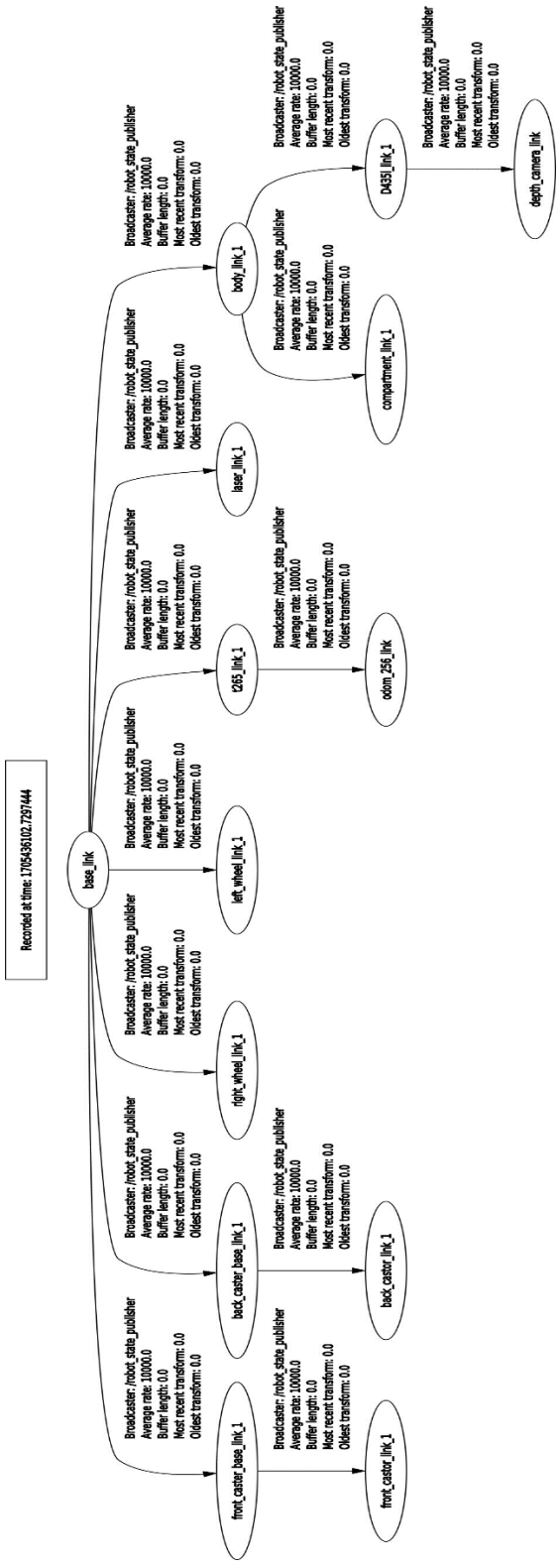
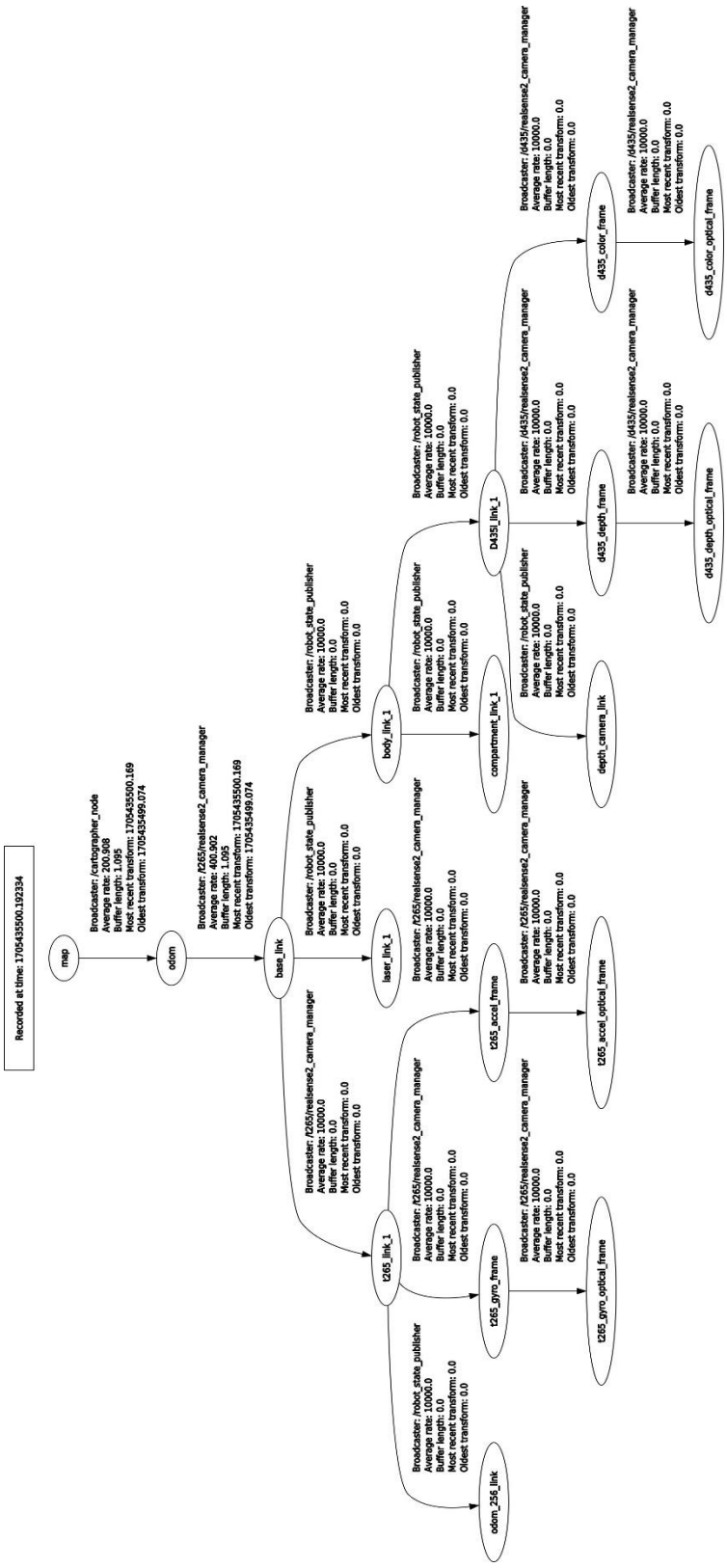


Figura D.5

Árbol de transformadas del robot (modo de navegación)



APÉNDICE E

Cálculos de ticks por metro

$$Ticks\ por\ metro = \frac{ticks\ por\ revolución}{circunferencia\ de\ la\ rueda} \tag{0.1}$$

$$Ticks\ por\ revolución = \ Posición\ del\ encoder * Factor\ de\ cuadratura \tag{0.2}$$

El encoder que se utilizó fue de 36 posiciones 4 cuadraturas:

$$Ticks\ por\ revolución = 36 * 4 = 144\ ti \tag{0.3}$$

$$Ticks\ por\ metro = \frac{144}{\frac{\pi * 15.3}{100}} = 300\ ti/m \tag{0.4}$$

La función que recibe el controlador para manejar las velocidad de las ruedas están en resolución de cantidad de ticks por segundo y se calcula de la siguiente forma:

$$Tasa\ de\ cambio\ de\ la\ rueda = ticks\ por\ metro * velocidad\ de\ la\ rueda \tag{0.5}$$

Cálculos de fuerzas que interactúan en el cuerpo del robot

Se considerará un peso de 550 g estándar para cada paquete que lleva el robot, también se considerará que el robot lleva una cantidad máxima de 3 paquetes, se calculó las fuerzas de la siguiente manera:

$$Peso\ del\ paquete = \ 0.550\ kg * 3 * 9\frac{m}{s^2} = \ 16.17\ N \tag{0.6}$$

$$Fuerza\ horizintal\ del\ paquete = \ 1.65\ kg * 0.9\frac{m}{s^2} = \ 1.485\ N \tag{0.7}$$

Para la inercia se consideró que el paquete posee una forma parecida a un cilindro de 20 cm.

$$Inercia\ del\ paquete = \frac{1}{2} * 1.65\ kg * (0.20)^2 = 0.033\ kg * m^2 \quad (0.8)$$

Cálculos de las fuerzas que interactúan en el eje del motor

Para empezar el análisis, se sumó todas las masas de los componentes del robot, finalmente se obtuvo un valor de 10.25 kg para todos los componentes del robot y 1.3 kg para cada rueda. De estas masas se calculó los pesos respectivos.

$$Peso\ del\ cuerpo\ = \ 10.25\ kg * 9.8\ \frac{m}{s^2} = 100.45\ N \quad (0.9)$$

$$Peso\ de\ las\ dos\ ruedas\ = \ 1.3\ kg * 9.8\ \frac{m}{s^2} = 25.48\ N \quad (0.10)$$

Para colocar las fuerzas en Inventor se distribuyó equitativamente en cada eje, finalmente en cada eje existen dos fuerzas una de 12.74 N y 50.23 N.

