ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

DISEÑO E IMPLEMENTACIÓN DE ROBOT MÓVIL AUTÓNOMO DE DESINFECCIÓN PARA NAVEGACIÓN SOCIAL EN ENTORNOS CERRADOS Y DINÁMICOS

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero en Mecatrónica

Presentado por:

Steven Alexander Silva Mendoza

GUAYAQUIL - ECUADOR

Año: 2021

DEDICATORIA

Este trabajo se lo dedico principalmente a mis padres y a quienes creyeron en mi para llegar hasta este momento.

AGRADECIMIENTO

Agradezco principalmente a mis padres por apoyarme en todos mis objetivos, por darme una excelente educación y por siempre creer en mí. También agradezco principalmente a Dennys Paillacho, José Laica, Jhon Merchán y Katerin Alarcón por haberme apoyado y por acompañarme en todos mis proyectos. Finalmente, agradezco a todos esos profesores dedicados que conocí en toda mi trayectoria educativa por los conocimientos y formación otorgada.

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; Steven Alexander Silva Mendoza y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Steven Alexander Silva Mendoza

EVALUADORES

BRYAN JOAO PURUNCAJAS MAZA

Firmado digitalmente por BRYAN JOAO PURUNCAJAS MAZA
Fecha: 2021.11.05 08:30:35 -05'00'

Bryan Puruncajas

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
DENNYS FABIAN PAILLACHO CHILUIZA

Dennys Paillacho

PROFESOR TUTOR

RESUMEN

Actualmente, empresas están regresando a actividades presenciales en espacios cerrados. Esto aumenta el riesgo de contagio del virus COVID-19. Se conoce que la principal vía de contagio es aerosol. Los métodos manuales de desinfección no son capaces de desinfectar el ambiente de lugares cerrados y la luz UV-C no puede ser usada por operarios. Nuestro objetivo es la implementación de un robot autónomo móvil de desinfección por luz UV-C totalmente autónomo.

Se implementó una arquitectura electrónica y software integrando sensores como lidar y cámara de profundidad. Se otorgó la capacidad de realizar acciones de Simultaneous Localization and Mapping (SLAM). Además se incluyeron distintos algoritmos de navegación basados en algoritmos reactivos y modelos. También se hizo el uso de controladores para el manejo de motores DC.

Se logró realizar simulaciones del robot obteniendo varios resultados respecto a sus capacidades y el control de movimiento tanto para navegación como acción de auto-recarga. La estación de recarga pudo ser implementada usando impresión 3D. Asimismo se construyó un prototipo con el cual se comprobó su funcionalidad en pruebas experimentales en las que se realizó mapeo, localización y navegación.

En conclusión, se diseñó robot autónomo capaz de desinfectar espacios sin supervisión humana. Este se mueve de manera satisfactoria y se ajusta a los requerimientos de peso y autonomía. Se asegura que el robot no tendrá fallos en su funcionamiento y que es capaz de navegar en espacios sin colisiones con objetos o personas mientras mantiene el confort de los humanos.

Palabras Clave: Robots Autónomos, COVID-19, Luz UV-C.

ABSTRACT

Currently, companies are returning to face-to-face activities in closed spaces. This increases the risk of contagion of the COVID-19 virus. It is known that the main route of infection is aerosol. Manual disinfection methods are not capable of disinfecting the indoor environment and UV-C light cannot be used by operators. Our goal is the implementation of a fully autonomous mobile autonomous UV-C light disinfection robot.

Currently, companies are returning to face-to-face activities in closed spaces. This increases the risk of contagion of the COVID-19 virus. It is known that the main route of infection is aerosol. Manual disinfection methods are not capable of disinfecting the indoor environment and UV-C light cannot be used by operators. Our goal is the implementation of a fully autonomous mobile autonomous UV-C light disinfection robot.

An electronic architecture and software were implemented integrating sensors such as lidar and depth camera. The ability to perform SLAM actions was granted. In addition, different navigation algorithms based on reactive algorithms and models were included. Controllers were also used to drive DC motors.

It was possible to carry out simulations of the robot obtaining several results regarding its capabilities and movement control for both navigation and auto-recharge action. The charging station could be implemented using 3D printing. Likewise, a prototype was built with which its functionality was verified in experimental tests in which mapping, location and navigation were carried out.

In conclusion, an autonomous robot capable of disinfecting spaces without human supervision was designed. This moves satisfactorily and adjusts to the weight and autonomy requirements. It is ensured that the robot will not have malfunctions in its operation and that it is capable of navigating in spaces without collisions with objects or people while maintaining the comfort of humans.

Keywords: Autonomous Robots, COVID-19, UV-C Light.

ÍNDICE GENERAL

ABREVIATURAS

ROS	Robot Operating System
SFM	Social Force Model
SLAM	Simultaneous Localization and Mapping
VSLAM	Visual Simultaneous Localization and Mapping
AMCL	Adaptive Monte-Carlo Localizer
RMA	Robot Móvil Autónomo
DWA	Dynamic Window Approach
TEB	Timed Elastic Band
IR	infrarrojo
IMU	Inertial Measurement Unit
PWM	Pulso De Ancho Modulado
YOLO	You Only Look Once
MQTT	MQ Telemetry Transport
URDF	Unified Robotic Description Format
PLA	Ácido Poliláctico
UART	Universal Asynchronous Receiver-Transmitter
CIDIS	Centro de Investigación, Desarrollo e Innovación de Sistemas Computacionales
FEA	Finite Element Analysis
SII	Social Individual Index
UV	ultravioleta

SIMBOLOGÍA

mm	milímetro
cm	centímetro
V	Voltios
A	amperios
m	metro
psi	libras por pulgada cuadrada
MPa	MegaPascal
g	gramos
kg	kilogramo
° C	grado Celsius
Hz	Hertzios
lb	libra

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE PLANOS

CAPÍTULO 1

1. INTRODUCCIÓN

Hasta inicios del 2021, el virus SARS-CoV-2 sigue cobrando vidas debido a su letalidad y capacidad de contagio. Ha causado grandes pérdidas en muchos países de todo el mundo a causa de las cuarentenas implementadas y a las medidas de prevención. En 2020 causó hasta una pérdida del 6 % del producto interno bruto para países como Argentina y el Reino Unido [1]. Se ha comprobado que este virus puede ser esparcido a través de objetos y contraído en la compartición de espacios de trabajo, principalmente en entornos cerrados como oficinas y lugares de atención médica [2]. Para inicios del 2021, varias empresas han comenzado a regresar a la modalidad de trabajo presencial.

Existen protocolos de desinfección manual que especifican el uso de líquidos como alcohol y amonio cuaternario. Para el uso de estos compuestos químicos se requiere de medidas de seguridad en contra de efectos como irritación en ojos y piel [3]. Por otro lado, se dispone de la desinfección por luz UV-C la cual es un 99 % efectiva. El virus debe ser expuesto a esta luz durante 9 minutos a una distancia de 2 metros mínimo para una desinfección óptima y es capaz de llegar a desinfectar lugares que no son alcanzables por agentes líquidos. La alta efectividad de este tipo de radiación, hace que no pueda ser usada por personas manualmente [4]. Por ello, se busca la implementación de un sistema autónomo de desinfección por luz UV-C sin necesidad de supervisión humana. Se realizará el diseño y la implementación de un robot móvil autónomo capaz de navegar entre personas cuidadosamente, desinfectar espacios cerrados con luz UV-C y auto-recargarse. Se deberá además definir una arquitectura de software de control para las acciones de SLAM, navegación social y sistema de auto-recarga, también una arquitectura de hardware para las conexiones de los distintos sensores y actuadores, como cámaras y motores. El documento se encuentra seccionado en 4 capítulos. El capítulo 1 describe el marco teórico y los objetivos, el capítulo 2 detalla la metodología seguida, el capítulo 3 detalla los resultados de las pruebas simuladas como de la implementación, y finalmente el capítulo 4 presenta conclusiones y recomendaciones.

1.1 Descripción del Problema

Todo espacio interior aloja virus y bacterias en el aire. Un ejemplo es el COVID-19, un virus altamente contagioso que es capaz de sobrevivir y mantenerse vivo incluso en objetos. Desde principios del 2021 se ha buscado el regreso a la modalidad de trabajo presencial en espacios cerrados. Este cambio provoca que sea más propenso la contracción del virus, debido al posible contagio por tacto y vía aerosol.

Es posible desinfectar manualmente con la ayuda de líquidos desinfectantes, sin embargo, este método no es totalmente eficiente, no cubre ciertas partes del espacio, existe error humano y además tiene efectos secundarios sobre las personas. Un método efectivo para la desinfección, es el uso de lámparas UV-C. Por otro lado, este método no puede ser usado por personas debido a efectos dañinos como quemaduras en la piel. Por lo tanto, no se dispone y se requiere de un método para desinfección eficiente que no disponga ningún riesgo en contra de la salud humana. Es necesario que se realice la desinfección a partir de lámparas UV-C automáticamente con seguridad, sin necesidad de intervención de personas.

1.2 Justificación del problema

El COVID-19 ha causado daños en todo el mundo y la disminución de su contagio hasta el día de hoy es una prioridad. “En Ecuador hasta el 21 de julio de 2020 se registró 74.620 casos confirmados por el Ministerio de Salud Pública, ..“ [4]. Incluso hasta mediados del 2021, siguen habiendo casos de contagio por COVID-19 [5].

Para disminuir su propagación debido a la modalidad de trabajo presencial, es importante realizar desinfecciones progresiva y periódicamente de los espacios. Como especifica la Cámara de Comercio de Bogotá, estos espacios deben de ser desinfectados por lo menos dos veces al día dado que el virus es capaz de mantenerse activo en objetos y en el aire hasta más de un día [6].

A pesar de que tales espacios pueden ser desinfectados de manera manual, hay un aumento de tiempo y riesgo de exposición al virus dado que se ha comprobado que el 56 % de contagios se dan por vía aerosol [7]. Una de las causas es la falta de implementación de un sistema autónomo y la desinfección manual poco efectiva,

dependiente de los químicos utilizados y tener efectos secundarios en contra de las personas [8].

A diferencia de una desinfección manual con químicos, se ha comprobado que la luz UV-C puede realizar una desinfección del 99 % con una correcta exposición, de manera eficiente y cubriendo un mayor espacio [9]. Desde el brote de la pandemia se implementaron robots de desinfección en más de 40 países, incluyendo Estados Unidos, países de Asia y Europa. Estos han presentado ser muy útiles en el proceso de eliminación del virus [10].

1.3 Objetivos

1.3.1 Objetivo general

Diseñar e implementar un robot móvil autónomo con sistema de auto-recarga y navegación social capaz de realizar desinfección de espacios cerrados con luz UV-C.

1.3.2 Objetivos específicos

- Diseño de controlador para el movimiento de la plataforma móvil del robot.
- Diseño e implementación de sistema de auto-docking y auto-recarga del robot.
- Diseño e implementación de arquitectura de control de software para navegación social.
- Implementación de modelo extendido de fuerzas sociales para el movimiento del robot.
- Evaluar arquitectura y algoritmo de navegación social de plataforma móvil.

1.4 Marco teórico

1.4.1 Robot Móvil Autónomo

Un Robot Móvil Autónomo (RMA) se caracteriza por tener la capacidad de navegar a través de un espacio evadiendo mientras realiza tareas sin necesidad de supervisión humana. Requiere el uso de sensores que le permitan conocer el espacio a su alrededor y actuadores para ejecutar movimientos. También debe disponer de un mapa, un método de localización y un algoritmo de navegación [11]. En la Figura 1.1 se puede observar algunos de los RMA de la empresa CONVEYCO.

Figura 1.1 Robots móviles autónomos de CONVEYCO [12].

1.4.2 Sensores de robots móviles

Se usan principalmente para poder conocer el entorno alrededor del robot, estimar su posición u otras características. Existen diferentes tipos basados en distintos fenómenos físicos como sonido, láser y magnetismo [13].

1.4.2.1 Encoders

Son sensores giratorios que ayudan a medir un grado específico de rotación. En robótica móvil, es común usar encoders incrementales que miden la cantidad de giro de los actuadores. Sirven principalmente para conocer el desplazamiento del robot [14].

1.4.2.2 Lidar

Es un sensor capaz de medir distancia usando la reflexión de luz. Algunos pueden medir distancias en una sola dirección y otros son capaces de medir en un rango más amplio con la ayuda de actuadores como motores. Los más comunes son rotatorios y proveen distancias en un plano en específico 2D [15].

1.4.2.3 Cámara de profundidad

Este tipo de cámaras son capaces de proveer datos sobre la distancia a obstáculos en su línea de visión. Generalmente se disponen de cámaras que dan imagen a color y junto a esta la distancia hacia cada píxel de la imagen [16].

1.4.3 Cinemática de robot móvil diferencial

Un robot diferencial se define por tener dos llantas centrales a su marco responsables de su movimiento y una rueda "loca" (caster wheel en inglés) para su estabilización. Este tipo de robot trabaja muy bien en interiores. Su forma se observa en la Figura 1.2. Se traslada únicamente en el eje X de su plano local como se especifica en la Ecuación 1.1, donde w_r y w_l corresponden a la velocidad rotacional de sus llantas y r corresponde al radio de las llantas [17].

$$v = v_x = v_1 + v_2 = r\frac{w_r + w_l}{2}; v_y = 0 \tag{1.1}$$

Su velocidad de rotación se expresa en la Ecuación 1.2, donde d corresponde a la distancia entre las llantas [17].

$$w = w_1 + w_2 = r\frac{w_r - w_l}{d} \tag{1.2}$$

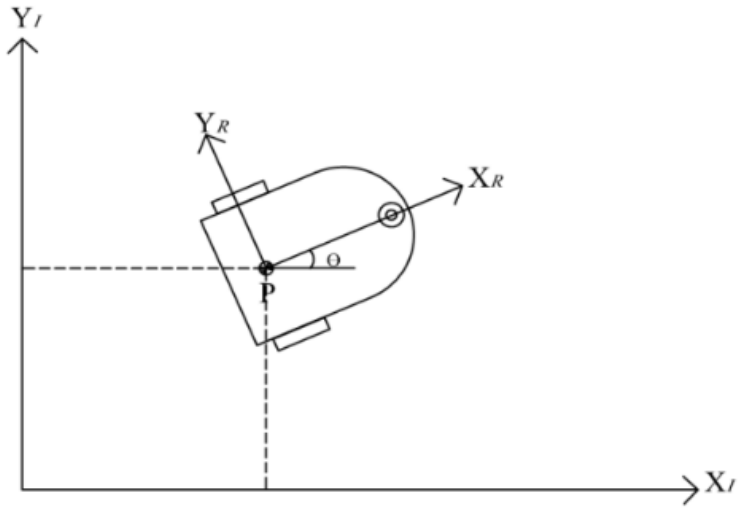


Figura 1.2 Robot diferencial en sistema de coordenadas Cartesianas [17].

1.4.4 Controlador PID

Un controlador PID, es un método de control ampliamente usado, principalmente para motores DC. Este controlador modifica la salida usando una componente proporcional del error, pero además también es capaz de agregar un componente integral y derivativo. Esto hace que el controlador se base en el valor de tres constantes, K_p, K_i y K_d. En la Figura 1.3 se puede observar un diagrama de control de un motor DC a través del uso de un PID [18].

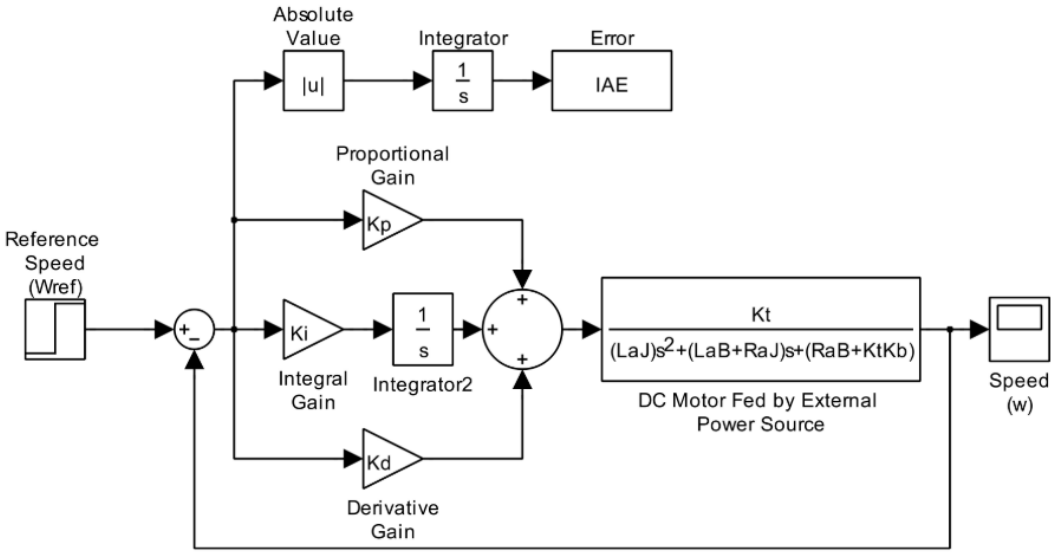


Figura 1.3 Diagrama de control PID de motor DC [18].

La constante proporcional modifica la salida en base al valor del error, a medida que se aumenta esta constante, se disminuye el tiempo para el estado estable de la señal de salida.

El uso de la parte integral, acumula el error en el tiempo y permite que el error de estado estacionario final del sistema se iguale a cero. El aumento de la constante integral, es capaz de llevar a la inestabilidad y aumenta levemente la velocidad de respuesta.

Para eliminar oscilaciones, se usa la constante K_d, esta deriva en el tiempo la señal de salida y busca predecir el cambio de la señal para compensarla y que no ocurran oscilaciones. Es altamente sensible y capaz de generar una señal inestable.

1.4.5 Métodos de sintonización de PID

1.4.5.1 Método de Ziegler Nichols de lazo cerrado

Para el uso de este método es necesario poder trabajar con el sistema o planta real y poder obtener una salida en base a una entrada.

Los pasos para el cálculo de las constante son:

1. Asignar valores de 0 para la constante integral K_i y derivativa K_d.
2. Elevar el valor de la constante proporcional K_p hasta obtener oscilaciones de frecuencia fija en la señal de salida.
3. Obtener el periodo P_σ de la señal de salida.
4. Seguir las expresiones de la Figura 1.4, donde K_σ corresponde al valor de K_P obtenido de la señal anterior y calcular el valor de las constantes para el controlador deseado.

PID Type	K_p	T_i	T_d
P	$0.5\,K_{cr}$	∞	0
PI	$0.45\,K_{cr}$	$\frac{P_{cr}}{1.2}$	0
PID	$0.6\,K_{cr}$	$\frac{P_{cr}}{2}$	$\frac{P_{cr}}{8}$

Figura 1.4 Constantes de controlador PID por el método de Ziegler Nichols de lazo cerrado [18].

1.4.5.2 Método heurístico o tanteo

Este método trata sobre asignar valores a las constantes del controlador PID como se crea conveniente hasta tener una señal de salida aceptable. Para ello, se debe tener en cuenta el efecto que tiene cada una de las constantes sobre el sistema. A continuación se especifica un posible procedimiento que se sigue para este método [18]:

1. Asignar valor de 0 a la constante K_i y K_d.
2. Asignar un valor bajo de K_p y duplicarlo progresivamente hasta obtener una rápida respuesta y oscilaciones antes de la inestabilidad.
3. Reducir el valor de K_p a la mitad.
4. Asignar un valor bajo de K_i y duplicar hasta tener oscilaciones leves y un error de estado estacionario de cero.
5. Si se desea un controlador de tipo PID, elevar el valor de la constante K_d hasta disminuir las oscilaciones como convenga.
6. Finalmente, modificar los valores de las constantes cerca de su último valor hasta obtener una respuesta deseada.

1.4.6 Simultaneous Localization and Mapping (SLAM)

Es el proceso de realizar localización y mapeo del entorno de un robot con el uso de distintos sensores de proximidad y odometría. Generalmente los sensores usados son basados en láser y encoders [19].

1.4.6.1 Odometría

Proveniente de las dos palabras griegas hodos (significa “viaje“) y metron (significa “medición“), odometría se refiere al cálculo y estimación del cambio de la pose del robot, puede ser un cambio por traslación o rotación [20].

1.4.6.2 Mapeo

Es la acción de generar y grabar la silueta de un espacio que rodea al robot. Un mapa es una descripción del entorno en el que se encuentra el robot, en el cual se

muestra información como puntos de referencia para su localización y obstáculos. Pueden ser en 2D y 3D dependiendo de qué método de SLAM se use [21]. Para generar un mapa existen varios métodos y algoritmos, se diferencian por los distintos tipos de sensores que usan y cómo usan su información. En la Figura 1.5 se puede observar un mapa generado por Cartographer ROS [22].

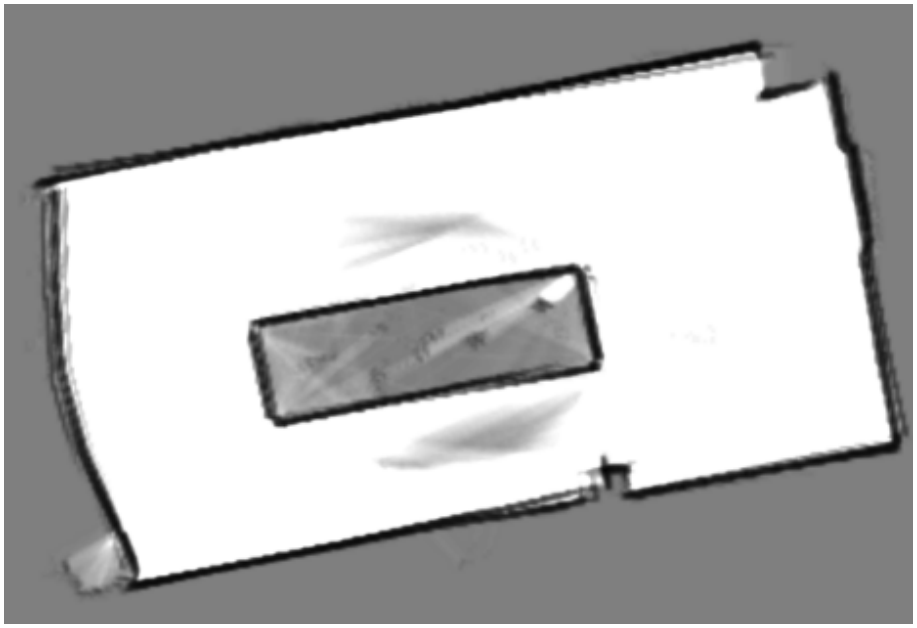


Figura 1.5 Mapa obtenido con Cartographer ROS [23].

1.4.6.3 Localización

Corresponde al cálculo de pose del robot en referencia al espacio que lo rodea y de sensores de proximidad. Los algoritmos de localización funcionan a partir de lecturas de encoders, lidares y cámaras de profundidad. La información de localización permite generar trayectorias de movimiento basadas en su posición en el mapa [24].

1.4.6.4 Planificación de trayectoria

Se define como el acto de generar el camino a seguir por un robot para llegar desde un punto A hacia un punto B, tal trayectoria puede ser estática o dinámica. Las técnicas para planificación se clasifican en reactivas y predictivas. Las reactivas se basan por generar su trayectoria y evadir obstáculos en el momento que este se encuentran a una distancia muy cercana, un ejemplo es DWA Planner [25]. El tipo de planificador predictivo, usa la información de obstáculos detectados junto a sus

velocidades para predecir su movimiento y evadir obstáculos con anterioridad junto a trayectorias más suaves, un ejemplo es TEB Planner [26] [27].

Los planificadores de ruta se basan en la eficiencia y efectividad del movimiento, en ocasiones estos pueden causar colisiones o movimientos bruscos cuando se usan en espacios donde se tiene de por medio la interacción humana [28].

1.4.7 Navegación social

Se centra en que un robot móvil debe de ser capaz de considerar aspectos sociales de la interacción con personas en su movimiento. El robot no debe asustar a las personas y las intenciones de sus movimientos deben de ser predecibles. Un robot de navegación social debe estar consciente de las acciones permitidas y prohibidas del espacio social [29].

1.4.7.1 Confort

Se define en el área de navegación social, como la forma de navegación que le da a un individuo un sentimiento de seguridad. Un robot mantiene el confort de una persona, cuando se mueve alrededor de la misma respetando su espacio personal haciéndola sentir segura de que ningún accidente puede ocurrir [30].

1.4.7.2 Modelo de Fuerzas Sociales

También llamado como Social Force Model (SFM) [31], este es un modelo que simula el movimiento de agentes (robots o personas) basado en fuerzas repulsivas entre obstáculos y atractivas entre puntos de interés. La Ecuación 1.3 define la fuerza principal de atracción en relación al punto de destino, de la cual v^o y ϵ corresponden a la velocidad deseada (velocidad máxima del agente) y su dirección respectivamente. La velocidad deseada es definida como $v^o(t) = v^o * \epsilon(t)$, mientras que τ es el tiempo de relajación. El cálculo de la fuerza repulsiva se calcula en base a obstáculos y otros agentes como la suma de todas las fuerzas repulsivas experimentadas como se especifica en la Ecuación 1.4. Se define como una fuerza decadente exponencialmente con la distancia euclidiana $d_{ij} = p_j - p_i$ desde el punto i hasta el punto j. A_o es una constante correspondiente a la magnitud de

la fuerza repulsiva y B_o a un parámetro de rango de efectividad [32].

$$f^{att}(t) = m * \frac{v^o(t) - v(t)}{\tau} \tag{1.3}$$

$$f_{ij}^{rep}(T) = \sum_{j \epsilon Q_o}^{Q_o} A_o * e^{\frac{-d_{ij}}{B_o}} * \vec{d}_{ij} \tag{1.4}$$

Con la suma de las fuerzas de atracción y repulsivas experimentadas por el agente, se dispone de una F_{total} de la cual se obtener la aceleración correspondiente para el movimiento del agente, como se especifica en la Ecuación 1.5 [33].

$$F_{total}(t) = F^{att}(t) + F^{rep}(t) \tag{1.5}$$

1.4.7.3 Índice Social Individual

El Social Individual Index (SII) en inglés, es una métrica de navegación social. Esta mide el confort físico de las personas en función de su espacio personal. Se calcula a partir de la Ecuación 1.6, donde (x_i^p, y_i^p) corresponde a la posición de la persona y (x_r, y_r) a la posición del robot o agente. σ_0^p es la desviación estándar de la distancia límite de confort, esta última se define como especifica la Ecuación 1.7 donde d_c es seleccionada heurísticamente como la distancia de confort de los agentes involucrados. N es la cantidad de personas alrededor del robot [34].

$$SII = \underset{i=1:N}{max}\, exp(-((\frac{x_r - x_i^p}{\sqrt{2}\sigma_0^p})^2 + (\frac{y_r - y_i^p}{\sqrt{2}\sigma_0^p})^2)) \tag{1.6}$$

$$\sigma_0^p = \frac{d_c}{2} \tag{1.7}$$

1.4.8 Auto-docking

El término auto-docking proviene del inglés, se refiere al sistema autónomo en el que un robot es capaz de regresar y conectarse a una estación automáticamente.

Este tipo de sistemas son muy útiles para casos en los que se desea la recarga automática de la energía de un robot [35].

Es común su implementación con sensores infrarojo (IR) como se especifica en la Figura 1.6. Con transmisores y receptores IR se permite la alineación de robot y poder ser insertado en su estación [36] [37]. Adicionalmente, la estación debe de tener una forma definida, para que el robot pueda insertarse de manera sencilla y precisa [35].

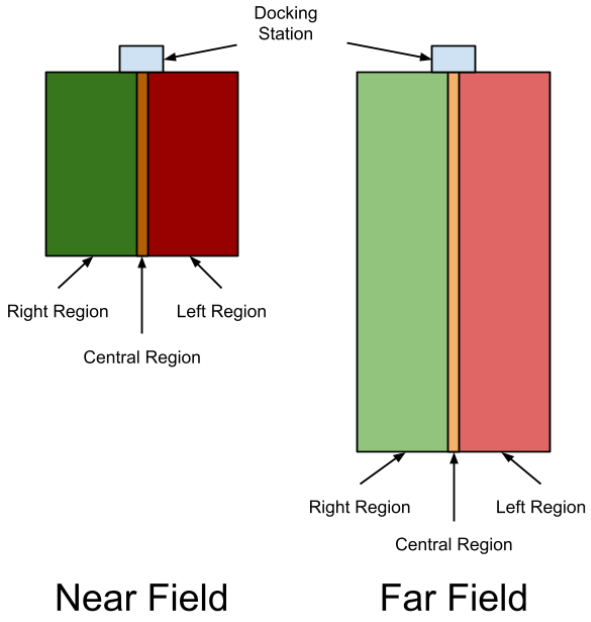


Figura 1.6 Leds IR para auto-docking usado en robot Kobuki [38].

1.4.9 Divisor de voltaje

Es un circuito que tiene como fin reducir el voltaje en un factor específico y en una relación lineal según la ley de Ohm. Es un circuito sencillo que tiene resistores en serie como se observa en la Figura 1.7.

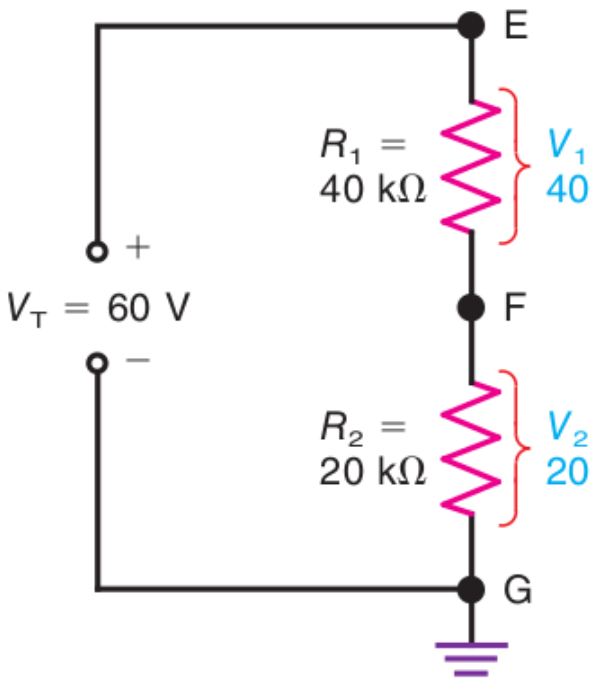


Figura 1.7 Circuito divisor de voltaje [39].

En un circuito divisor de voltaje, siguiendo la Figura 1.7, el voltaje en V_F es disminuido a partir de V_T dependiendo de las resistencias empleadas. Esta interacción es definida por la Ecuación 1.8 [39].

$$V_F = \frac{R_1}{R_1 + R_2} * V_T \tag{1.8}$$

CAPÍTULO 2

2. METODOLOGÍA

Todos los archivos del proyecto, distintos paquetes y códigos pueden ser encontrados en un repositorio de gitlab: `https://gitlab.com/sasilva1998/amr`

2.1 Antecedentes al Diseño

Previo al presente proyecto, se realizó como parte de prácticas preprofesionales y materia integradora el diseño de la estructura para el robot de desinfección por parte del estudiante Fernando Allauca.

Se definió la estructura necesaria para poder colocar los distintos sensores en el robot y las lámparas UV-C y se realizaron todos los análisis de esfuerzo correspondientes. El diseño se observa en la Figura 2.1.

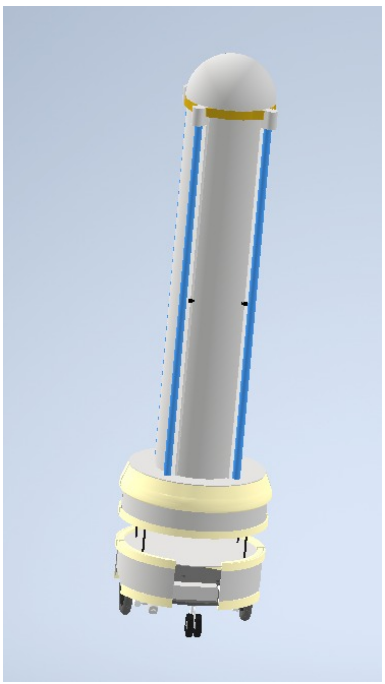


Figura 2.1 Estructura base para el robot de desinfección (Cortesía de Fernando Allauca).

Adicionalmente, se realizó por parte de Ronald Vélez el desarrollo de una aplicación para paros de emergencia manuales de las lámparas UV-C como parte de una materia

de itinerario. Esta aplicación usa un servidor MQ Telemetry Transport (MQTT) para la comunicación. La pantalla de la aplicación se encuentra en el Anexo 1.

2.2 Hardware inicial

En un principio se otorgó por el cliente un hardware específico para el desarrollo de la solución. Este se especifica en las siguientes subsecciones.

2.2.1 Plataforma Robótica Arlo

Figura 2.2 Plataforma Arlo Robot completa [40].

Esta es una plataforma móvil creada para llevar a cabo misiones complejas autónomas, se la puede observar en Figura 2.2. Las características se encuentran especificadas en la Tabla 2.1 [40].

Tabla 2.1 Características de Plataforma Arlo de Parallax [40].

Características de plataforma Arlo	
Material de pisos/placas	Polietileno de alta densidad
Material de motores	Aluminio
Actuadores	Motores DC 12V
Tipo de encoders	Ópticos de 144 posiciones
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 ensamblado	20 libras
Temperatura de operación	0 a 70 ℃

2.2.1.1 Motores DC

Los motores usados, corresponden a la plataforma ARLO de Parallax. Estos son motores de 12V con escobillas de cubierta de aluminio 6061 maquinado. Se encuentran en la Figura 2.3.

Figura 2.3 Motor DC 12V de Parallax [41].

Sus características se encuentran en la Tabla 2.2 [41].

Tabla 2.2 Características de motores DC [41].

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 ℃
Corriente	2 – 8 Amperios dependiendo del terreno
Diámetro llantas	6 pulgadas

Estos motores representan ser muy útiles para la solución a armar debido a que disponen de un alto torque y capacidad de carga.

2.2.1.2 Encoders ópticos

Se usaron los encoders propios de la plataforma Arlo como se muestran en la Figura 2.4. Usa dos sensores y receptores IR que le dan la capacidad de conocer el sentido y magnitud de giro de los motores. Tiene 144 pulsos por giro [42].

Figura 2.4 Encoders de cuadratura de 36 posiciones [42].

Estos sensores nos permiten obtener retroalimentación del giro de los motores y

realizar la implementación de un controlador.

2.2.1.3 DHB-10 Driver de motores DC

Es un controlador dual de puente H para motores DC. Puede ser controlado a través de comandos por comunicación serial y Pulso De Ancho Modulado (PWM). Las características del driver se encuentran en la Tabla 2.3 y sus conexiones en la Figura 2.5 [43].