Cálculos de las velocidades máximas para el análisis dinámico

Se tomó la distancia entre centro y ruedas igual a 19.90 cm, se calcularon las velocidades utilizando las velocidades angulares críticas de 0.2 rad/s y 0.9 rad/s.

$$Velocidad\ 1 = \ 0.2 \frac{rad}{s} * 0.1984\ m = 0.03967 \frac{m}{s} \quad (0.11)$$

$$Velocidad\ 2 = \ 0.9 \frac{rad}{s} * 0.1984\ m = 0.17854 \frac{m}{s} \quad (0.12)$$

Considerando el radio de la rueda igual a 0.0762 m se procede a calcular la velocidad angular de la rueda en cada caso.

$$Velocidad\ angular\ 1 = \frac{0.03967}{0.0762} = 0.5206 \frac{rad}{s} \tag{0.13}$$

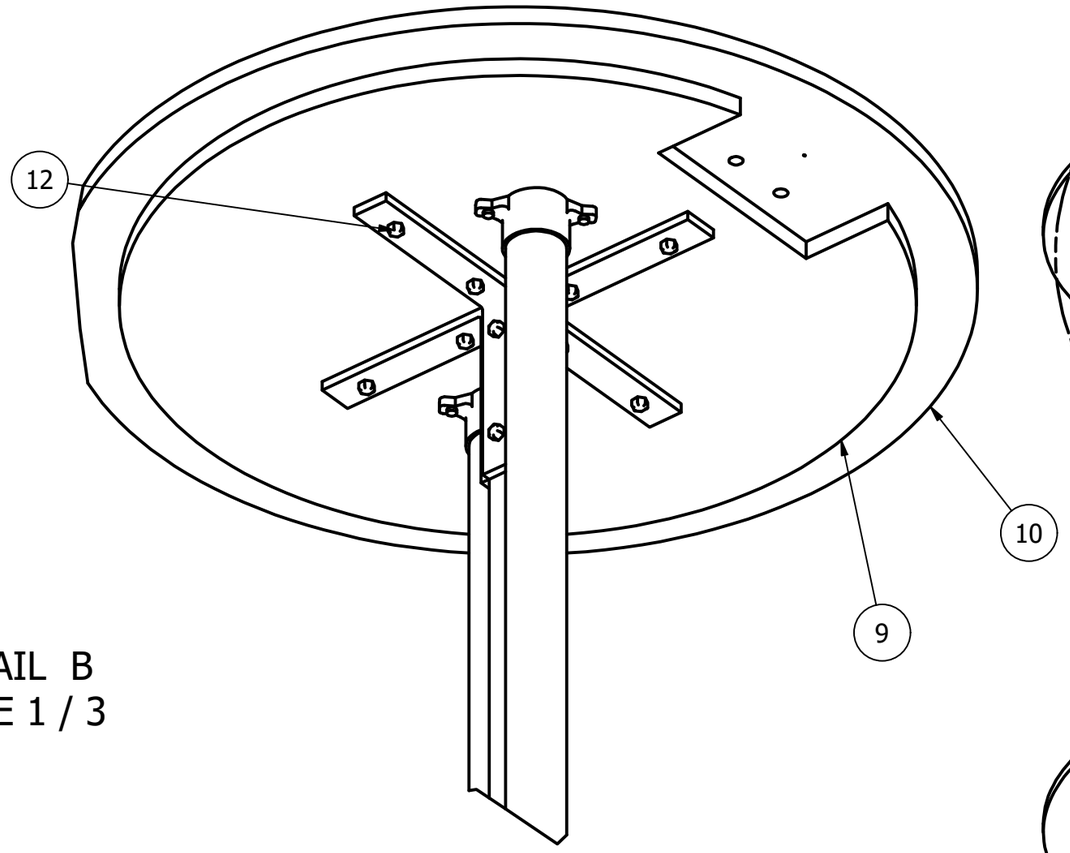
$$Velocidad\ angular\ 2 = \frac{0.17854}{0.0762} = 2.343 \frac{rad}{s} \tag{0.14}$$

APÉNDICE F

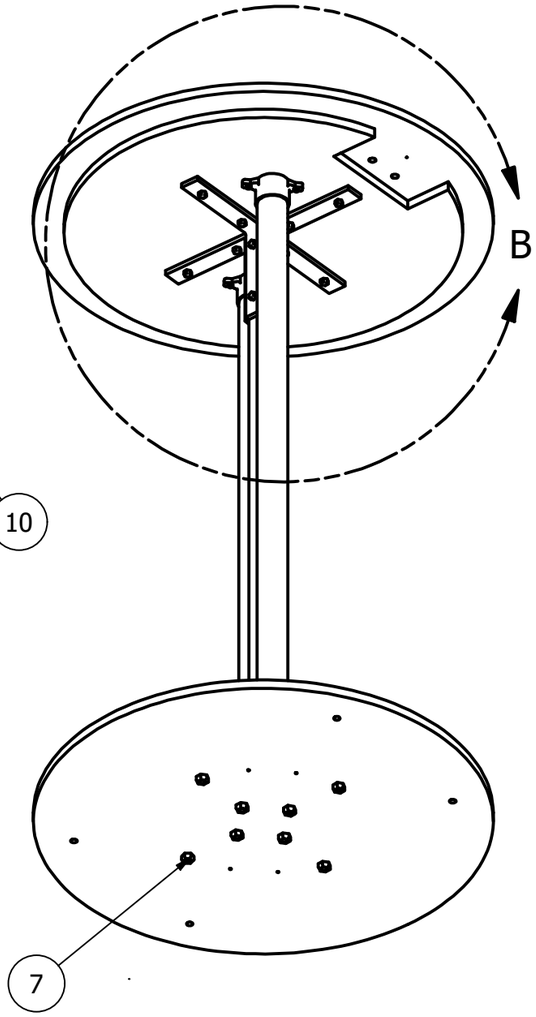
Planos de las piezas del proyecto

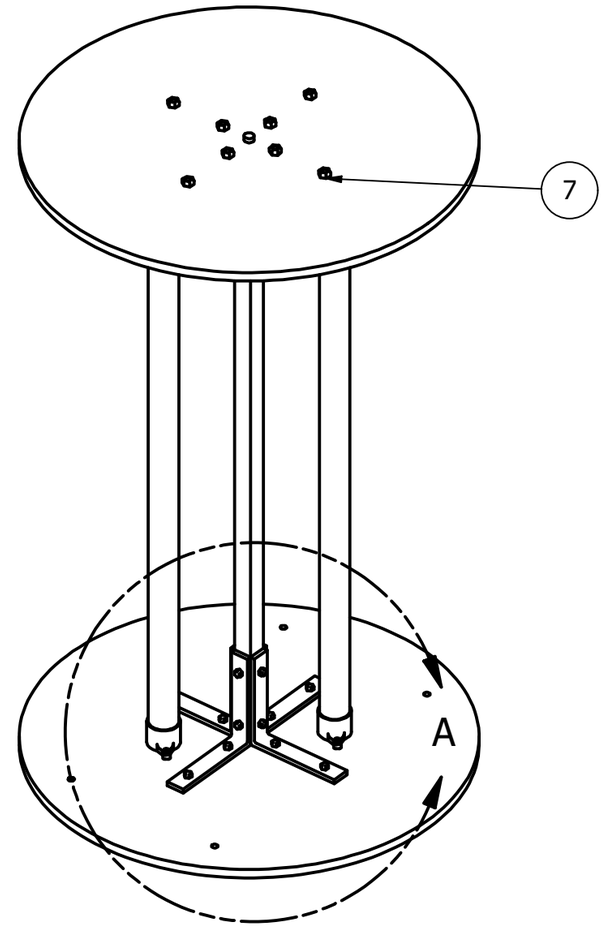
Los planos se muestran en el siguiente orden:

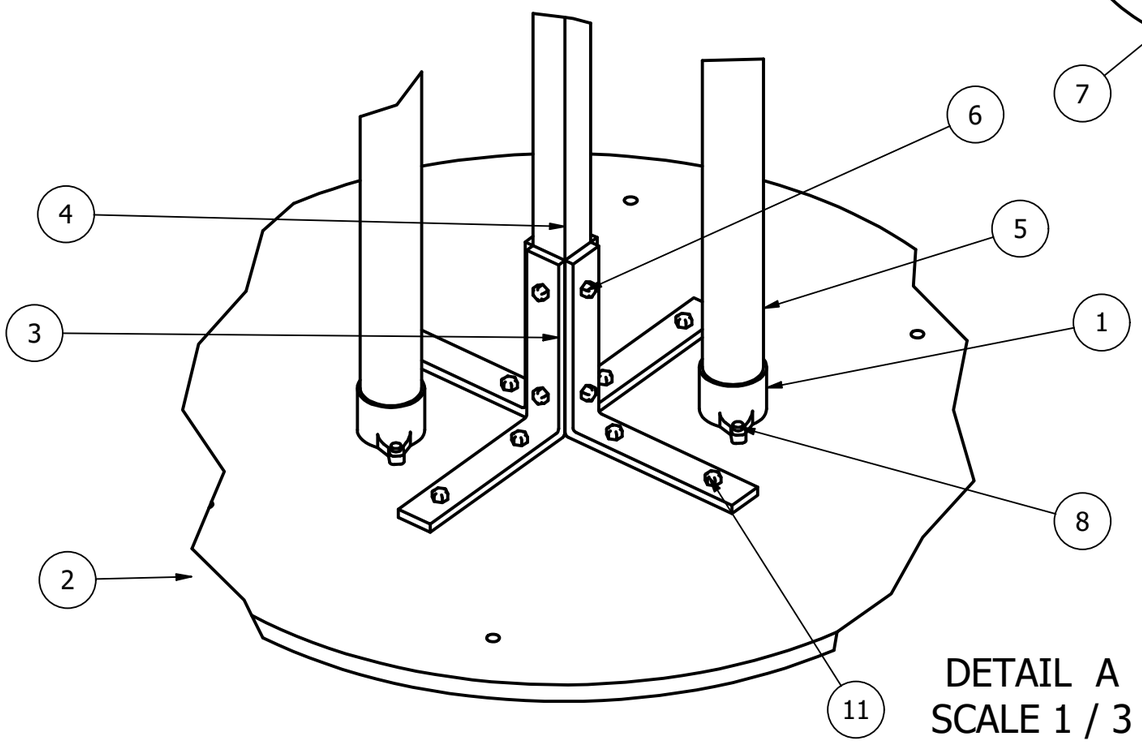
- PLANO 1. Plano del ensamble del cuerpo del robot
- PLANO 2. Plano de la vista explosionada del cuerpo del robot
- PLANO 3. Plano de la base media del cuerpo del robot
- PLANO 4. Plano de la base inferior del cuerpo del robot
- PLANO 5. Plano de la base superior del cuerpo del robot
- PLANO 6. Plano del eje lateral del cuerpo del robot
- PLANO 7. Plano del eje central del cuerpo del robot
- PLANO 8. Plano del soporte del eje central del cuerpo del robot
- PLANO 9. Plano del soporte del eje lateral del cuerpo del robot
- PLANO 10. Plano del compartimiento superior del cuerpo del robot
- PLANO 11. Plano del soporte de la cámara T265