Tabla 2.3 Características de DHB-10 [43].

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 ℃

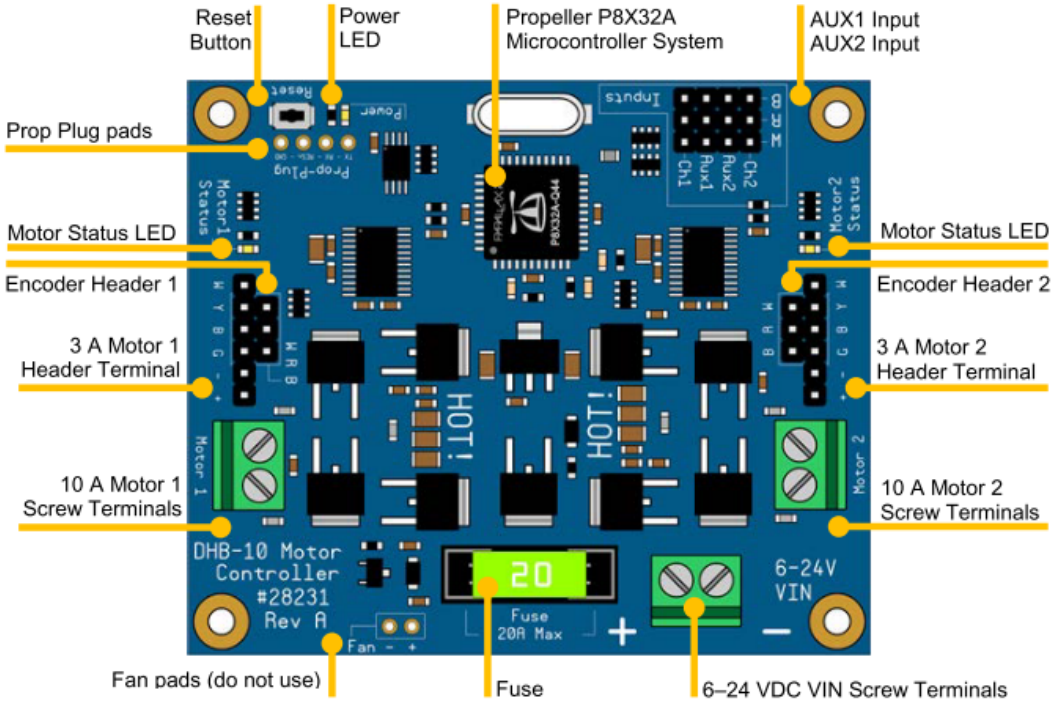


Figura 2.5 Controlador de motores DHB-10 [43].

Este controlador de motores satisface nuestra necesidad principalmente porque nos permite obtener la retroalimentación sobre la velocidad de los motores. Además su uso por comunicación serial permite que sea fácil de implementar y controlar desde un microcontrolador.

2.2.1.4 Baterías

Las baterías que se nos entregaron son de ácido-plomo. Se pudieron obtener dos de estas, ambas de 12V 7Ah. Desde un principio se consideró que ambas fueran conectadas en paralelo para aprovechar la mayor cantidad de carga posible.

2.2.2 RPLIDAR A1

Es un escáner láser 2D de 360 grados que puede medir hasta una distancia de 12 metros. Los datos de nube de punto que genera pueden ser usados para generar mapas y localización. Tiene una frecuencia media de escaneo de 5.5hz y máximo de 10hz. En la Figura 2.6 se puede observar todo el sistema que compone al lidar mencionado [44].

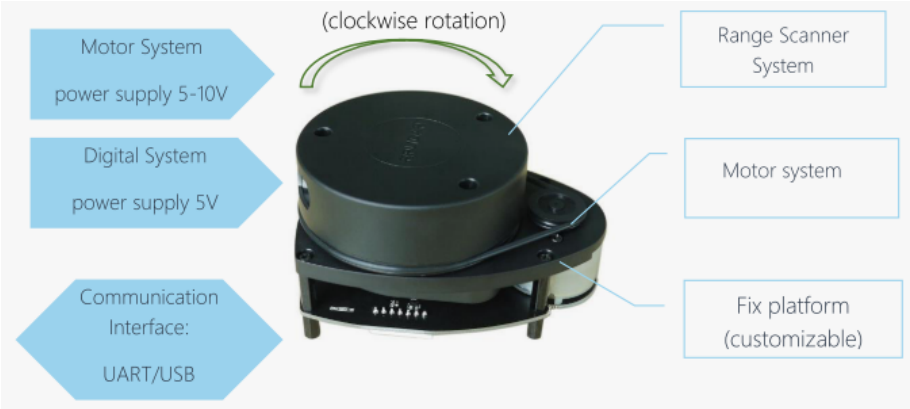


Figura 2.6 Composición de RPLIDAR A1 [44].

Este sensor es uno de los más primordiales ya que otorga una gran cantidad de información al robot sobre su entorno y permite generar mapas, localizar y navegar.

2.2.3 Cámara de profundidad Intel D435i

Es una cámara de sistema estéreo de profundidad. Es de tamaño pequeño y además contiene un Inertial Measurement Unit (IMU) capaz de medir en 6 grados de libertad. Hace uso de láser IR para el cálculo de distancias y provee imágenes con una resolución HD. La cámara se puede observar en la Figura 2.7.

Figura 2.7 Cámara Intel D435i [45].

Especificaciones adicionales de la cámara se especifican en la Tabla 2.4

Tabla 2.4 Características de cámara D435 [45].

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

Fue necesario conocer la distancia hacia un persona para poder implementar medidas de seguridad, esta cámara cumplió con este requerimiento, tanto para hacer un procesamiento de imágenes y conocer la distancia del robot hacia una persona.

2.2.4 Cámara de odometría Intel T265

Es una cámara de seguimiento basada en la fusión de visión y sensores inerciales. Está compuesta por dos cámaras de ojo de pescado, un IMU y un procesador ASIC. Esta se puede observar en la Figura 2.8.

Se usa principalmente para conocer la posición del robot y poder generar una buena odometría.

Figura 2.8 Cámara Intel T265 [46].

Es importante conocer la posición del robot para su navegación. La capacidad de otorgar odometría de este dispositivo, ayudó a cumplir con los requerimientos de autonomía del robot.

2.2.5 Inversor de 12V DC a 110V AC

Para el encendido de las lámparas UV-C se nos otorgó un inversor de 12V DC a 110V AC de 95 % de eficiencia, igual que el que se observa en la Figura 2.9.

Figura 2.9 Inversor de 12V DC a 110V AC.

2.2.6 Cargador de baterías de 12V

Se dispuso de un cargador de baterías de 12V estándar para construir la estación de auto-recarga. Este se observa en la Figura 2.10.

Figura 2.10 Cargador de baterías de 12V.

2.2.7 Lámparas de luz tipo UV-C

Obtuvimos lámparas de luz UV-C con sus respectivos marcos tamaño T8. Estos pueden ser observados en la Figura 2.11. Características adicionales de las lámparas se encuentran en la Tabla 2.5.

Tabla 2.5 Características de lámparas UV-C [47].

Características de lamparas UV	
Potencia (W)	40
Diámetro de tubo (mm)	26
Potencia ultravioleta (W)	20.8
Intensidad de radiación ultravioleta (uW/cm)	200
Tiempo de vida (h)	6000

Figura 2.11 Foco de tipo UV-C de 40W [47].

2.3 Requerimientos del diseño

Previo al diseño, especificaron ciertos requerimientos para la solución:

- El robot completo no debe de exceder las 50 libras de masa.
- Debe navegar de manera autónoma fuera y dentro del área de desinfección.
- Dedicado para espacios cerrados.
- No debe afectar al confort de las personas a menos de 0.9m.
- Debe de detectar personas y apagar las lámparas UV-C a menos de 4 metros.
- Bajo cierto nivel de voltaje (11.5V) el robot debe recargarse autónomamente.

2.4 Escenario

El escenario en que se desenvuelve el robot se ha definido en dos etapas especificadas a continuación. Ambos espacios deben ser cerrados y de piso plano liso.

2.4.1 Escenario de desinfección activa

Corresponde al espacio específico que el robot desinfectará. A continuación se especifican las características del mismo:

- Es delimitado por un área en específico, puede ser un cuarto o un pasillo.
- No debe haber presencia de personas en el espacio.

- Se debe de disponer de un camino abierto para el movimiento del robot por donde se desea desinfectar.
- No debe haber objetos que puedan ser afectados por la radiación UV-C.

En general, se recomienda que previo a la desinfección, se debe notificar a las personas que un espacio va a ser desinfectado y ser evacuadas.

2.4.2 Escenario de desinfección inactiva

Corresponde al escenario y trayectoria en que el robot se mueve hacia el área de desinfección sin usar las lámparas UV-C. El escenario de desinfección inactiva debe de tener las siguientes características:

- Puede darse la presencia de humanos.
- Debe ser amplio.

2.5 Hardware adicional

En esta sección se especifica el hardware adicional que se consideró para la construcción de la solución en base a los requerimientos.

2.5.1 Jetson Nano

Es una computadora de inteligencia artificial, ideal en aplicaciones en las que se necesita altas tasas de procesamiento. Es pequeña, potente, permite realizar procesamiento de imágenes. Tiene las especificaciones definidas en la Tabla 2.6.

Tabla 2.6 Especificaciones de Jetson Nano de 2GB de memoria RAM [48].

Jetson Nano	
Característica	**Especificación**
GPU	128-core NVIDIA Maxwell
CPU	64-bit Quad-core ARM A57 (1.43GHz)
Memoria RAM	2GB 64-bit LPDDR4
Redes	Gigabit ethernet
Conexión inalámbrica	Disponible con uso de adaptador inalámbrico 802.11ac
USB	1x puerto USB 3.0 tipo A, 2x puertos USB 2.0 tipo A
Pantalla	HDMI
Cabecera de 40 pines	GPIOs, I2C, I2S, SPI, PWM, UART
Cámara	1x MIPI CSI-2 connector
Almacenamiento	MicroSD (no incluida)
Otras entradas y salidas	Cabecera de 12 pines (potencia y UART) y cabecera de 4 pines para ventilador
Tamaño	100mmx80mmx29mm
Potencia	10W (máximo)

Su tarea es obtener todos los datos de sensores y enviar los comandos de control para el movimiento de la plataforma. Los datos son enviados y recibidos vía WiFi haciendo uso de Husarnet.[1]

Para seleccionarla, se realizó una matriz de decisión. En la Tabla 2.7 se puede observar los criterios considerados y en la Tabla 2.8 los valores de cada criterio demostrando la mejor opción.

Tabla 2.7 Ponderación de características para decidir el centro de cómputo del robot.

Centro de cómputo de robot			
Disponibilidad	**Procesamiento de imágenes**	**Peso**	**Tamaño**
10.00 %	30.00 %	25.00 %	25.00 %

Tabla 2.8 Puesta de valores de criterios para centro de cómputo de robot.

	Disponibilidad	**Procesamiento de imágenes**	**Peso**	**Tamaño**	**Total**
Jetson Nano	5	5	5	5	4.5
Laptop Asus	5	2	3	3	2.6
Raspberry PI 3	5	2	5	5	3.6

[1] https://husarnet.com/

2.5.2 Placa de desarrollo ESP32

Es un microcontrolador multiuso, tiene un alto nivel computacional y es altamente versátil. Las especificaciones de este microcontrolador se encuentran en la Tabla 2.9. En la Figura 2.12 se observa al microcontrolador.

Tabla 2.9 Especificaciones de placa de desarrollo ESP32 [49].

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

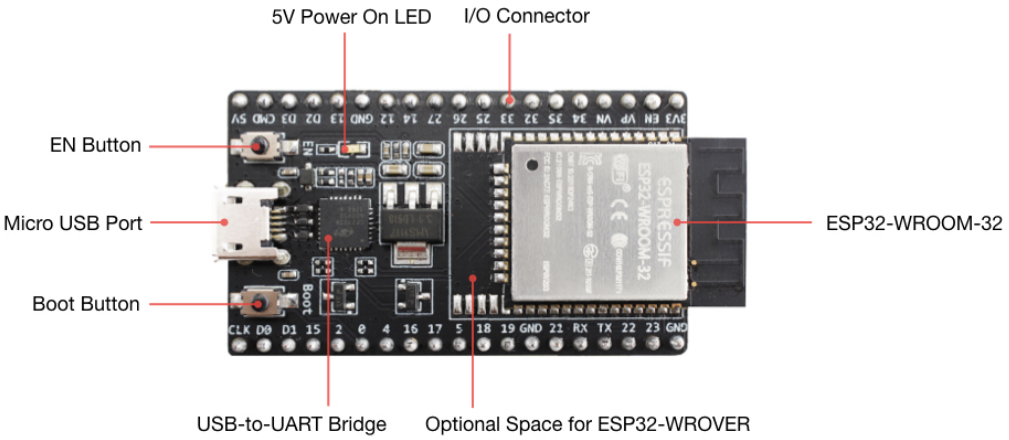


Figura 2.12 Kit de desarrollo ESP32 [49].

Para la selección de este microcontrolador se usaron las ponderaciones de la Tabla 2.10 y en la Tabla 2.11 se observa los valores otorgados.

Tabla 2.10 Ponderaciones para decisión de microcontrolador.

Microcontrolador				
Costo	**Disponibilidad**	**Procesamiento**	**WiFi**	**Tamaño**
10 %	10 %	20 %	40 %	20 %

Tabla 2.11 Valores de los criterios para cada microcontrolador considerado.

	Costo	**Disponibilidad**	**Procesamiento**	**WiFi**	**Tamaño**	**Total**
ESP32	3.5	3.5	5	5	5	4.7
Arduino Uno	4.5	5	3	0	4	2.35
Arduino Mega	3.5	5	3.5	0	3	2.15

2.6 Software

2.6.1 Robot Operating System

Se usó como base para el control del hardware del robot y simulaciones. Va instalado en un sistema operativo y ayuda a hacer una abstracción del hardware desde el software. Dispone del ROS API que permite realizar el intercambio de información como mensajes, generar acciones y servicios [50].

2.6.1.1 Sistema de archivos

Robot Operating System (ROS) tiene un sistema de archivos separados entre distintas carpetas y funcionalidades como se describe brevemente a continuación [51]:

- Paquetes: Mantienen la funcionalidad de lo que ejecuta ROS. En Internet se pueden encontrar una serie de paquetes libres que realizan distintas tareas.
- Tipos de mensajes: Incluye las distintas estructuras de mensajes a usar.
- Tipos de servicios: Define estructuras de peticiones y respuestas de una acción en específico que se desea, un ejemplo es el movimiento de un actuador del robot [51].

En la Figura 2.13 se puede observar la jerarquía del mismo.

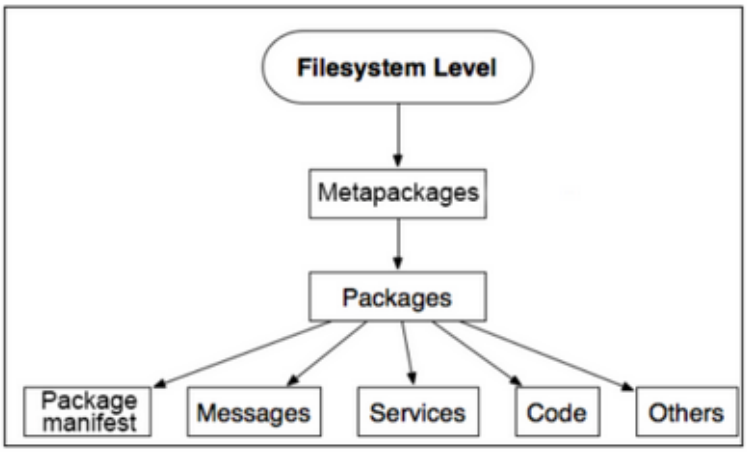


Figura 2.13 Sistema de archivos de ROS [51].

2.6.1.2 Estructura de software

ROS crea una red en la que todos los procesos ejecutados se conectan y se comunican. Cada uno es llamado un nodo. La forma de su estructura se puede observar en la Figura 2.17.

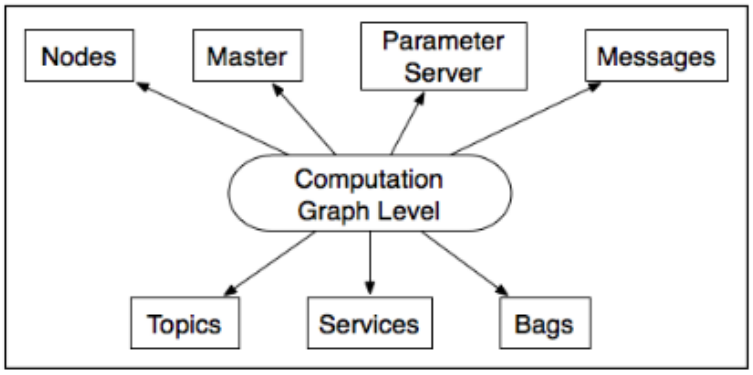


Figura 2.14 Estructura de software de ROS [51].

Su estructura se describe muy breve a continuación:

- Nodos: Son procesos donde el cómputo es realizado. Si se desea que un proceso esté conectado a la red de ROS, debe ser definido como un nodo.
- ROS master: Hace de servidor para la conexión de todos los nodos. Si no se tiene el master en el sistema, no se puede hacer la conexión entre nodos y otros.
- Mensajes: Un mensaje contiene datos en una estructura definida para proveer información a otros nodos.
- Tema: Cada mensaje debe estar asignado a una ruta en la red de ROS, es a lo

que se le llama tema o topic en inglés. Cuando un nodo envía datos usando un mensaje, se dice que este está publicando datos a un topic.

2.6.2 Lenguaje de programación

En ROS, es posible programar con C++ y Python. Para reconocer que lenguaje de programación usar, se utilizaron los criterios especificados en la Tabla 2.12. Así se obtuvo el lenguaje a usar como se especifica en la Tabla 2.13.

Tabla 2.12 Criterios de evaluación para lenguaje de programación.

Lenguaje de programación			
Experiencia	**Capacidad de cómputo**	**Cantidad de información**	**Facilidad**
30.00 %	20.00 %	30.00 %	20.00 %

Tabla 2.13 Matriz de decisión para escoger el lenguaje de programación a usar.

	Experiencia	**Capacidad de cómputo**	**Cantidad de información**	**Facilidad**	**Total**
Python	5	4	5	5	4.80
C++	3.5	5	4	3	3.85

Para la programación de los microcontroladores se escogió el lenguaje MicroPython[2] con el fin de mantener una misma línea de lenguaje de programación con los nodos de Python en ROS.

2.6.3 Algoritmos de SLAM

Se escogió delimitando en base al hardware disponible. De los diversos algoritmos para generar un mapa, se escogió Cartographer ROS[3] en base a la investigación realizada en [52].

Este método es capaz de hacer uso del lidar para generar un mapa y es robusto frente a fallas de odometría y ruido. Como especifica en [53], Cartographer ROS se encuentra entre los mejores algoritmos para la generación de mapas debido a

[2] https://micropython.org/

[3] https://google-cartographer-ros.readthedocs.io/en/latest/

que usa un ciclo de optimización para el mapa global y también dispone de localización [54].

2.6.4 Algoritmos de navegación

Se definió el uso de dos distintos métodos de navegación para cada etapa de movimiento del robot.

2.6.4.1 Algoritmo de navegación de desinfección activa

Se tienen las siguientes especificaciones:

- Empieza la desinfección en un espacio libre de personas.
- El espacio a desinfectar tiene obstáculos presentes.

Según las especificaciones, se delimitó a que se use un planeador reactivo para la navegación, en este caso DWA Planner [25]. Este es fácil de implementar, no requiere de mucho procesamiento, y además se basa en la mejor trayectoria según distancias lo cual lo hace eficiente en espacios de solo obstáculos [55].

2.6.4.2 Algoritmo de navegación de desinfección inactiva

Para este caso se plantearon las siguientes especificaciones:

- El robot debe llegar al lugar de desinfección mientras se mueve entre personas y obstáculos.
- El movimiento del robot no debe afectar al confort de las personas alrededor.

Se ha seleccionado el uso del SFM [31] para generar el movimiento del robot. Se ha comprobado que es uno de los modelos que más se apega a generar movimientos humanos, definido en varias investigaciones como [56], [34] y [57].

2.6.5 Detección de personas

2.6.5.1 leg_detector

Es un paquete de ROS que permite la detección de personas usando lidar. Usa como principio la posición y separación común de las piernas de una persona. Hace

uso de machine learning y tiene previamente cargado un modelo entrenado para la detección.[4]

Se incluyó para generar una de las entradas del SFM y poder calcular la fuerza social que experimenta el robot.

2.6.5.2 darknet_ros

Es un paquete desarrollado para procesamiento de imágenes. Se basa en el sistema de detección y procesamiento de imágenes usando You Only Look Once (YOLO). Puede detectar hasta 80 objetos distintos, entre estos se incluye personas.[5]

Fue incluido para generar paros de emergencia en caso de la presencia de un humano mientras se realiza la desinfección.

2.6.6 Movimiento del robot

Se usó el paquete de ROS differential_drive[6]. Contiene para mover robots diferenciales basados en la Ecuación 1.1. Recibe como entrada la velocidad deseada y da salida las velocidades angulares correspondientes para cada llanta.

2.6.7 Simulación de personas

Se usó el paquete pedsim_ros[7]. Permite simular agentes con movimiento en base al SFM. Fue muy útil para realizar las pruebas simuladas de la implementación del SFM en el robot.

Los agentes de la simulación interactúan con el robot y se mueven entre distintas coordenadas configuradas previamente.

[4] http://wiki.ros.org/leg_detector
[5] https://github.com/leggedrobotics/darknet_ros
[6] https://wiki.ros.org/differential_drive
[7] https://github.com/srl-freiburg/pedsim_ros

2.6.8 Comunicación entre dispositivos

Se usaron algunos softwares para la conexión entre los dispositivos. Se presentan brevemente a continuación:

2.6.8.1 Shiftr

Es un servicio de broker MQTT que permite la transferencia de datos por Internet. Se lo consideró para el envío de información y mensajes entre los microcontroladores, aplicación de celular y Jetson Nano.

2.6.8.2 Husarnet

Permite conectar de manera inalámbrica por Internet varios dispositivos al servidor master de ROS y la obtención de los datos. Esto ayuda a aliviar el procesamiento de datos en el centro de cómputo para que otra computadora externa los procese.

2.6.8.3 rosserial

Se usó para conectar el ESP32 con la Jetson Nano por serial[8]. Este middleware permite enviar y recibir mensajes desde microcontroladores. Desde el microcontrolador, se usó la librería micropython-rosserial[9].

2.7 Arquitectura del sistema del robot

2.7.1 Hardware

2.7.1.1 Plataforma robótica

Se realizó un diagrama de conexiones de hardware detallando cómo se integraba cada componente de la plataforma móvil robótica. Este diagrama es presentado en la Figura 2.15.

Para energizar a la Jetson Nano se decidió usar dos Step Down de 5V 5A conectados en paralelo. También se consideró el uso de relés para el control del

[8] https://wiki.ros.org/rosserial

[9] https://pypi.org/project/micropython-rosserial/

inversor y luces RGB.

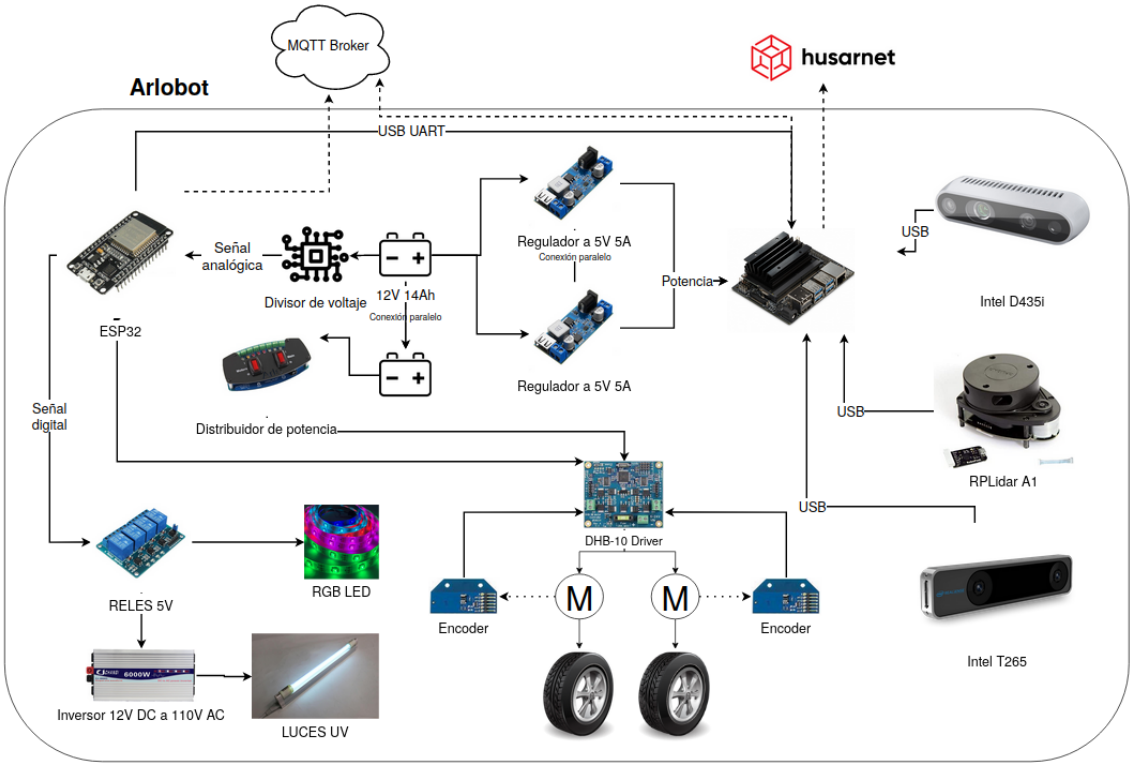


Figura 2.15 Conexiones generales de hardware en arlobot.

2.7.1.2 Estación de recarga

En la Figura 2.16 se puede observar el diagrama generado para la estación de recarga. Se dispuso a usar un adaptador de 5V para energizar el microcontrolador y un relé para cerrar y abrir el circuito de recarga. El microcontrolador se conectó al broker MQTT para recibir la señal cuando se deba de activar el relé.

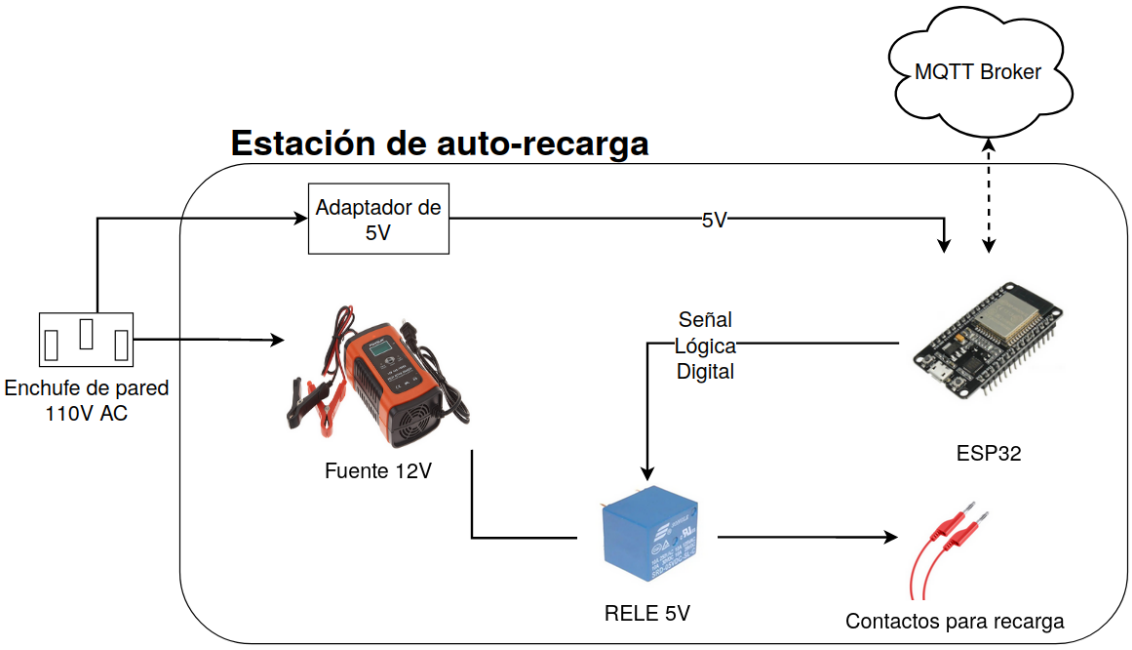


Figura 2.16 Estación de recarga.

2.7.2 Software

Se realizó un diagrama de la arquitectura de todos los nodos y algoritmos a implementar. Este puede ser observado en la Figura 2.17

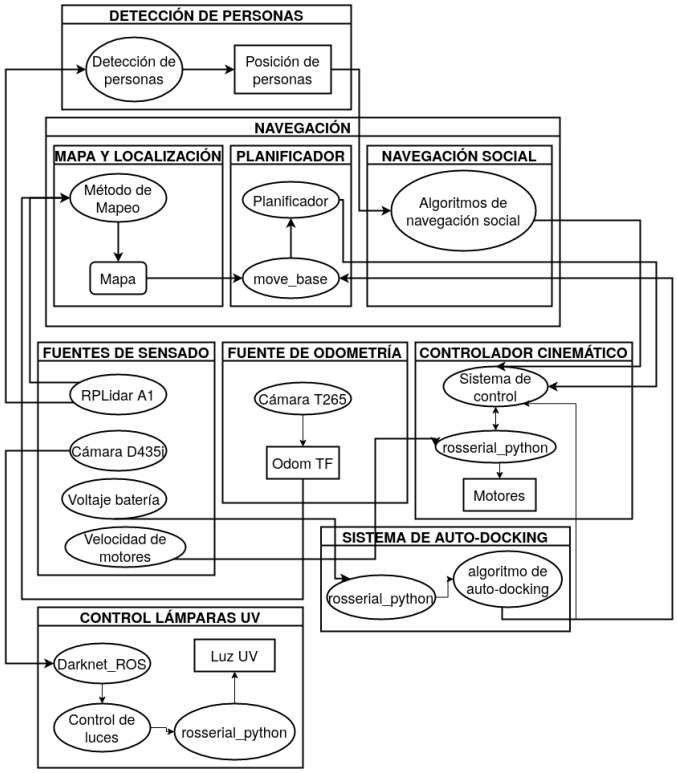


Figura 2.17 Estructura de software de solución.

Como se puede observar, consideraron las distintas etapas como de navegación

donde se requiere los algoritmos de SLAM y además la percepción con los distintos sensores.

2.8 Simulación en Gazebo

Para el desarrollo de los algoritmos y probarlos previo a la implementación, se hizo el usó de Gazebo[10] para realizar la simulación. En esta se usó el diseño estructural de Fernando Allauca.

En la Figura 2.18 se puede observar el entorno en 3D que se usó como escenario de simulación.

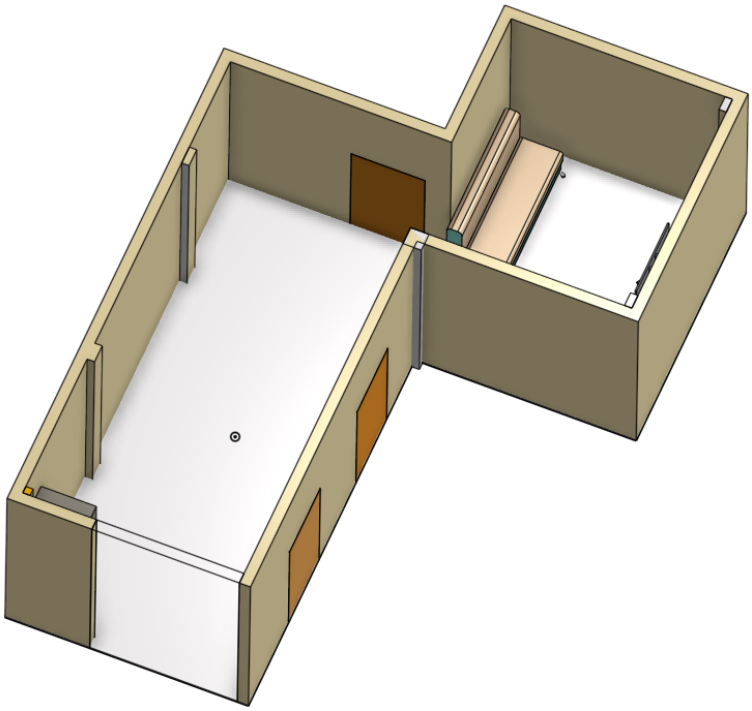


Figura 2.18 Entorno de simulación.

Para agregar el robot a la simulación se paso el diseño a OnShape[11] que es gratuito para estudiantes. El diseño se puede observar en Figura 2.19, este fue segmentado en varias partes (distinguidas por el color) para asignar el material de manera más simple.

[10] http://gazebosim.org/

[11] https://cad.onshape.com

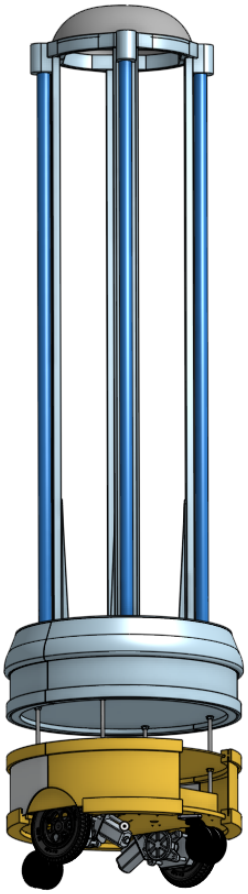


Figura 2.19 Diseño base de robot en OnShape.

Usando OnShape-To-Robot[12] se exportó un archivo de tipo Unified Robotic Description Format (URDF) para ingresar en la simulación de ROS y se incluyeron los plugins de Gazebo presentados brevemente a continuación:

2.8.1 Sensores

- Lidar: Para este sensor se incluyó el plugin de `libgazebo_ros_laser.so`.
- Cámara de profundidad: Se usó el plugin de `libgazebo_ros_openni_kinect.so`.
- Sensor de torque y fuerza: Se usó el plugin de `libgazebo_ros_ft_sensor.so`. Este último sirvió para conocer el torque generado en las llantas para su movimiento según la simulación.

[12] https://github.com/Rhoban/onshape-to-robot

2.8.2 Movimiento diferencial

Se usó el plugin `libgazebo_ros_diff_drive.so` que se basa en la Ecuación 1.1 como base para el cálculo de las velocidades.