DETAIL B
SCALE 1 / 3







DETAIL A
SCALE 1 / 3

12	perno de la base superior	PZ # 12	Acero ASTM 36	11	M6x1x30mm	5 g	
11	perno de la base inferior	PZ # 11	Acero ASTM 36	10	M6x1x15mm	5 g	
10	base superior del robot	PZ # 10	Plywood	7	--	500 g	
9	base media del robot	PZ # 9	Plywood	6	--	400 g	
8	tornillo del soporte lateral	PZ # 8	Acero ASTM 36	8	--	4 g	
7	tuerca	PZ # 7	Acero ASTM 36	12	6mm	3 g	
6	perno del eje central	PZ # 6	Acero ASTM 36	9	M6x1x8mm	5 g	
5	soporte lateral	PZ # 5	Aluminio 6061	2	--	1000 g	
4	eje central	PZ # 4	Aluminio 6061	1	--	600 g	
3	soporte para eje central	PZ # 3	Acero AISI 304	3	--	1500 g	
2	base inferior	PZ # 2	Plywood	5	--	500 g	
1	soporte para ejes laterales	PZ # 1	PLA	4	--	20 g	
Nº DE PIEZA	DENOMINACIÓN	No. DE NORMA/DIBUJO	MATERIAL	Nº DE ORDEN	No. DEL MODELO/SEMIPRODUCTO	PESO kg/PIEZA	OBSERVACIONES

EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE

TOLERANCIAS
(1) NO SE INDICA LO CONTRARIO
LAS COTAS SE EXPRESAN EN MILIMETROS
ACABADO SUPERFICIAL N7
TOLERANCIAS:
LINEAL: ±1
ANGULAR: ±1

PESO
4,54 kg

MATERIALES
Ver lista de materiales

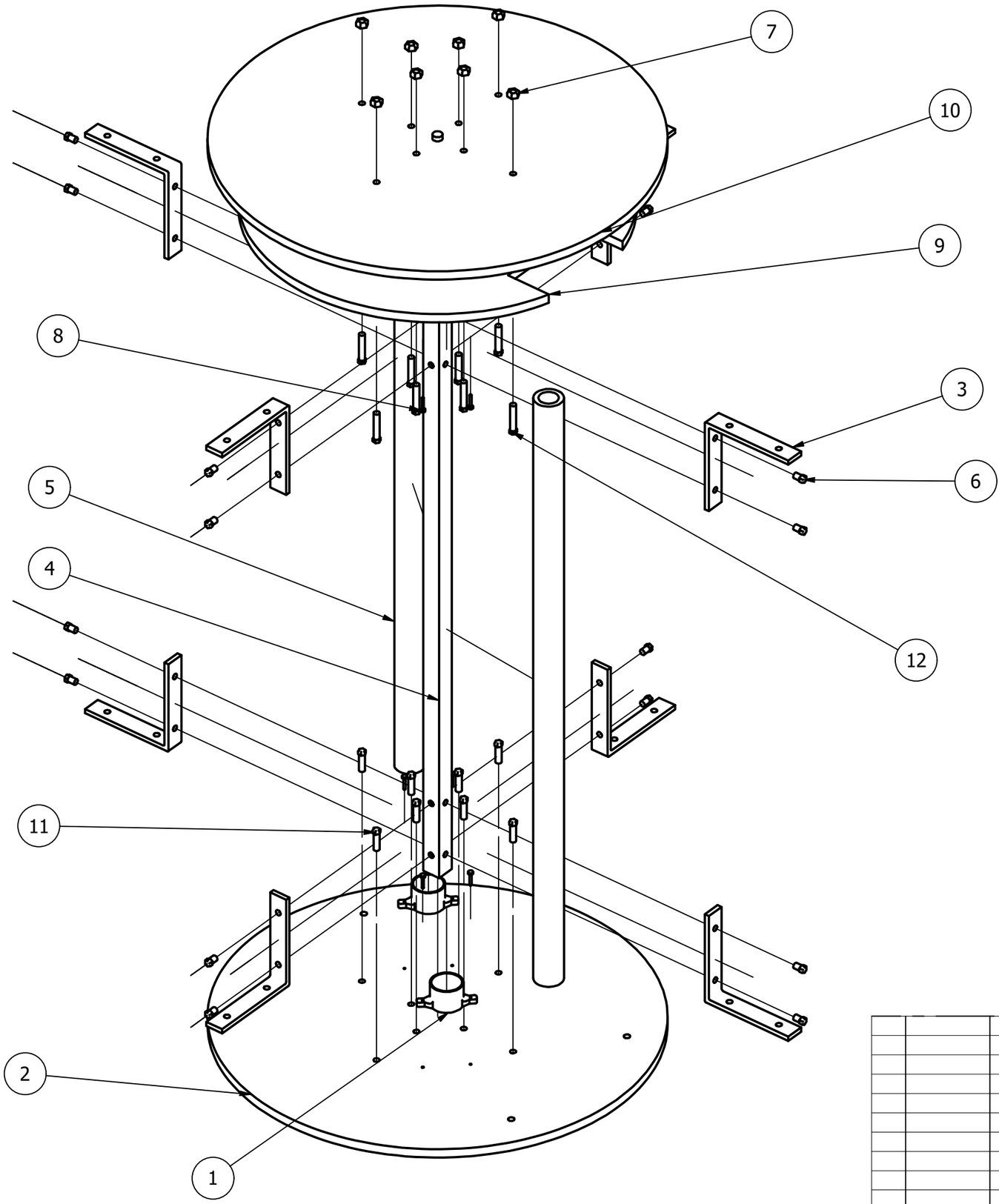
	FECHA	NOMBRE
DIB.	24/01/2024	valladarez/Párraga.
REV.		
APROB.		