En Gazebo, se realizaron pruebas de SLAM, navegación por planeador y social y movimiento de auto-docking. La simulación del robot en Gazebo se puede observar en la Figura 2.20.

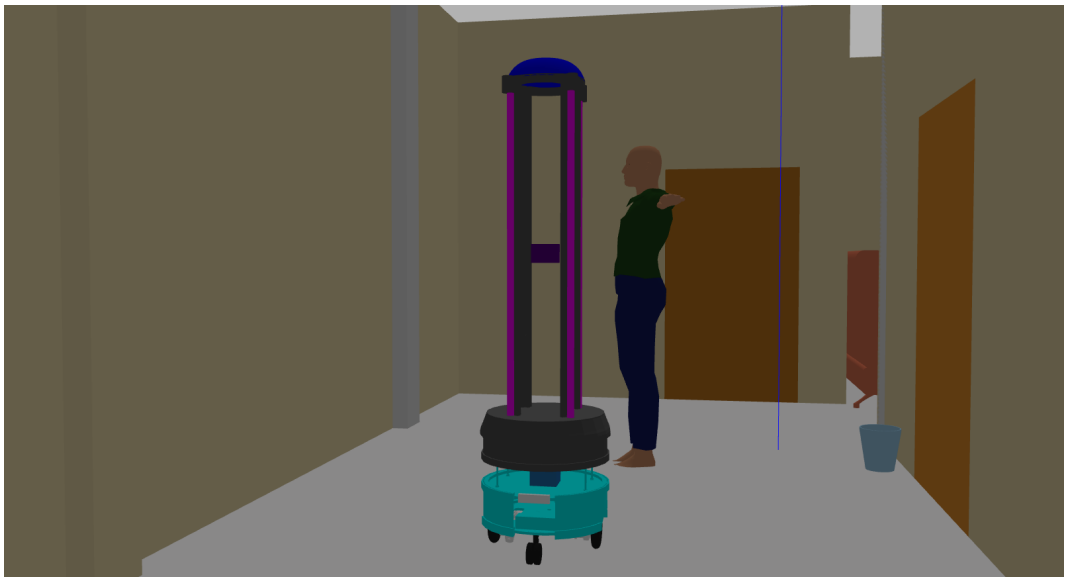


Figura 2.20 Simulación de robot en Gazebo.

2.8.3 Cartographer ROS

Para usar Cartographer ROS, se realizó la configuración de un archivo `.lua`. Este nos permite definir distintas características del mapeo como sensores a usar.

Los principales campos que se configuraron fueron los siguientes:

- map_frame: ‘map‘ (define el marco de referencia para el mapa)
- tracking_frame: ‘base_footprint‘ (define el marco de eje central del robot)
- odom_frame: ‘odom‘ (define el marco de odometría)

El resto del archivo de configuración puede ser visto en el Anexo 2.

2.8.4 Agentes sociales en simulación

Se usó el paquete `pedsim_ros` para el que se generó un archivo de escenario `.xml` en el cual se especificó los agentes y la silueta del espacio.

Para generar la silueta del espacio se generó el mapa y se utilizó el paquete `ros_maps_to_pedsim`[13] que permite pasar un mapa a archivo `.xml` de `pedsim_ros`.

Se agregaron dos agentes. Como se observa en la Figura 2.21, los agentes se mueven entre los distintos puntos A, B y C. La configuración del archivo se puede ver en el Anexo 4.

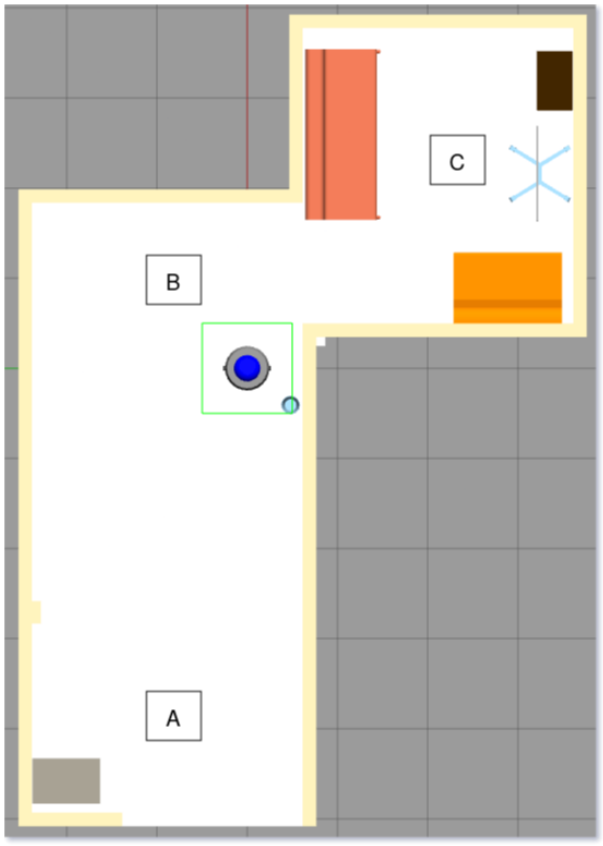


Figura 2.21 Espacio de entorno de simulación con coordenadas para el movimiento de agentes.

[13] https://github.com/fverdoja/ros_maps_to_pedsim

2.8.5 Nodo de modelo de fuerzas sociales

Se generó un servidor de acción en ROS con las entradas respectivas para el cálculo de las fuerzas sociales que se experimenta el robot. Las fuerzas usadas corresponden a la Ecuación 1.3 y Ecuación 1.4. Para este caso se consideraron dos fuerzas repulsivas y una atractiva tal y como se observa en la Figura 2.22 donde F_{att} corresponde a la fuerza de atracción hacia el destino deseado, F_{rep} corresponde a las fuerzas repulsivas de agentes y obstáculos. F_{res} corresponde a la fuerza resultante de la combinación de todas las fuerzas.

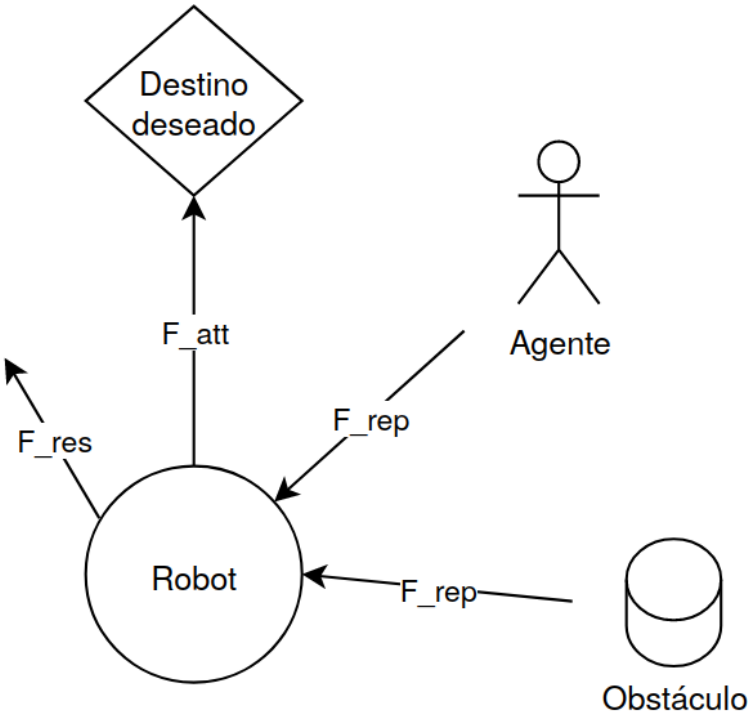


Figura 2.22 Fuerzas experimentadas por el robot para el movimiento por el modelo de fuerzas sociales.

En la Figura 2.23 observan las entradas y salidas del nodo implementado. En este tf corresponde a las transformadas de los distintos elementos.

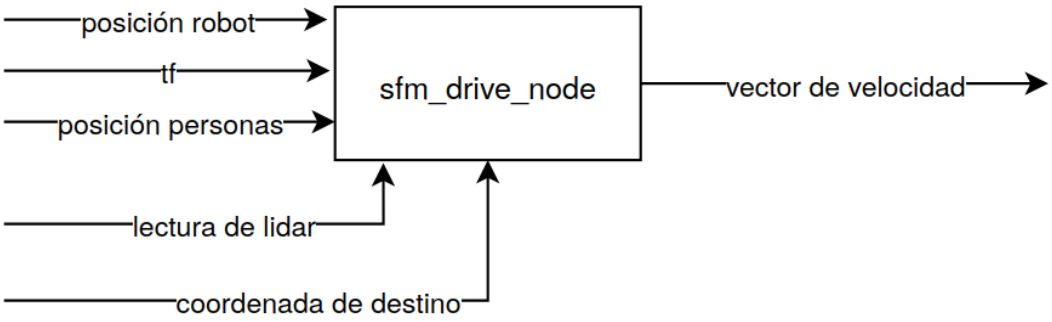


Figura 2.23 Nodo de ROS del modelo de fuerzas sociales junto con sus entradas y salidas.

2.8.6 Nodo de movimiento para auto-docking

Se consideró realizar el movimiento de auto-docking usando detección de marca de Aruco. Para el movimiento se usó el paquete `aruco_detect`[14]. Este determina y provee la posición de la marca a través de imágenes. Para el movimiento se usa planner reactivo. En la Figura 2.24 se puede observar el procedimiento del auto-docking.

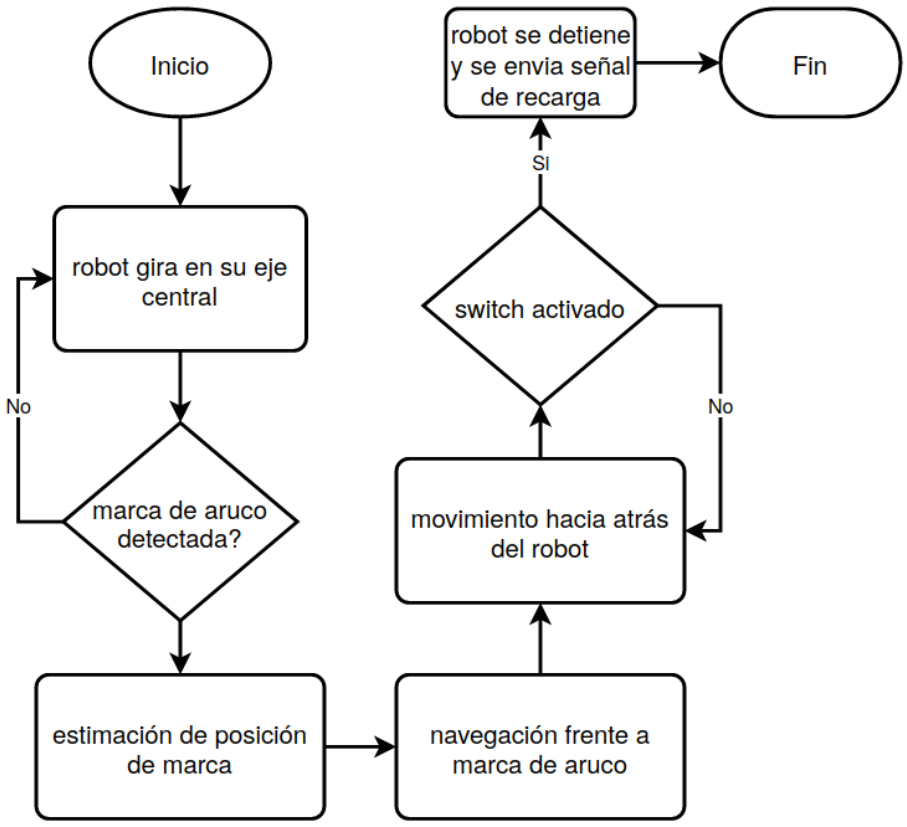


Figura 2.24 Diagrama de flujo para el procedimiento de auto-docking.

[14] https://wiki.ros.org/aruco_detect

Se generó un nodo de servidor de acción para el auto-docking. Sus entradas y salidas se encuentran en la Figura 2.25, donde $fiducials\_tf$ corresponde a la transformada de la marca de Aruco en base al mapa.

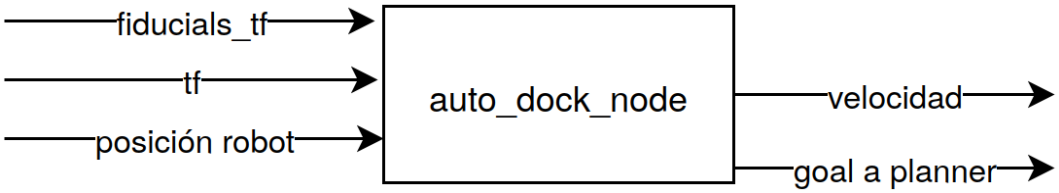


Figura 2.25 Nodo para auto-docking con sus entradas y salidas.

Como marca de Aruco, se usó una en específico que corresponde al número ocho. Esta marca se observa en la Figura 2.26.

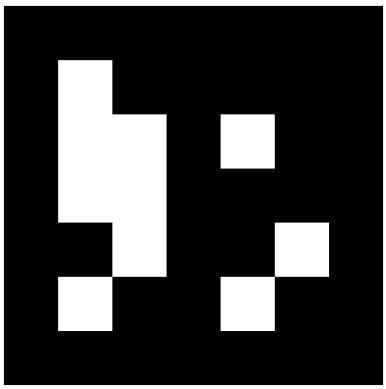


Figura 2.26 Marca de Aruco de ID #8.

2.9 Diseño de controlador de motores DC

Como controlador para los motores DC se consideró uno de tipo PID ya que es ampliamente usado en estas aplicaciones por su buen funcionamiento.

Se consideró lo plasmado en el diagrama de la Figura 2.27. Para el cómputo del controlador se generó un nodo en ROS para cada motor y que da como salida la potencia, un valor entre -127 y +127. El código de este nodo se encuentra en el Anexo 3. Adicionalmente se realizó la programación del ESP32 que se encuentra en el Anexo 8.

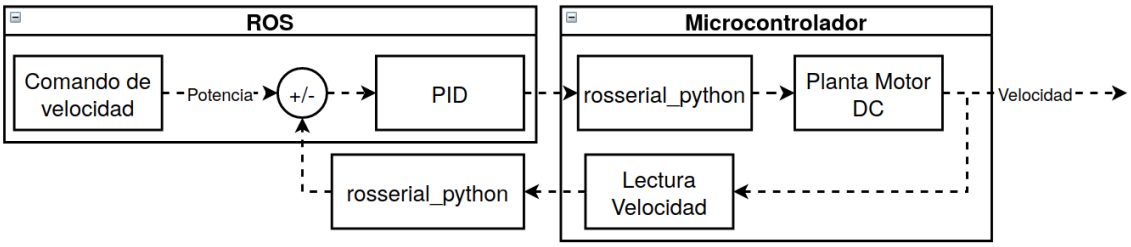


Figura 2.27 Diagrama de conexiones para diseño de controlador PID.

El controlador se diseñó a través de dos distintos métodos.

2.9.1 Método de Ziegler Nichols de lazo cerrado

Se aplicó el método de Ziegler Nichols de lazo cerrado como se especifica en la Figura 1.4. La Figura 2.28 se usó para obtener el valor de P_σ y así se calcularon las constantes para un controlador PID con los cuales se generó una gráfica posteriormente. El cálculo se encuentra en el Anexo 5.

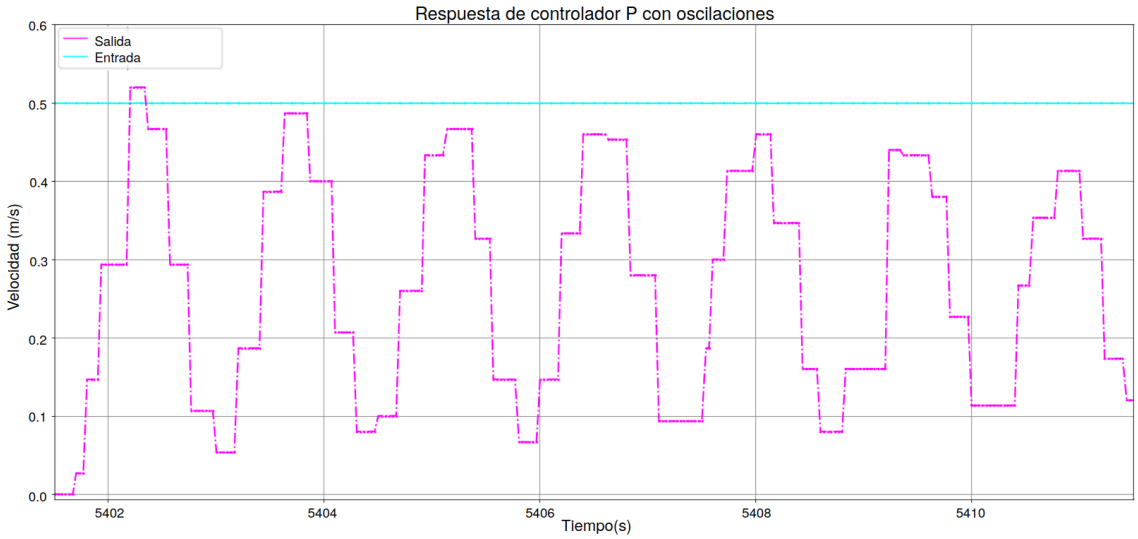


Figura 2.28 Gráfica de señal de salida para el uso del método de Ziegler Nichols.