DENOMINACIÓN
Cuerpo del robot autónomo

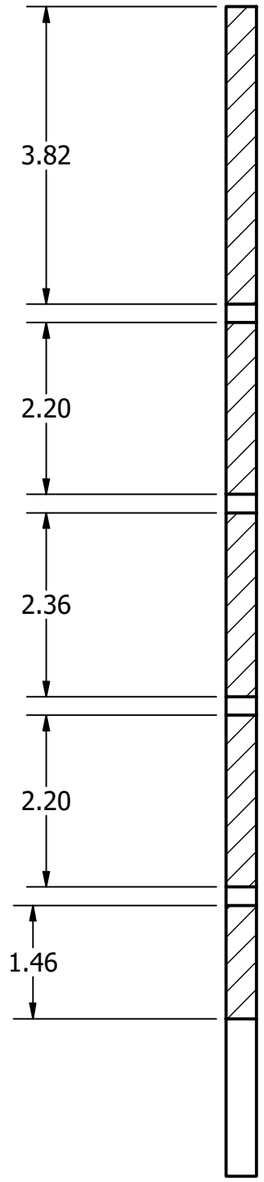
ESCALA
1:6

NÚMERO DEL DIBUJO
MCT-24-01

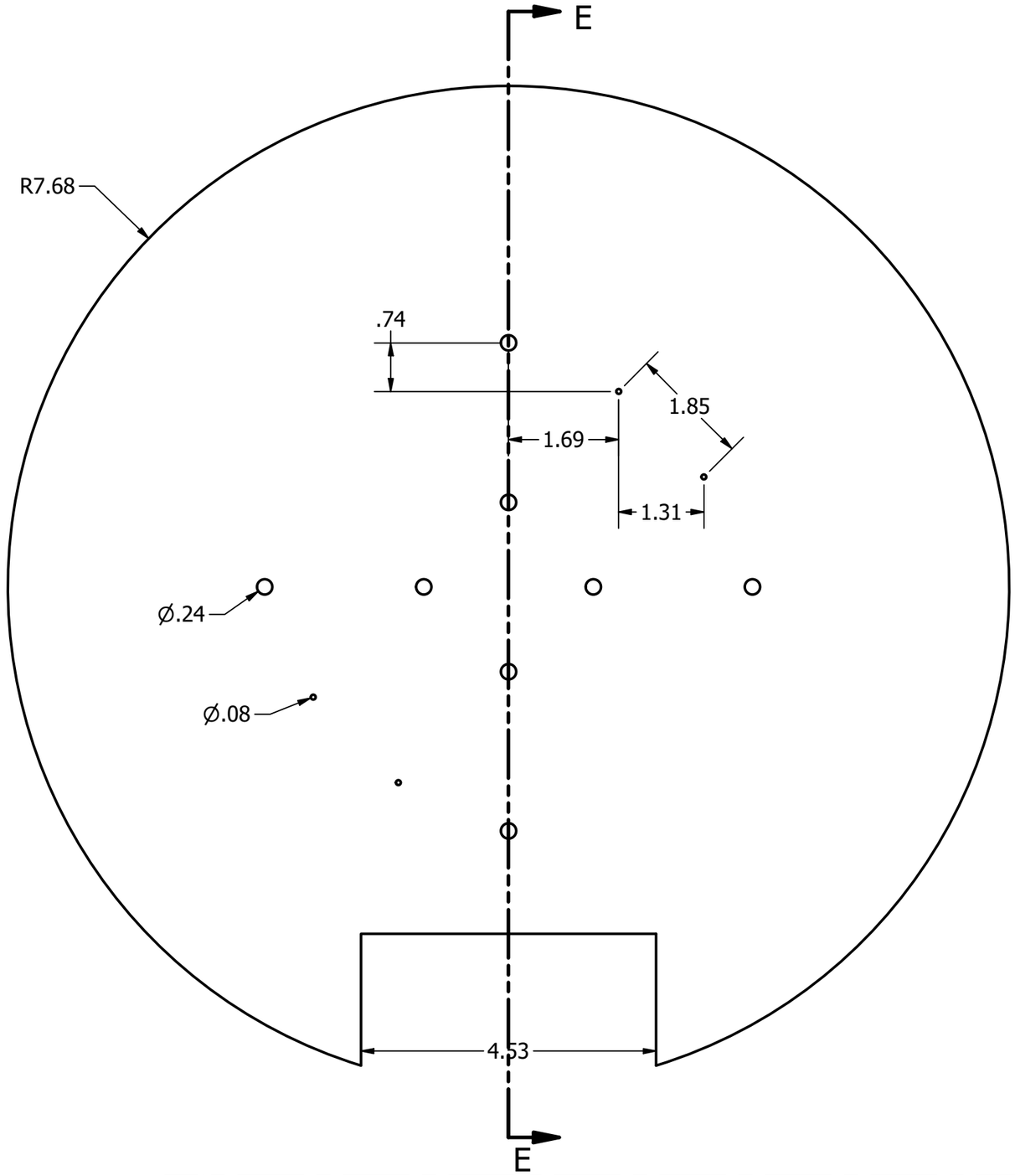
SUSTITUYE A:



				TOLERANCIAS SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS LINEAL: ±1 ANGULAR: ±1	PESO 4,54 kg	MATERIALES Ver lista de materiales	
					FECHA	NOMBRE	DENOMINACIÓN Cuerpo del robot autónomo vista explosionada	ESCALA 0,45:2
				DIB.	24/01/2024	valladarez/Párraga		
				REV.				
				APROB.			NÚMERO DEL DIBUJO MCT-24-11	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:	

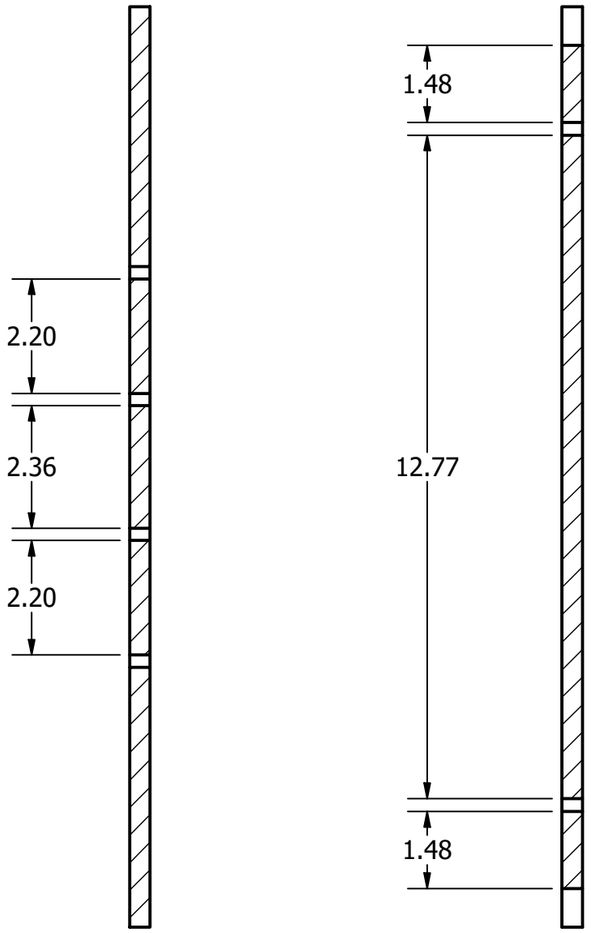


SECTION E-E
SCALE 1 / 2



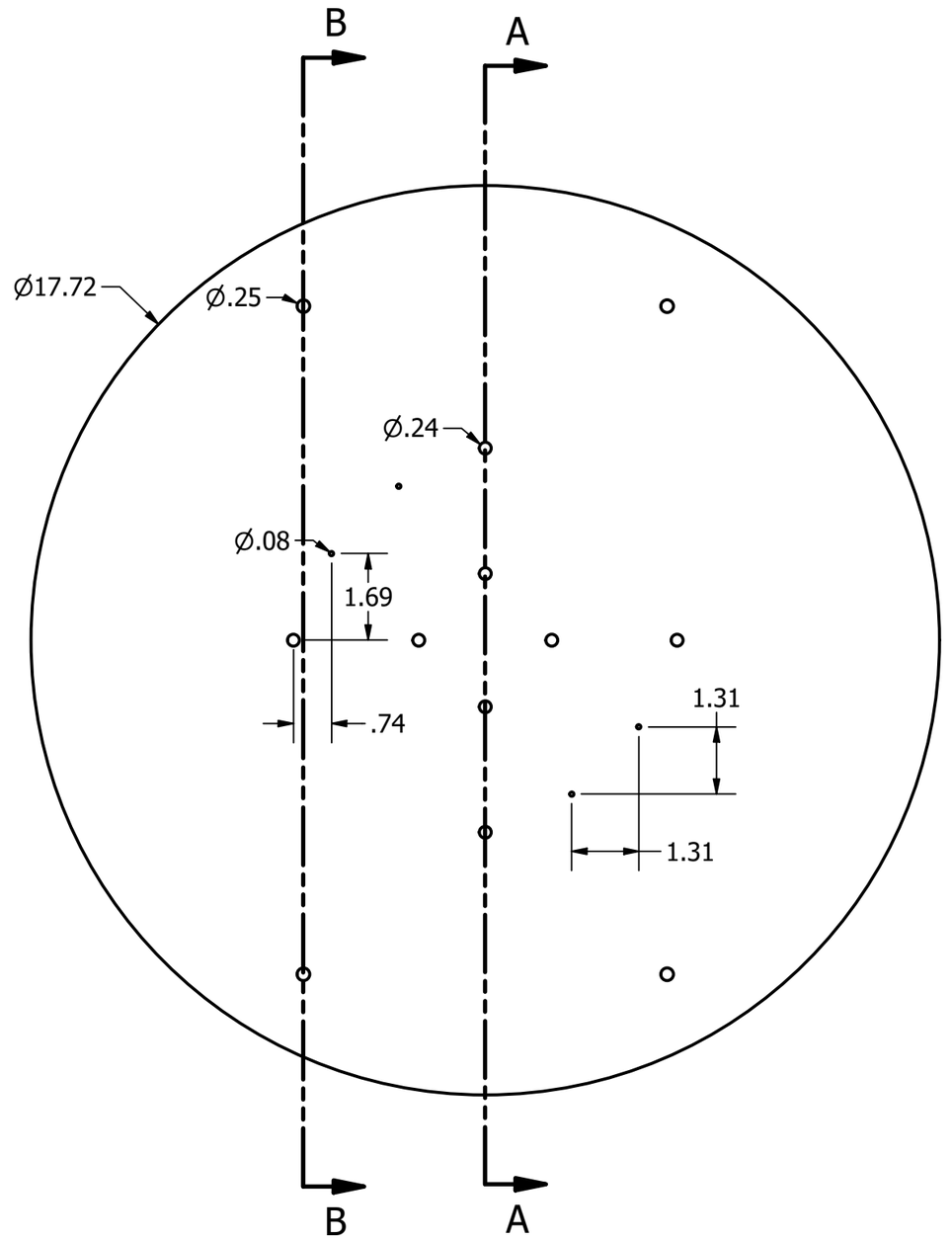
				TOLERANCIAS SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS LINEAL: ±1 ANGULAR: ±1	PESO 0,4 kg	MATERIALES Plywood	
					FECHA	NOMBRE	DENOMINACIÓN Base media del cuerpo del robot
				DIB.	24/01/2024	valladarez/Párraga	
				REV.			
				APROB.			
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				NÚMERO DEL DIBUJO MCT-24-03
							SUSTITUYE A:

ESCALA
1:2

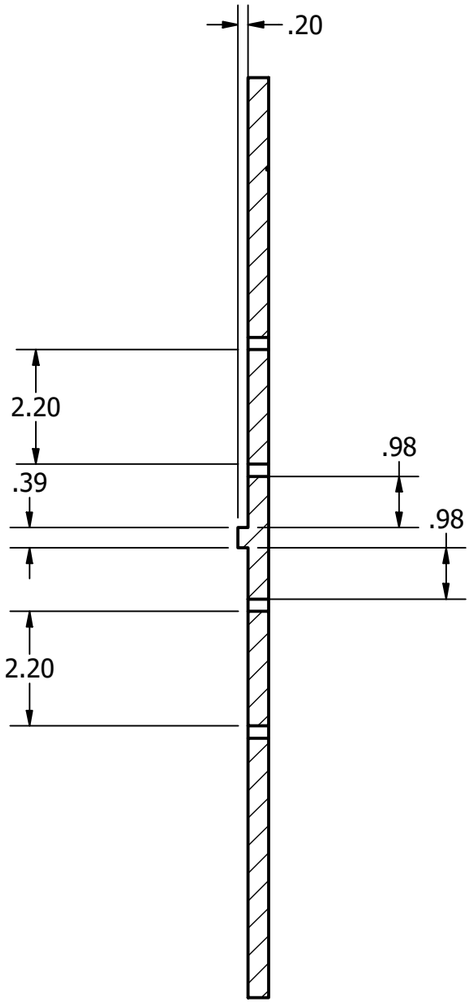


SECTION A-A
SCALE 1 / 3

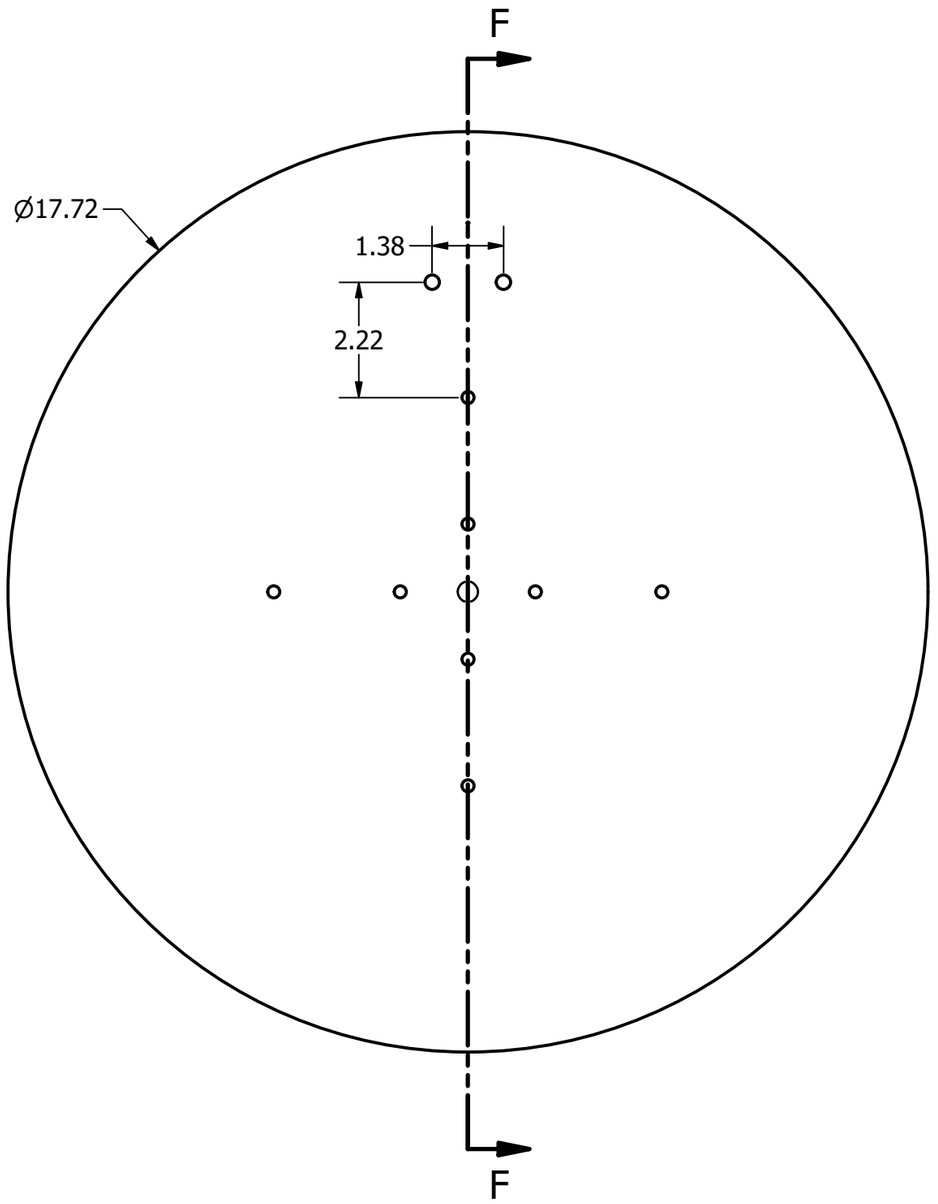
SECTION B-B
SCALE 1 / 3



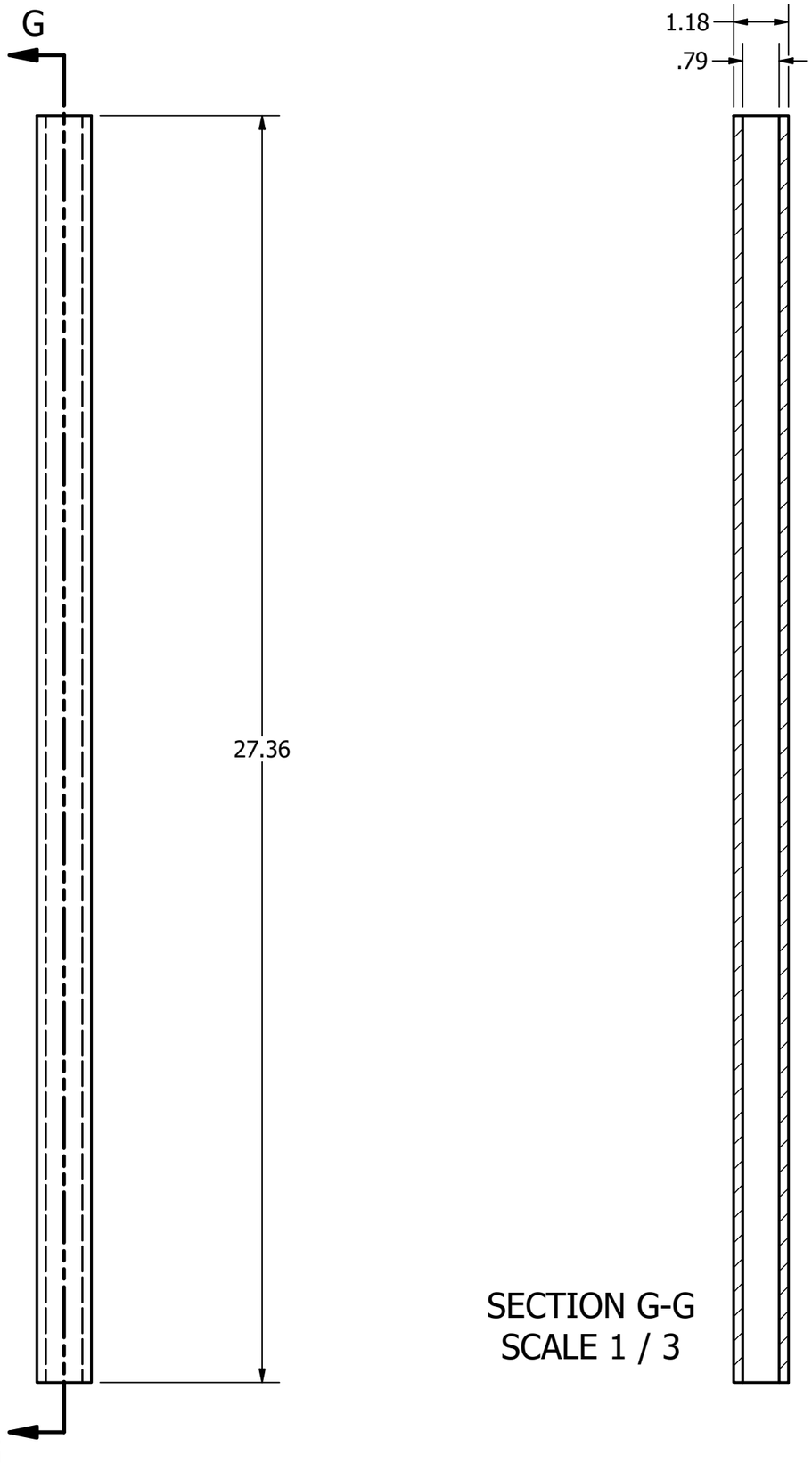
				TOLERANCIAS SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: ±1 ANGULAR: ±1	PESO 0,5 kg	MATERIALES Plywood	
					FECHA	NOMBRE	DENOMINACIÓN Base inferior del robot	ESCALA 1:3
				DIB.	24/01/2024	valladarez/Párraga		
				REV.				
				APROB.			NÚMERO DEL DIBUJO MCT-24-02	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:	