2.9.2 Método heurístico

Con el anterior método no se obtuvo una respuesta totalmente aceptable y se usó el método heurístico de sintonización como se especifica en el marco teórico y en [18].

En este proceso se jugaron con las variables hasta obtener buenos resultados en las gráficas generadas.

2.10 Diseño e implementación de sistema de auto-recarga

La base de auto-recarga fue diseñada en el software OnShape. El diseño fue inspirado la estación de auto-recarga de Neobotix[15].

Se divide en dos partes: puerto de carga y estación de recarga. La diferencia está en que el puerto de recarga es el que se encuentra adherido al robot y hace la conexión a las baterías.

Para la unión de la salida del cargador y las baterías del robot, se usó puertos o conectores de tipo banana como se observan en la Figura 2.29.

Figura 2.29 Conectores de tipo banana macho y hembra.

Para la sujeción de todos los elementos, se modelaron abrazaderas dentro de las piezas. Todos los pernos escogidos fueron de tamaño M3. No se realizó un análisis para la selección del módulo y la cantidad de pernos ya que no soportaban altas cargas.

Para la implementación de las piezas, usó impresión 3D y el software gratuito Ultimaker Cura[16]. La configuración usada para la impresión 3D se observa en la Tabla 2.14.

[15] https://github.com/neobotix/neo_docking
[16] https://ultimaker.com/software/ultimaker-cura

Tabla 2.14 Configuración principal usada para impresión 3D.

Especificaciones de Impresión 3D	
Característica	**Especificación**
Altura de capa (mm)	0.2
Grosor de pared (mm)	0.8
Relleno (%)	20
Patrón de relleno	Cúbico
Temperatura de impresión (grados Celsius)	210
Velocidad de impresión (mm/s)	50
Retracción habilitada	Si
Enfriamiento	Si

Todas las impresiones se hicieron usando una impresora Anet A8 propia como se ve en la Figura 2.30.

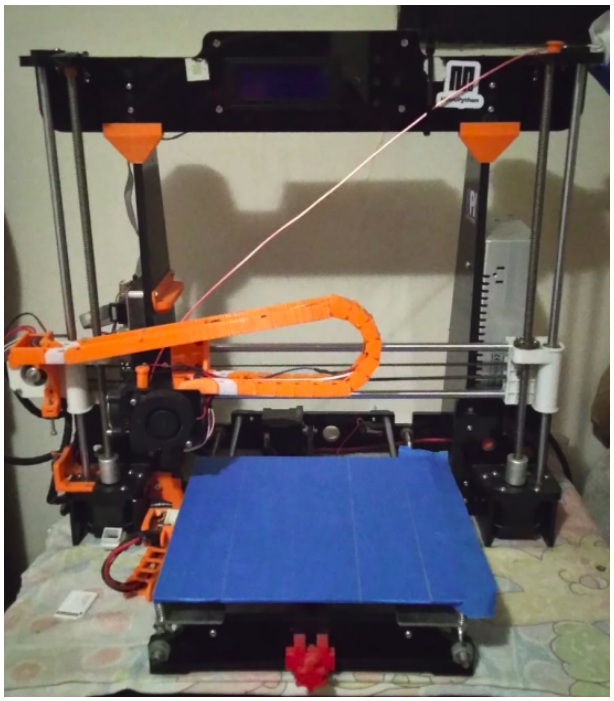


Figura 2.30 Impresora Anet A8.

2.10.1 Puerto de recarga de plataforma móvil

Como requerimientos del puerto de recarga se definió lo siguiente:

1. Alta tolerancia al ingreso de los puertos machos banana en el puerto de recarga.
2. Mecanismo sencillo y compacto.

3. Fácil instalación.

En la Figura 2.31 se observa una vista explotada del diseño. En el Anexo 12 se puede observar más a detalle los planos.

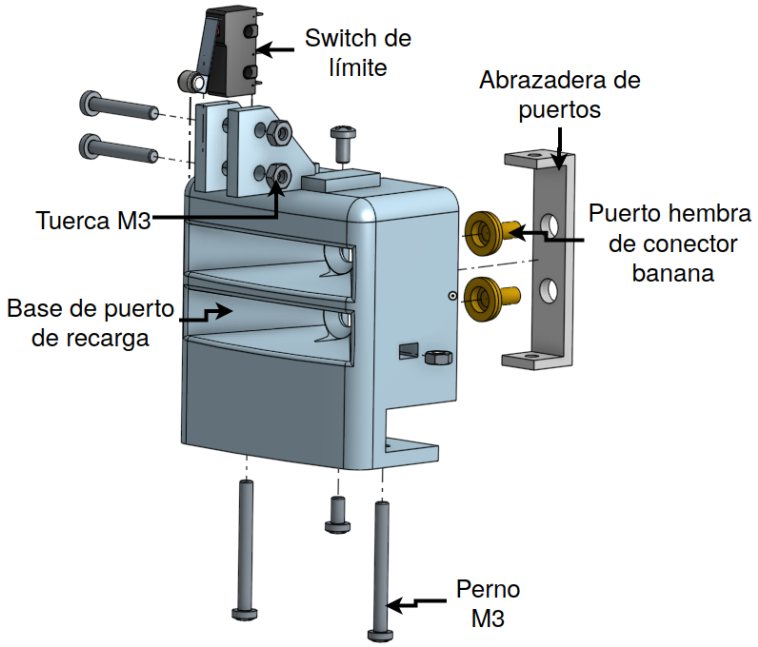


Figura 2.31 Vista explotada de ensamble de puerto de recarga de robot.

2.10.2 Estación de recarga

Junto al diagrama de la Figura 2.16, se consideraron los siguientes requerimientos para la estación de auto-recarga:

- En sus salidas debe tener los puertos machos de tipo banana.
- Tiene que haber un resorteo o amortiguamiento en los puertos machos.
- Debe ser capaz de encajarse en el puerto de recarga de la plataforma robótica, entrando en los puertos hembra banana.

En la Figura 2.32 se puede observar una vista explotada del diseño de la estación de auto-recarga. Los planos detallados se encuentran en el Anexo 11.

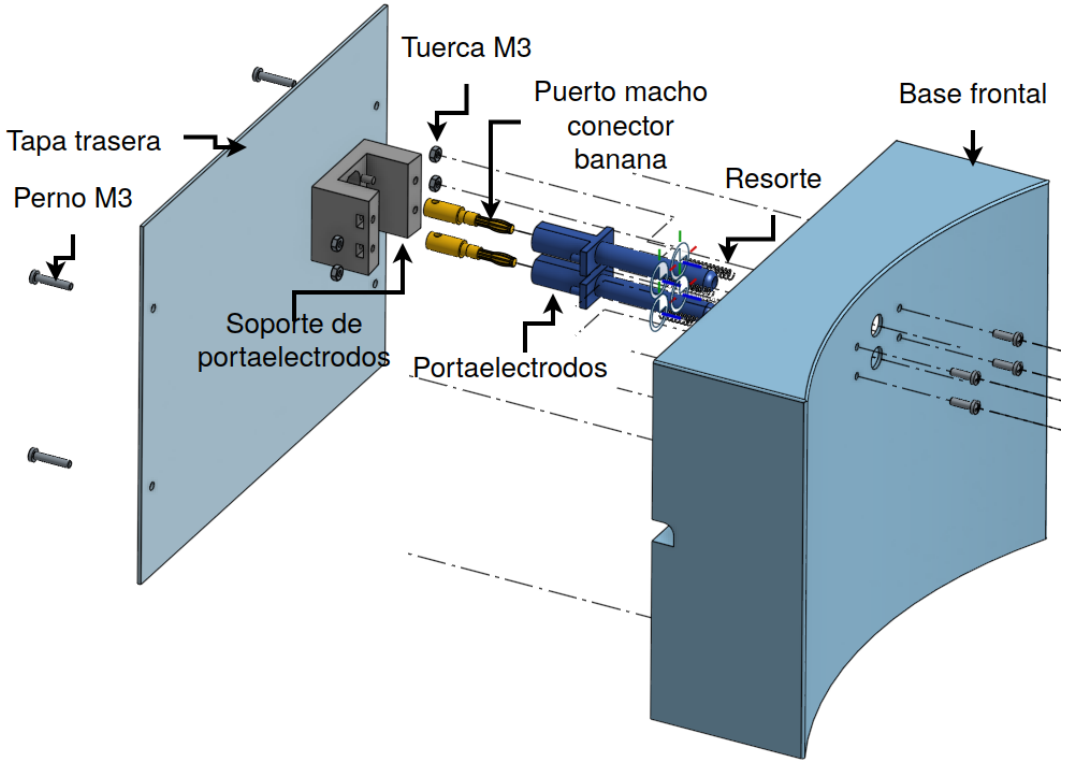


Figura 2.32 Vista explotada de ensamble de estación de recarga.

2.10.2.1 Análisis de esfuerzos

Para determinar si la estación de auto-recarga podría soportar el empuje del robot al conectar el puerto de recarga, se realizó un análisis de esfuerzo Finite Element Analysis (FEA) en Inventor Autodesk en base a la siguiente información:

- La velocidad de entrada a la estación es de $15cm/s$.
- El material de las piezas es Acido Poliláctico (PLA).

Como material, se ingresó en Inventor las características del PLA según lo encontrado en [18], estas se muestran en la Figura 2.33.

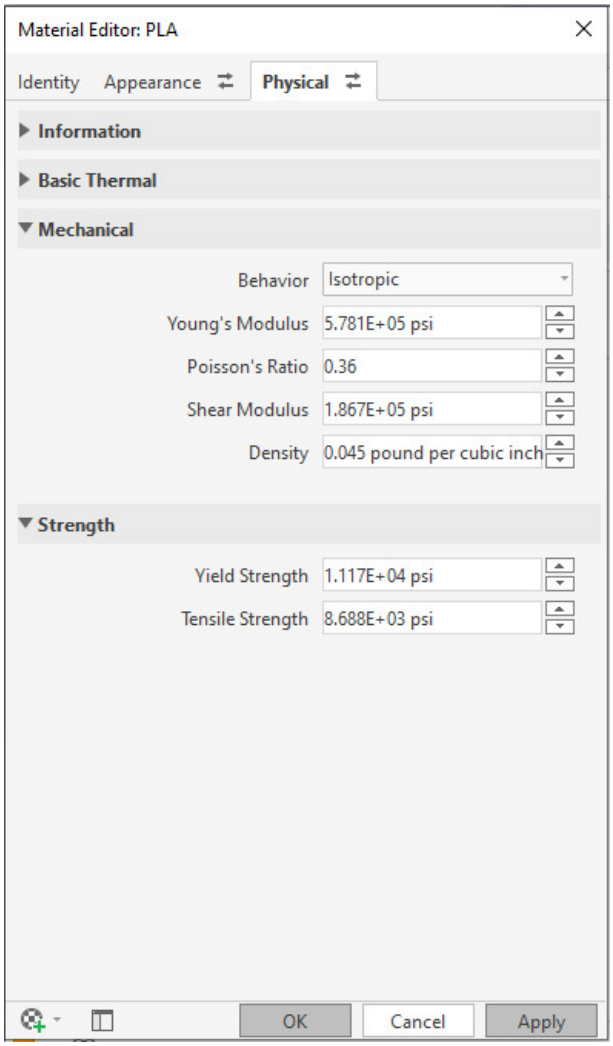


Figura 2.33 Características físicas del PLA como material en Inventor Autodesk.

En la simulación se consideró que las piezas tienen un relleno del 100 %. Aunque la estructura de las piezas se hace por capas, en [58] se encontró que el análisis de esfuerzos por simulación a uno experimental.

2.10.3 Nodo de activación de auto-docking

2.10.3.1 Medición de carga

Para medir la carga de las baterías, se propuso implementar un circuito de divisor de voltaje Conociendo que las entradas analógicas del ESP32 corresponde a 3.3V, se necesitó rebajar el voltaje de 12V.

Dado que la entrada digital del microcontrolador es de 12bits, se decidió asumir que el voltaje de las baterías podría llegar a 16V para evitar saturación. Así, se

realizaron los cálculos para el circuito como se muestra en el Anexo 6 en lo que se determinó que para la R_2 el valor de resistencia sería igual a 4.7kOhms, mientras que para R_1 el valor sería de 18kOhms.

La programación para la medición de voltaje se puede ver en el Anexo 8.

2.10.3.2 Nodo de activación

Este nodo de activación se implementó para recibir los datos de voltaje de las baterías, parar las acciones en caso de una descarga, y generar el proceso de la Figura 2.24. En la Figura 2.34 se observan las entradas y salidas que tiene el nodo que activa el proceso del docking.

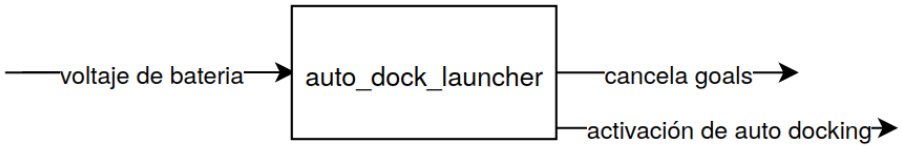


Figura 2.34 Conexiones para nodo de activación de auto-docking.

2.10.3.3 Control de estación de recarga

Para controlar la estación de recarga, se incluyó un microcontrolador con un relé según la Figura 2.16. El relé es activado con una señal booleana cuando el switch de límite del puerto de recarga es activado. La programación de el microcontrolador de la estación se encuentra en Anexo 7.

2.11 Construcción de robot

Finalizando las etapas anteriores, se realizó la construcción del robot con todos sus componentes físicos.

Primero se realizaron las conexiones electrónicas sobre el primer piso de la plataforma como se aprecia en la Figura 2.35.

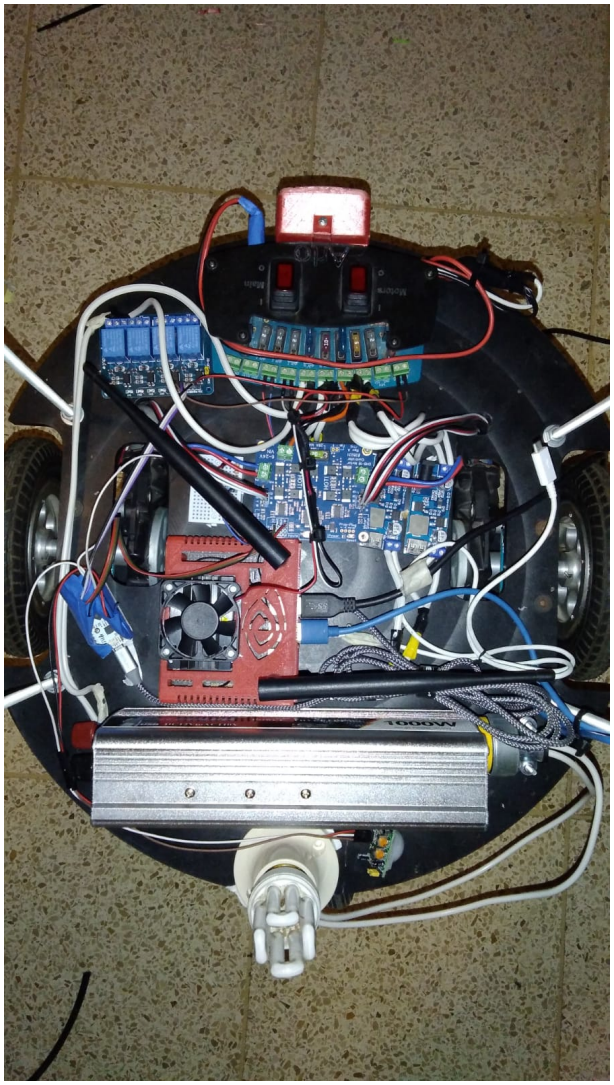


Figura 2.35 Implementación de conexiones eléctricas.

En la Figura 2.36 se puede observar la cámara T265 en el segundo piso del robot, el lidar y el puerto de recarga instalado.

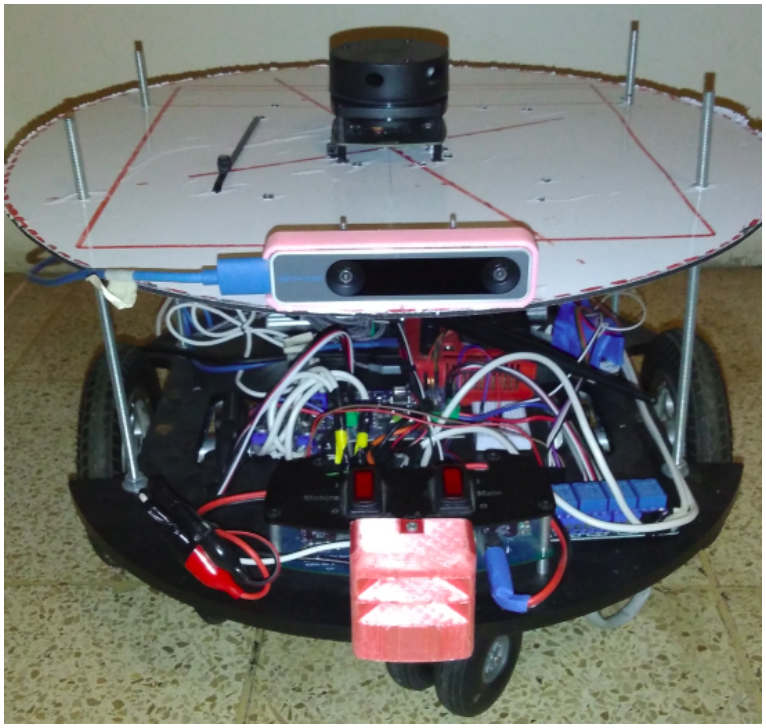


Figura 2.36 Instalación de cámara T265 y lidar en plataforma.