SECTION F-F
SCALE 1 / 3

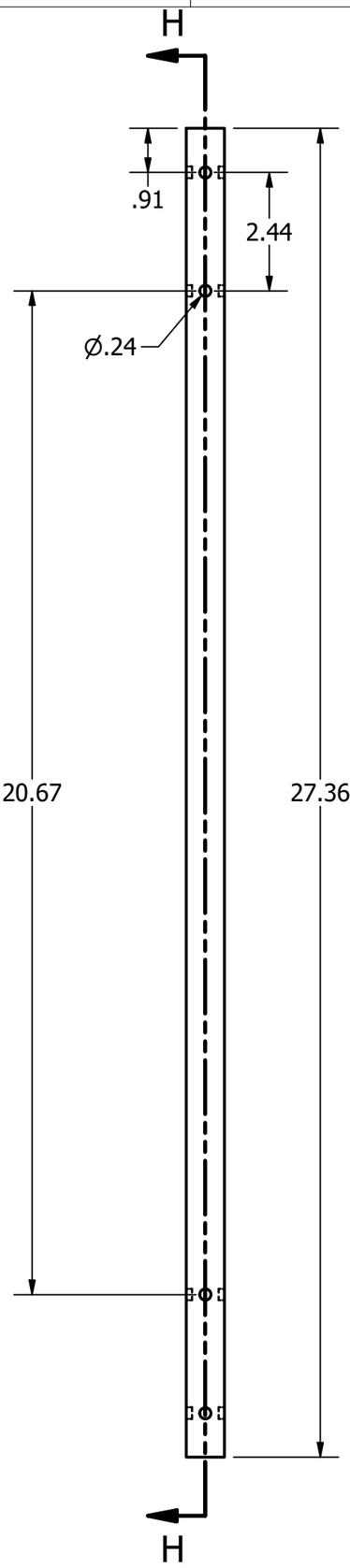


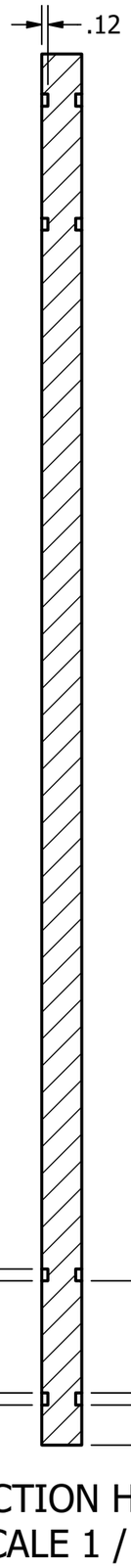
				TOLERANCIAS SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL N7 TOLERANCIAS LINEAL: ±1 ANGULAR ±1	PESO 0,5 kg	MATERIALES Plywood	
					FECHA	NOMBRE	DENOMINACIÓN Base superior del cuerpo del robot	ESCALA 1:3
				DIB.	24/01/2024	valladarez/Párraga		
				REV.				
				APROB.				
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				NÚMERO DEL DIBUJO MCT-24-04	
							SUSTITUYE A:	



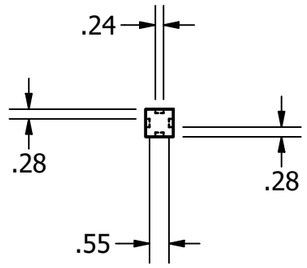
SECTION G-G
SCALE 1 / 3

				TOLERANCIAS: SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS LINEAL: ±1 ANGULAR: ±1	PESO: 0,5 kg	MATERIALES: Aluminio 6061	
					FECHA	NOMBRE	DENOMINACIÓN: soportes laterales del cuerpo del robot	ESCALA: 1:3
				DIB.	24/01/2024	valladarez/Párraga		
				REV.				
				APROB.			NÚMERO DEL DIBUJO: MCT-24-05	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:	