En la Figura 2.37 se muestra la estructura con los marcos de lámparas UV-C construida para el robot.

Figura 2.37 Implementación de marco de lámparas UV-C.

En la Figura 2.38 está la cámara D435i instalada su respectivo soporte que va en la solución.

Figura 2.38 Cámara D435i puesta en soporte.

En la Figura 2.39 está la estación de auto-recarga implementada. Para la implementación solo se imprimió la mitad de toda la base de recarga por falta de recursos. Como método de sujeción contra la pared se usó cinta doble faz que tiene un gran agarre con impresiones 3D.

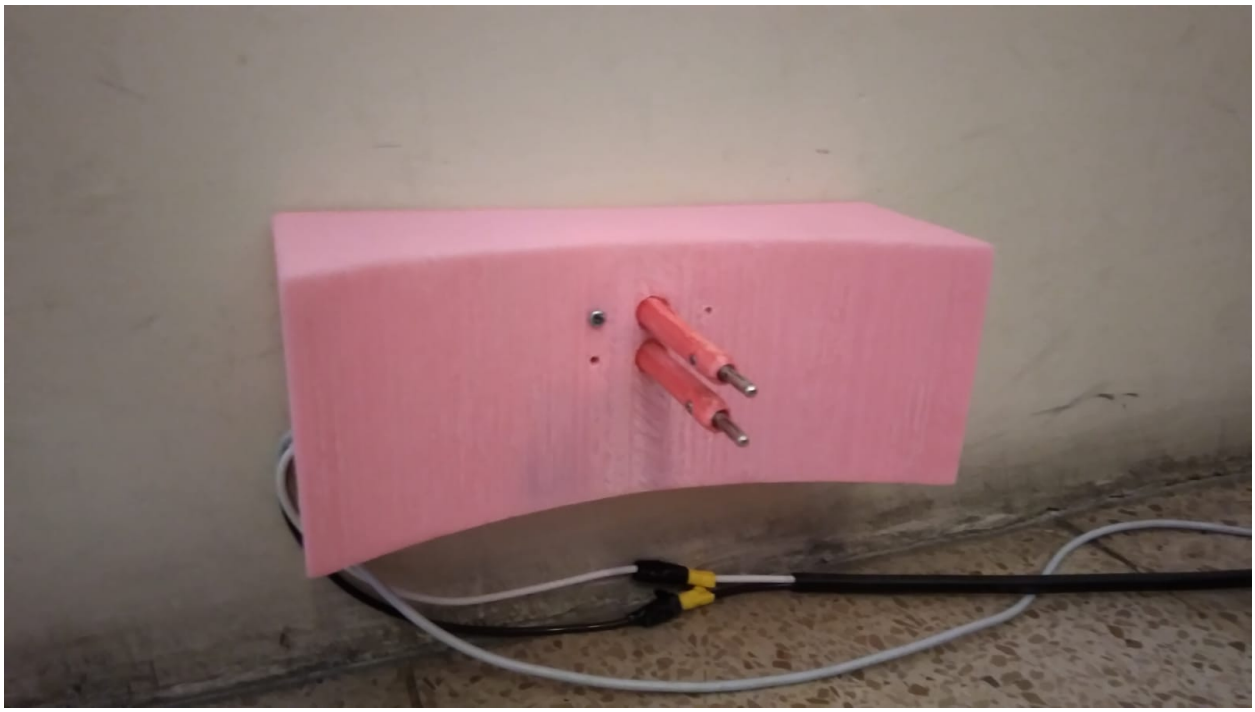


Figura 2.39 Estación de recarga de plataforma móvil implementada.

Finalmente, en la Figura 2.40 se puede observar el robot de desinfección totalmente

armado y terminado. Para este punto se decidió la palabra CoviBot como nombre del robot.

Figura 2.40 Construcción de robot de desinfección completa.

Con el robot totalmente construido se realizó una medición del peso para determinar si cumple con los requerimientos, ello se muestra en la Tabla 2.15.

Tabla 2.15 Masas de secciones de robot completas.

Masa de robot construido			
Sección	**Unidad**	**Masa (lb)**	**Masa total sección (lb)**
Ensamble superior – Lamparas UV-C y cámara D435	1	9.81	9.81
Ensamble inferior – Sensores, actuadores y controladores	1	25.52	25.52
Impresión 3D – Guías	8	0.21	1.71
Cubiertas de acero	4	1.00	4.00
Masa total de robot (lb)			41.04

También se hicieron mediciones y cálculos de corriente del robot para conocer un aproximado del tiempo de funcionamiento. Este se presenta la Tabla 2.16.

Tabla 2.16 Consumo energético de robot completo con el uso de las lámparas UV-C.

Consumo energético			
Componente	**Unidades**	**Corriente DC (A)**	**Corriente DC total (A)**
Robot sin focos	1	0.32	0.32
Lámparas UV-C	4	3.33	13.32
Corriente total de robot con focos (A)			13.64

Tomando en cuenta las tablas anteriores, se definieron las características y especificaciones del robot en la Tabla 2.17.

Tabla 2.17 Especificaciones de CoviBot finalmente construido.

Especificaciones de robot	
Centro de cómputo	Jetson Nano
Baterías	2 x 12V 7Ah
Dimensiones	242mmx242mmx1700mm
Tiempo de funcionamiento	1 hora
Seguridad	Detección de personas en frente
Autonomía	Navegación y recarga
Peso	41 libras
Velocidad máxima	22cm/s
Terreno	Interiores., piso liso, cemento, poco inclinado
Percepción	Mapeo y localización
Desinfección	4 x lámpara UV-C 40 Watts
Conexión	WiFi
Cinemática	Diferencial de 2 llantas
Software	Robot Operating System

CAPÍTULO 3

3. RESULTADOS Y ANÁLISIS

3.1 Simulación

En esta subsección se muestran los resultados obtenidos a partir de la simulación.

3.1.1 Torque en Motores

En la Figura 3.1 se observa el torque desarrollado cuando se acelera al robot a su velocidad máxima de 22cm/s. Se denota que el pico es de aproximadamente 3.4Nm. Como se especifica en la Tabla 2.2, el máximo torque que soportan los motores es hasta 9.6Nm. El torque medido es casi tres veces menos al permisible. Esto muestra que no existe ningún problema en el movimiento del robot por la carga que lleva.

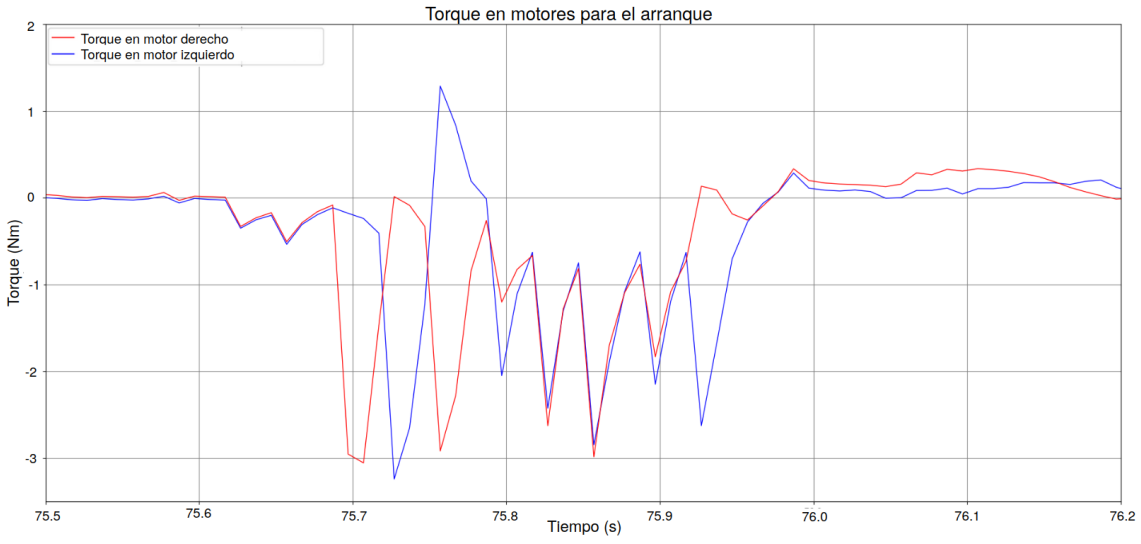


Figura 3.1 Torque en motores de robot al arranque hacia el frente.

En la Figura 3.2 se puede observar el torque desarrollado en un movimiento de arranque de reversa. De igual manera se observa que no hay un torque mayor en comparación con el permisible.

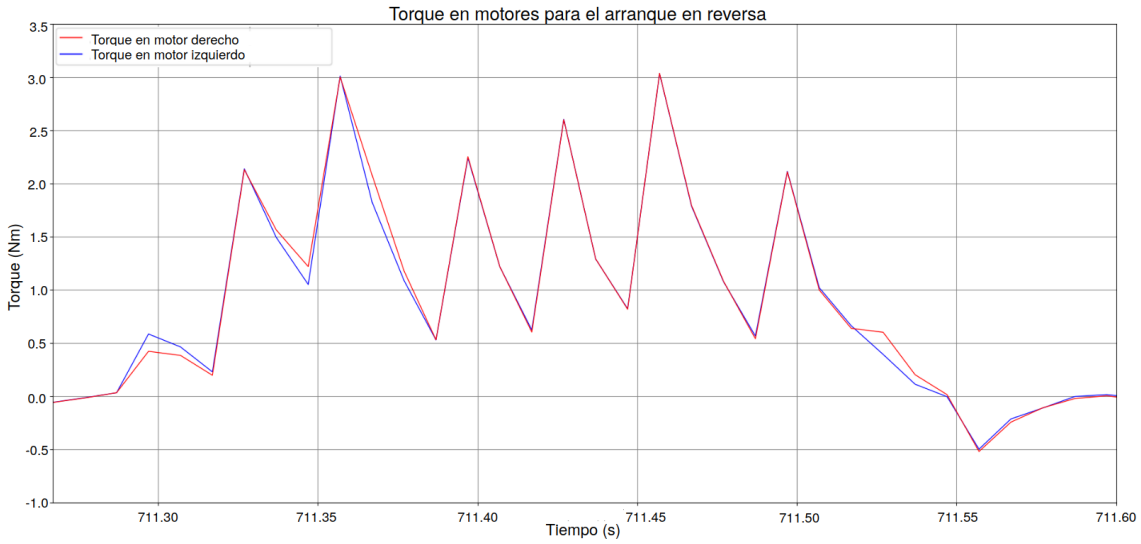


Figura 3.2 Torque en motores de robot al arranque en sentido de reversa.

En la Figura 3.3 se muestra el torque en un movimiento giratorio de arranque.

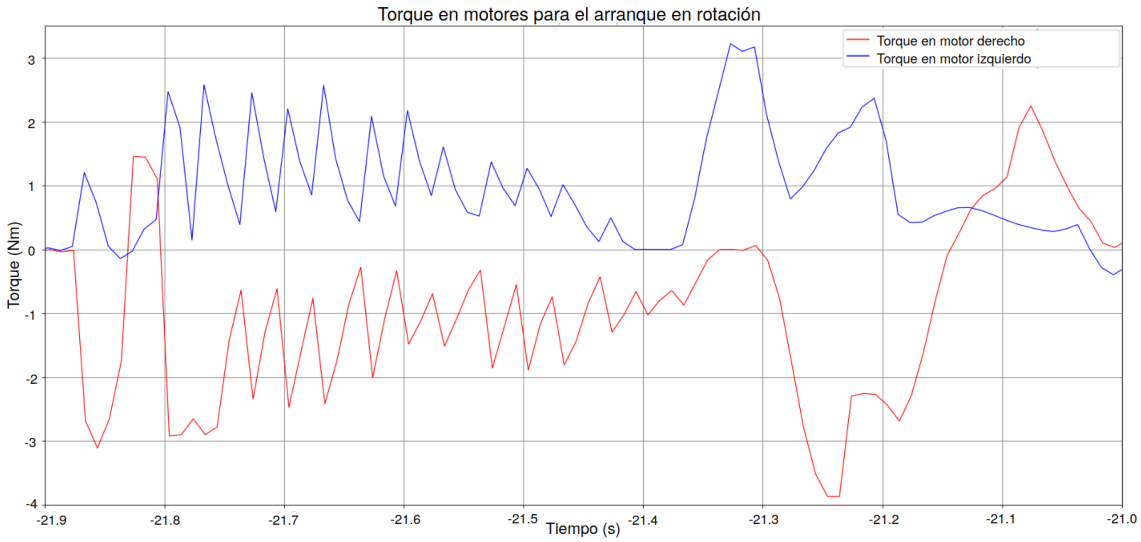


Figura 3.3 Torque en motores de robot al arranque en giro.

Se determina que se usa máximo el 40.62 % de la capacidad de torque de los motores. Esto asegura que no serán forzados y que además no demandan corriente fuera de lo normal, dando así un correcto funcionamiento.

3.1.2 Mapeo

En la Figura 3.4 se observa el resultado de mapeo en simulación. Según la silueta del mapa, los bordes demuestran tener una excelente definición. Las esquinas están muy bien especificadas y el grosor de las paredes es pequeño y nítido. En general corresponde a un mapa muy bien realizado.

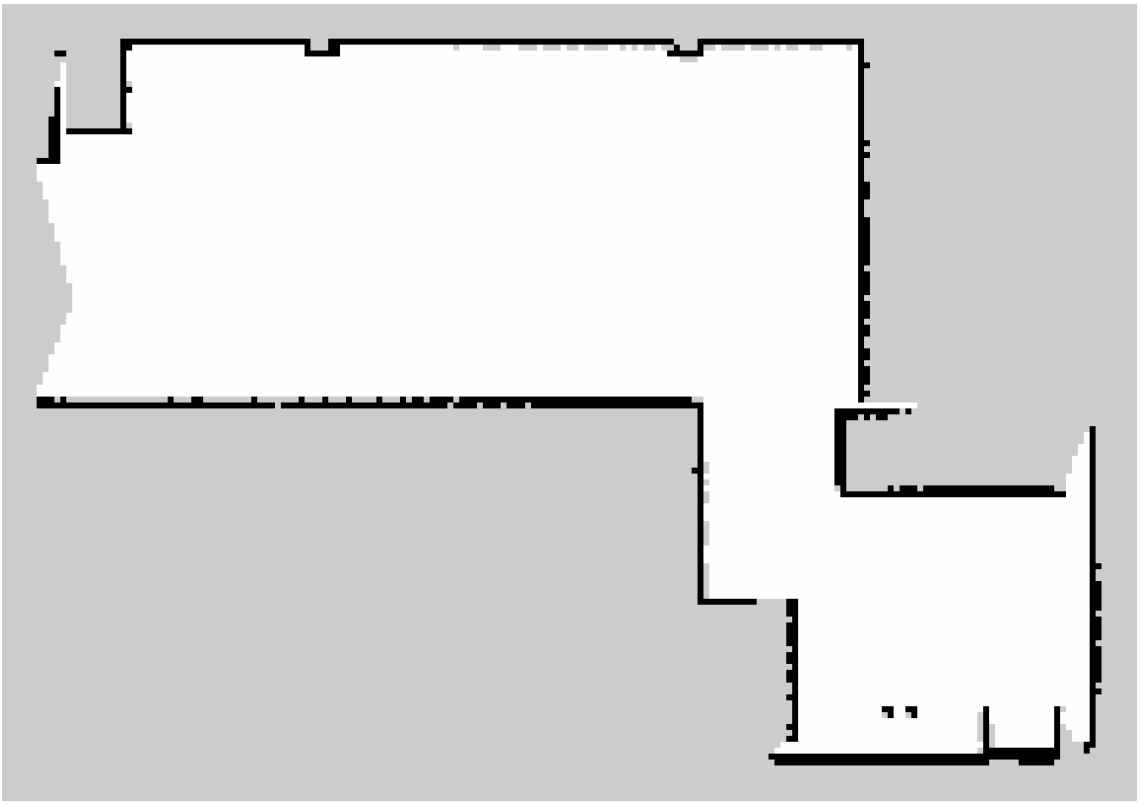


Figura 3.4 Mapa obtenido en simulación por Cartographer ROS.

3.1.3 Localización

En la Figura 3.5 se puede observar como la percepción del robot del entorno no se encuentra alineado con el mapa inicialmente. Este es un robot no localizado.