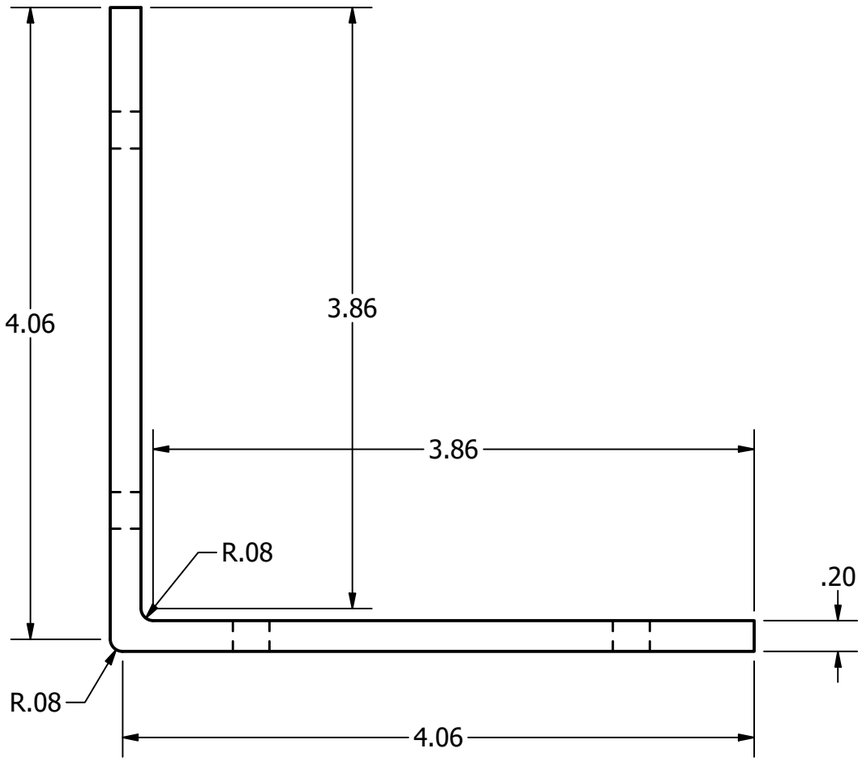


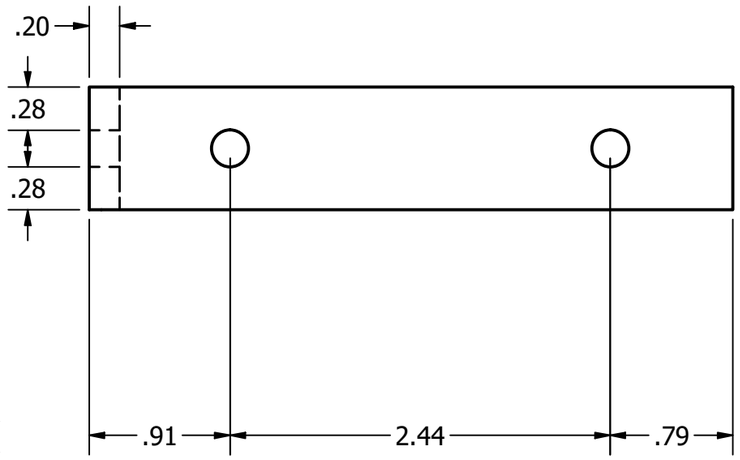
SECTION H-H
SCALE 1 / 3

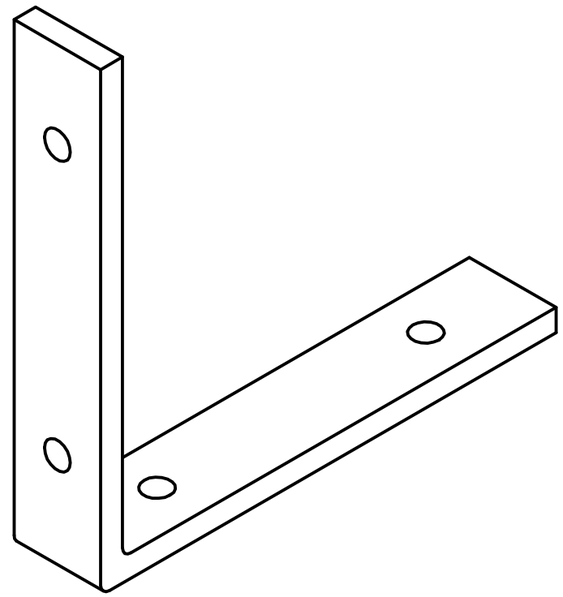


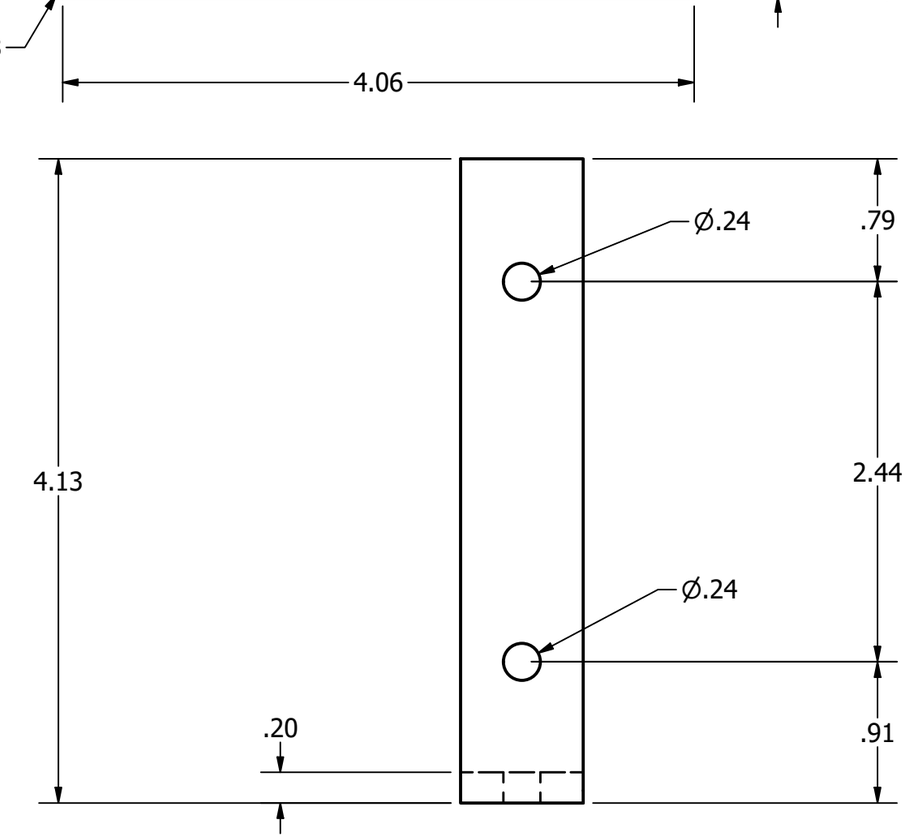
				TOLERANCIAS: SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1	PESO: 0,6 kg	MATERIALES: Aluminio 6061	
					FECHA	NOMBRE	DENOMINACIÓN: Soporte principal del cuerpo del robot
				DIB.	24/01/2024	valladarez/Párraga	
				REV.			
				APROB.			NÚMERO DEL DIBUJO: MCT-24-06
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

ESCALA: 1:3

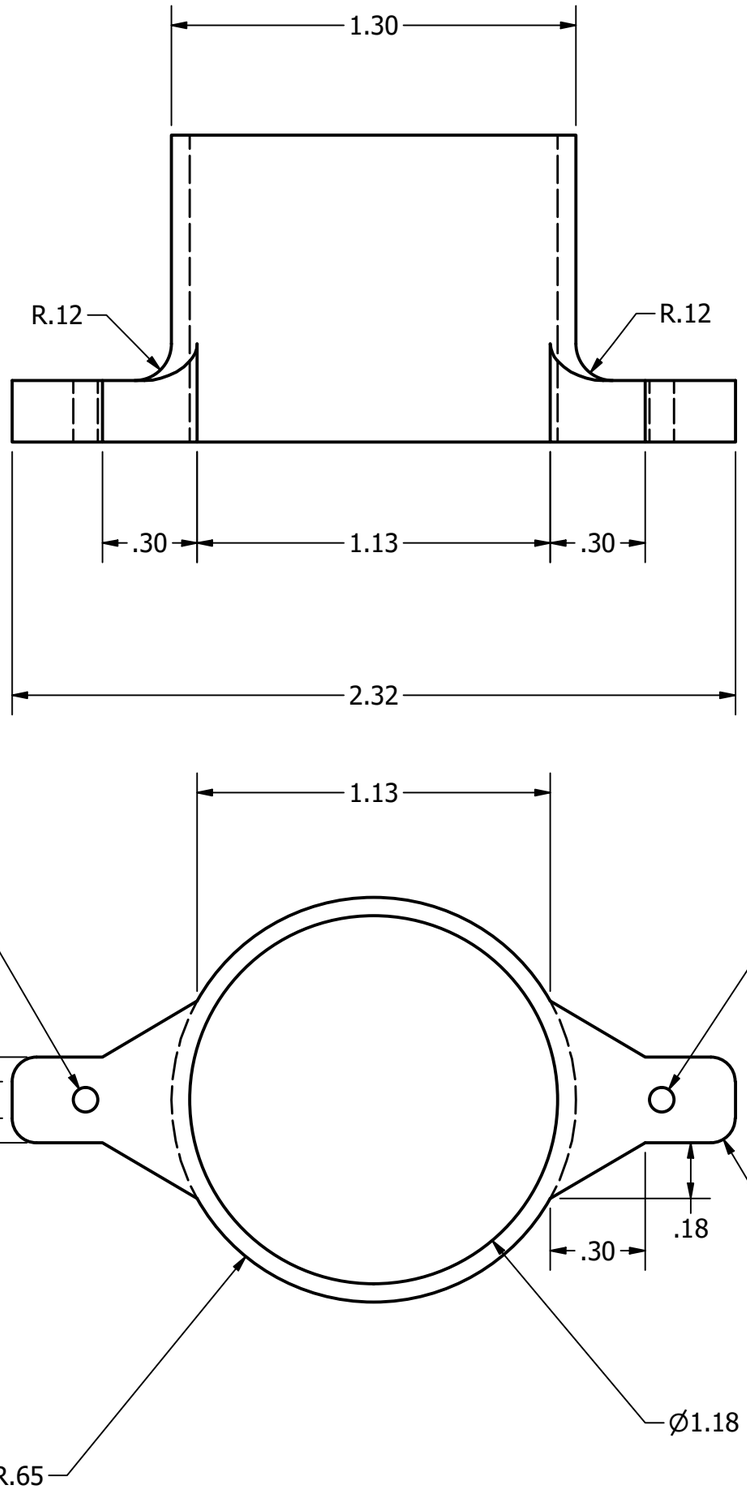


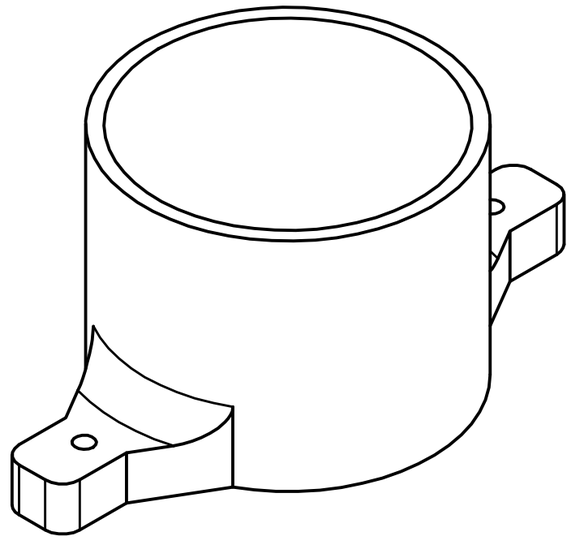




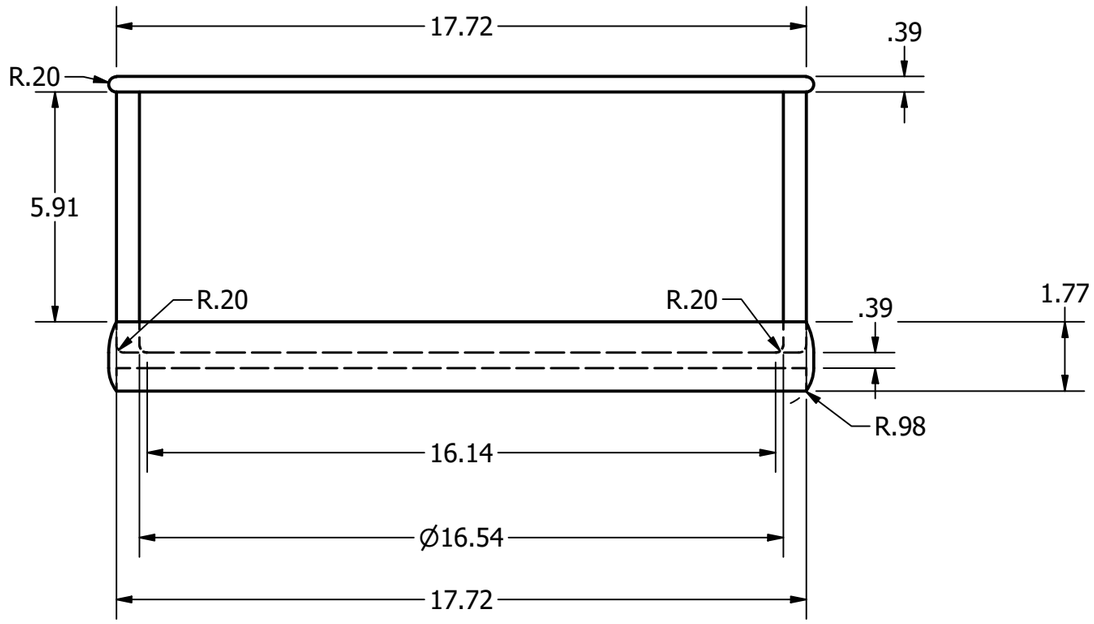


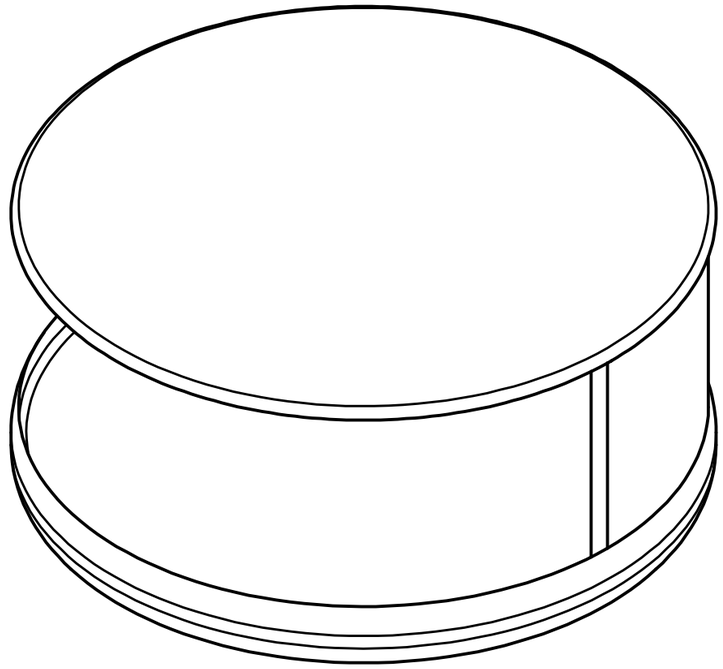
				TOLERANCIAS: SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS LINEAL: ±1 ANGULAR: ±1	PESO: 0,188 kg	MATERIALES: Acero AISI 304	
					FECHA	NOMBRE	DENOMINACIÓN: Soportes para el eje central del cuerpo del robot
				DIB.	24/01/2024	valladarez/Párraga	
				REV.			ESCALA: 1:1
				APROB.			NÚMERO DEL DIBUJO: MCT-24-07
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

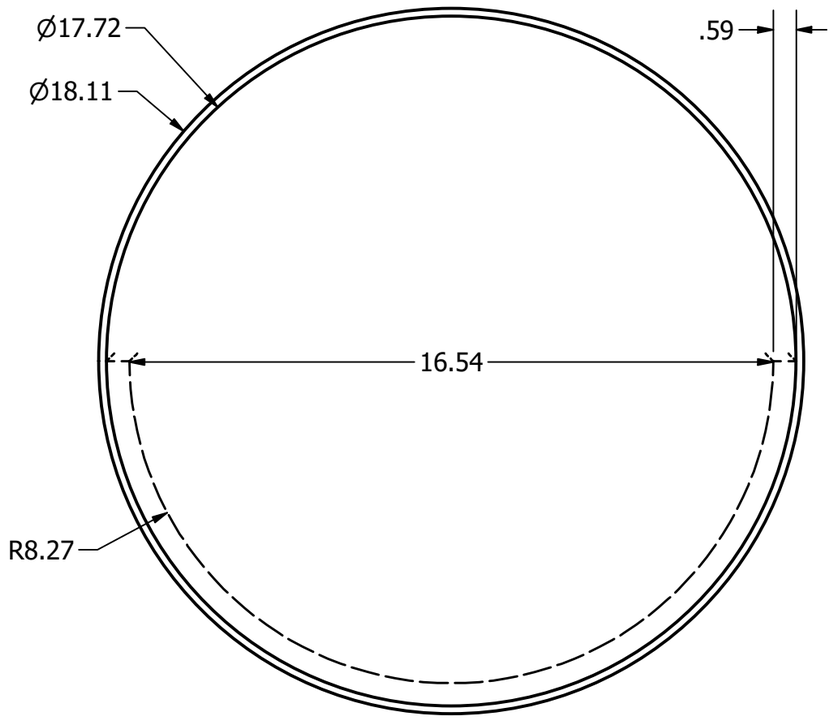




				TOLERANCIAS SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS LINEAL: ±1 ANGULAR: ±1	PESO 0,010 kg	MATERIALES PLA	
					FECHA	NOMBRE	DENOMINACIÓN Soportes para el eje lateral del cuerpo del robot	ESCALA 2:1
				DIB.	24/01/2024	valladarez/Párraga		
				REV.				
				APROB.				
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				NÚMERO DEL DIBUJO MCT-24-08	
							SUSTITUYE A:	

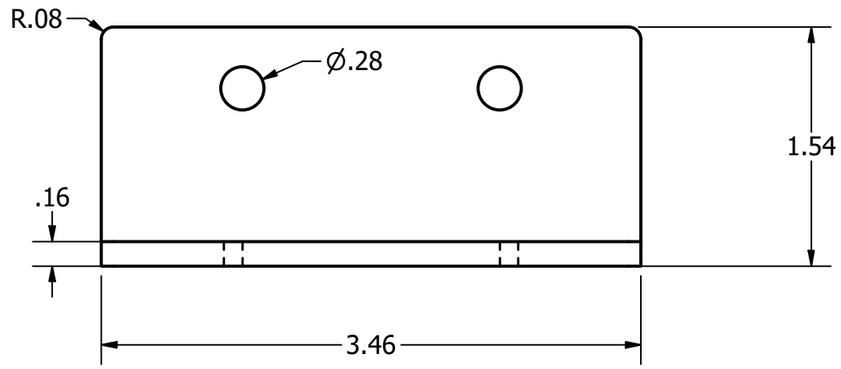


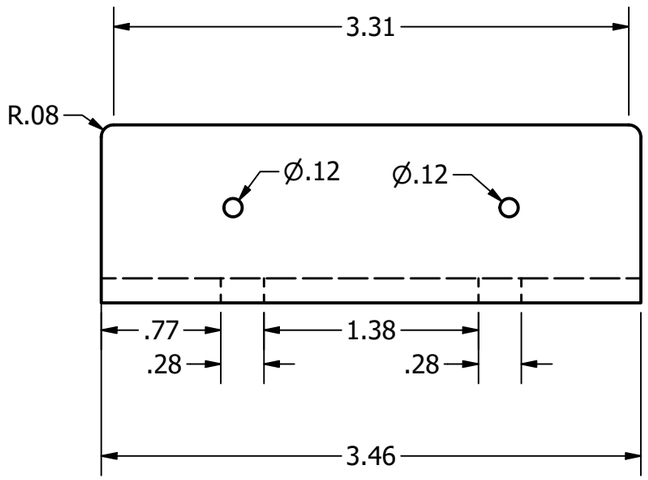


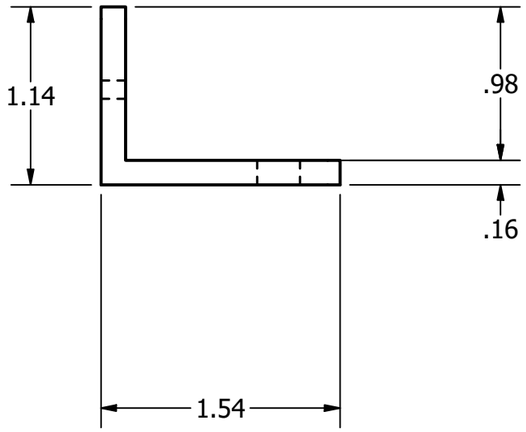


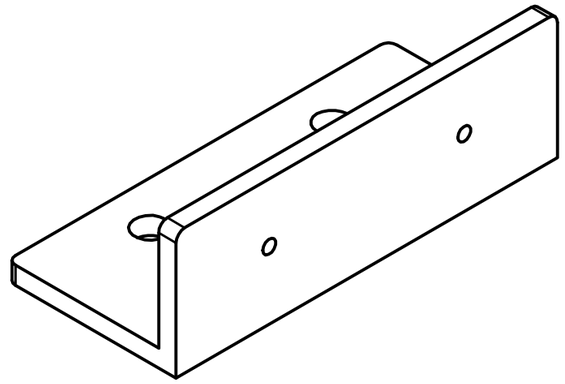
				TOLERANCIAS: SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: ±1 ANGULAR: ±1	PESO: 2 kg	MATERIALES: PLA	
					FECHA	NOMBRE	DENOMINACIÓN: compartimiento superior del cuerpo del robot
				DIB.	24/01/2024	valladarez/Párraga	
				REV.			
				APROB.			
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				NÚMERO DEL DIBUJO: MCT-24-09
							SUSTITUYE A:

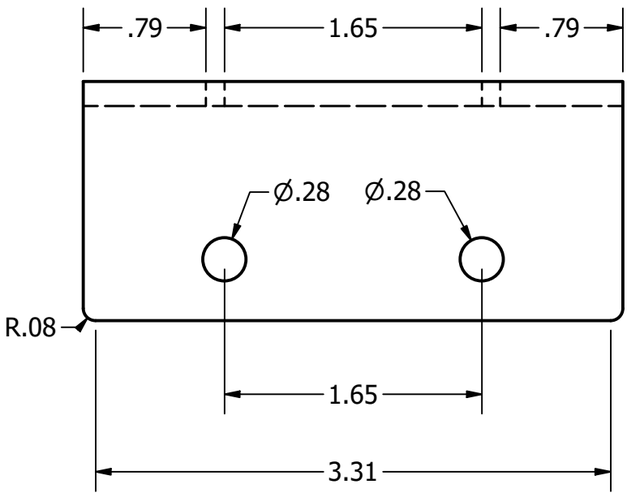
ESCALA: 1:4











				TOLERANCIAS SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N? TOLERANCIAS LINEAL: ?1 ANGULAR: ?1	PESO 0,050 kg	MATERIALES PLA	
					FECHA	NOMBRE	DENOMINACIÓN Soporte para cámara D435i	ESCALA 1:1
				DIB.	24/01/2024	valladarez/Párraga		
				REV.				
				APROB.				
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				NÚMERO DEL DIBUJO MCT-24-10	
							SUSTITUYE A:	