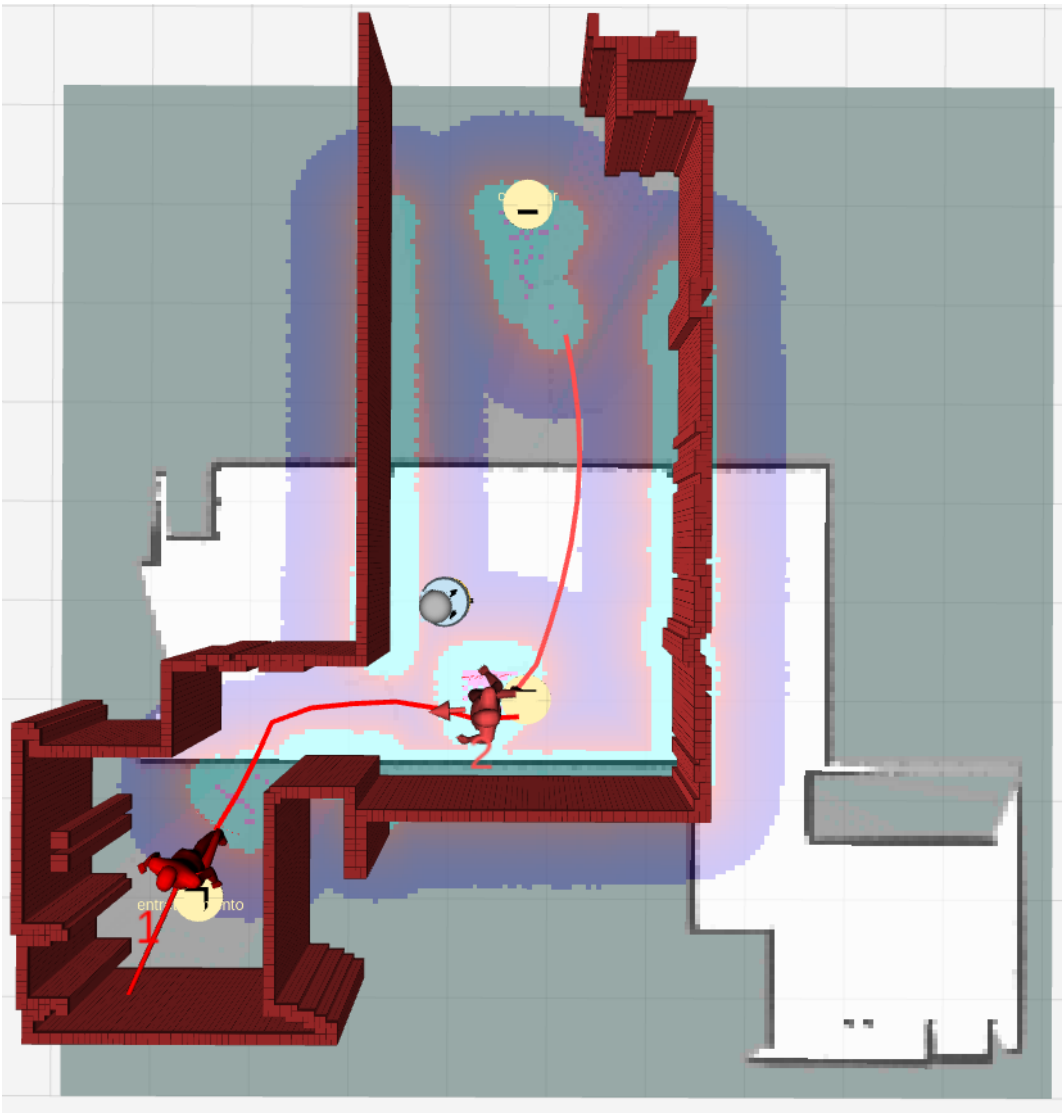


Figura 3.5 Robot no localizado en el espacio simulado.

Después de un tiempo y con un poco de movimiento, Cartographer ROS localiza al robot y se ajusta la percepción del mismo junto con el mapa como se ve en la Figura 3.6. En este caso, la silueta de las paredes se encuentran correctamente alineadas con el mapa.

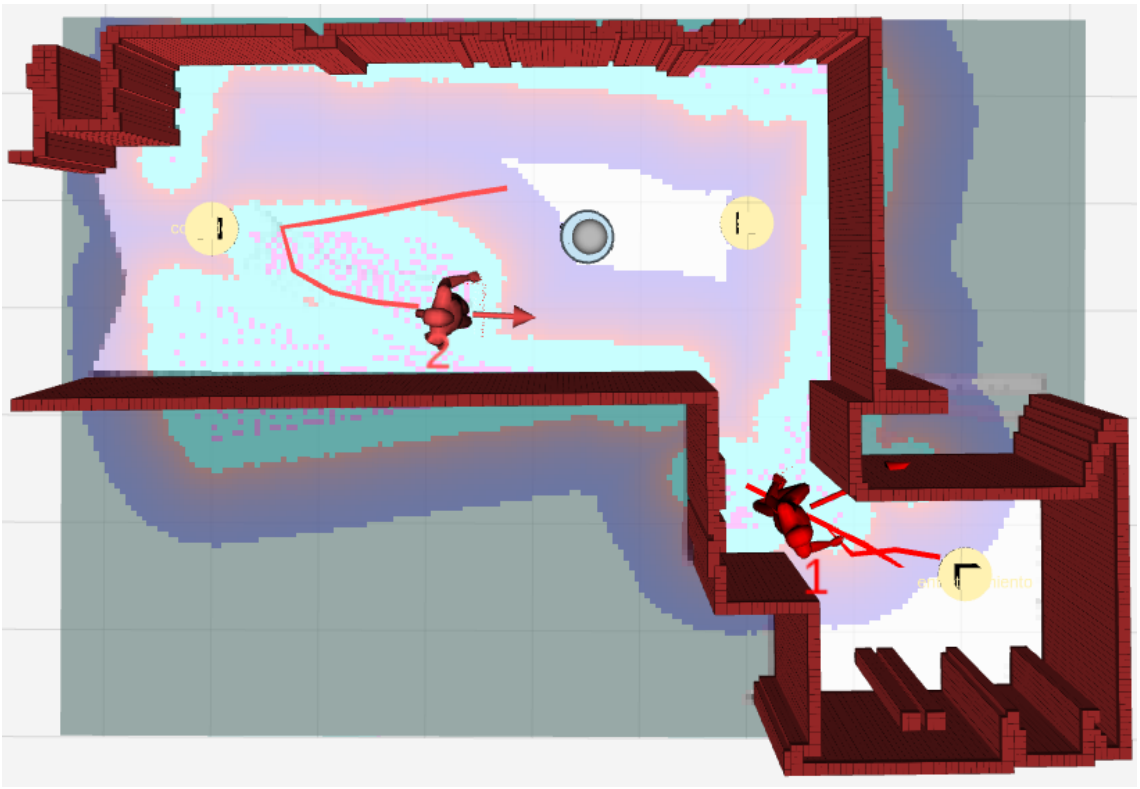


Figura 3.6 Robot localizado en el espacio simulado.

En la simulación se observó que este es capaz de localizar al robot incluso sin tener que moverse. Se observó que se localiza de manera correcta con el mapa, y que el tiempo que tarda en hacerlo es aproximadamente menor a 20 segundos.

3.1.4 Navegación Por DWA Planner

Se obtuvo la trayectoria del movimiento del robot desde un punto inicial hasta un punto final. En la Figura 3.7 se puede observar el punto de partida (A) del robot hacia otro punto (B). En la imagen, se muestra la trayectoria trazada por DWA Planner (líneas verdes).

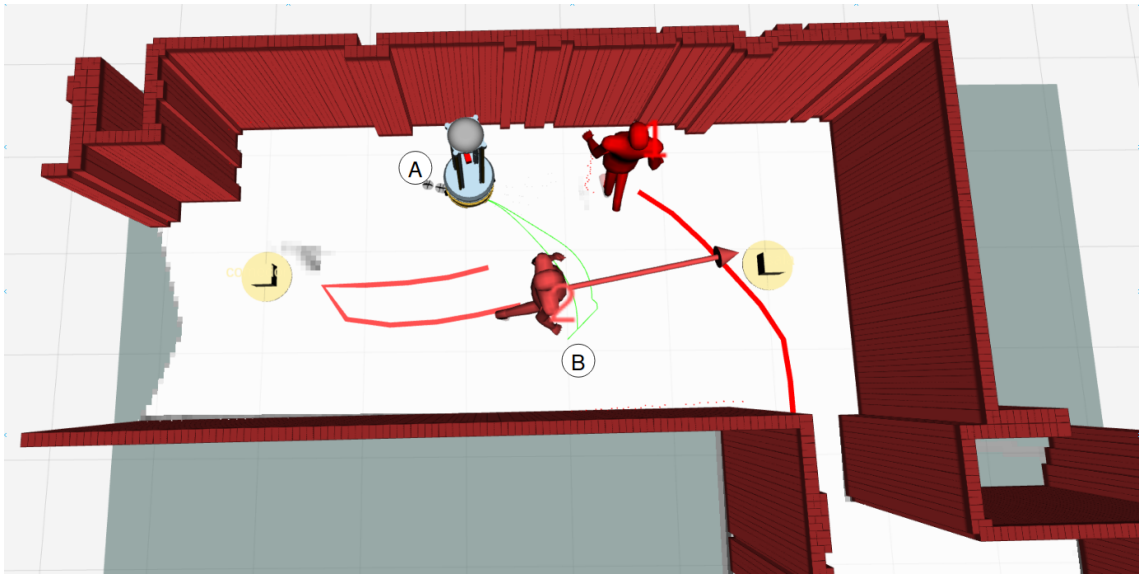


Figura 3.7 Posición inicial del robot antes de la navegación.

En la Figura 3.8 se ve al robot a la mitad de su trayectoria. La trayectoria se enmarca por los puntos negros en el centro de la imagen.

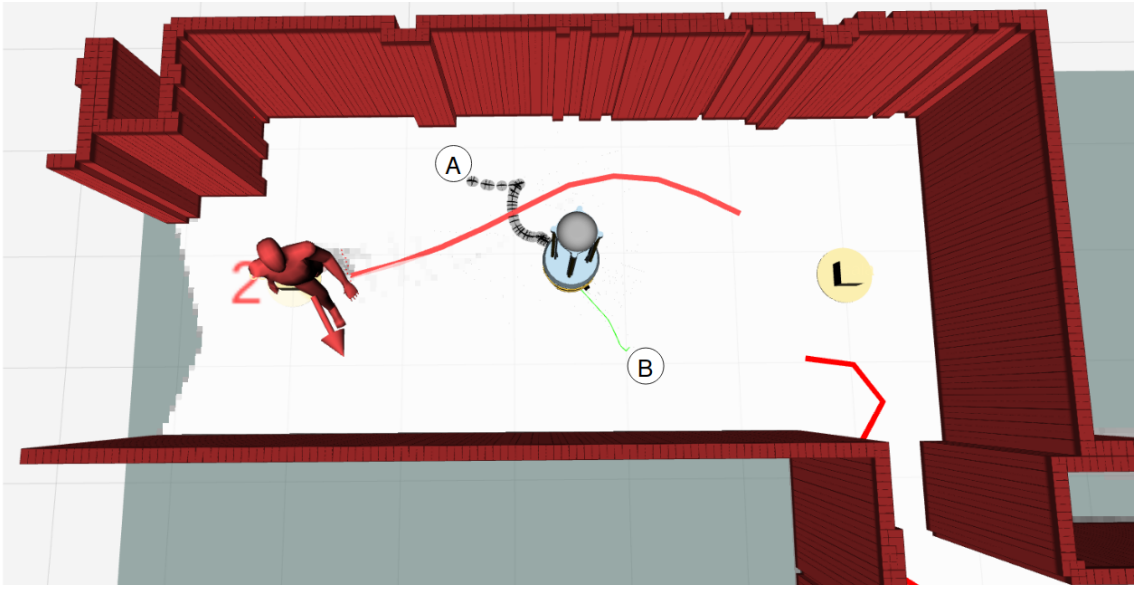


Figura 3.8 Posición media del robot en su navegación.

En la Figura 3.9 se ve la llegada del robot a su punto final (B).

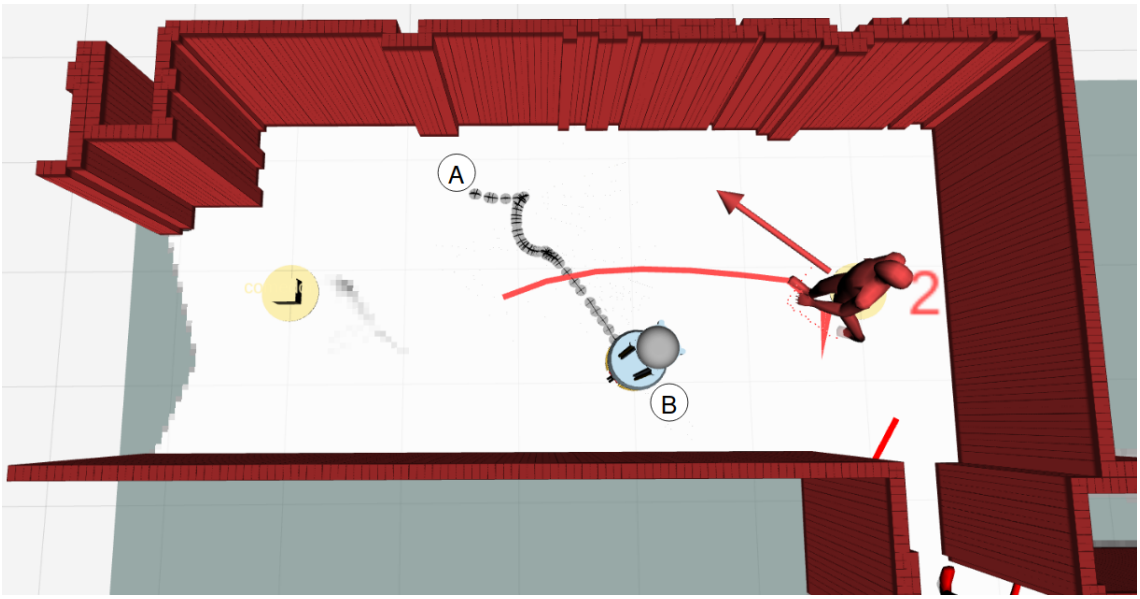


Figura 3.9 Posición final del robot en su navegación.

En la simulación el robot demostró ser capaz de navegar efectivamente evitando obstáculos incluso dinámicos. Como se observa en el inicio de la trayectoria un agente se encontraba cerca del mismo, por lo que el planner generó una trayectoria curva para esquivar al agente.

3.1.5 Navegación Por Modelo De Fuerzas Sociales

En la Figura 3.10 se observa la trayectoria tomada por el robot usando el SFM. El movimiento se define por los puntos negros en la imagen, el robot se mueve desde el punto A al B.

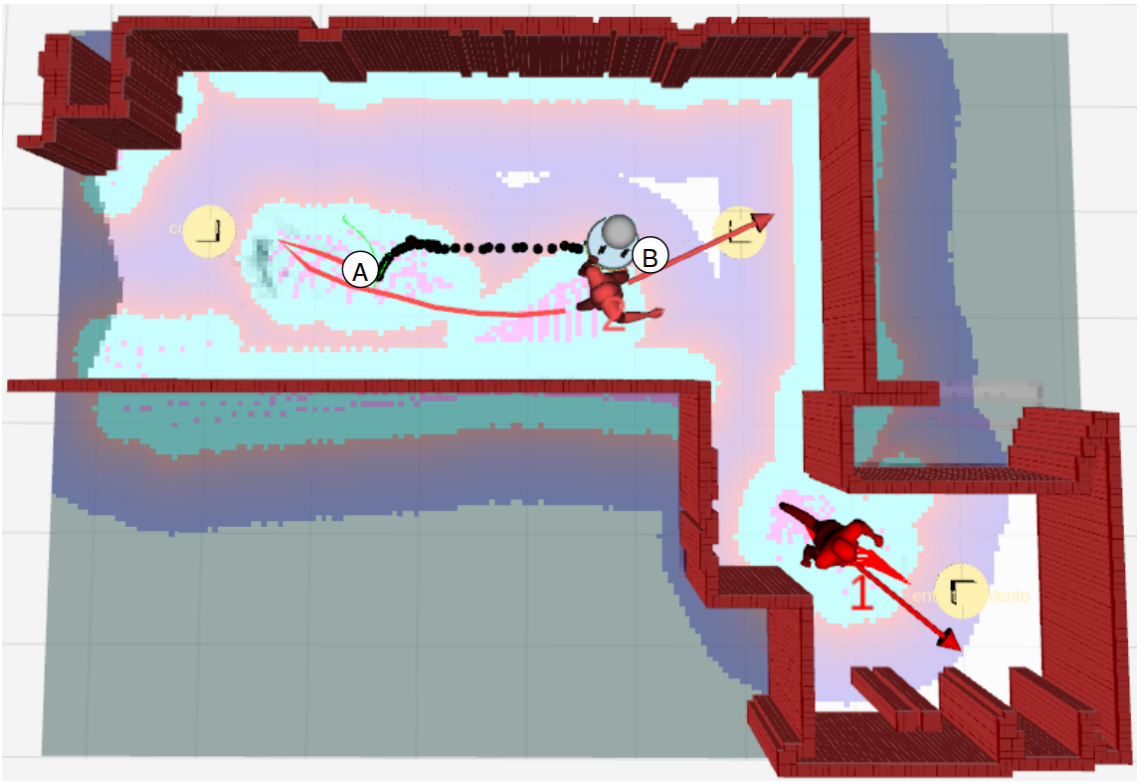


Figura 3.10 Trayectoria de robot por el Modelo De Fuerzas Sociales.

El algoritmo incluido para el movimiento del robot por el SFM muestra que es capaz de llevarlo entre distintos puntos de manera efectiva. No se observan oscilaciones en el movimiento y se da una maniobra de giro. En la imagen anterior se observa un agente cerca del robot debido a un inconveniente de `pedsim_ros` en la simulación.

3.1.6 Detección de Marcas Aruco Y Movimiento de auto-docking

En la Figura 3.11 se puede observar como en la simulación se detecta la marca de aruco por el paquete de `aruco_detect`. En este caso se está detectando el ID 8. La marca se detecta aproximadamente hasta una distancia de dos metros.

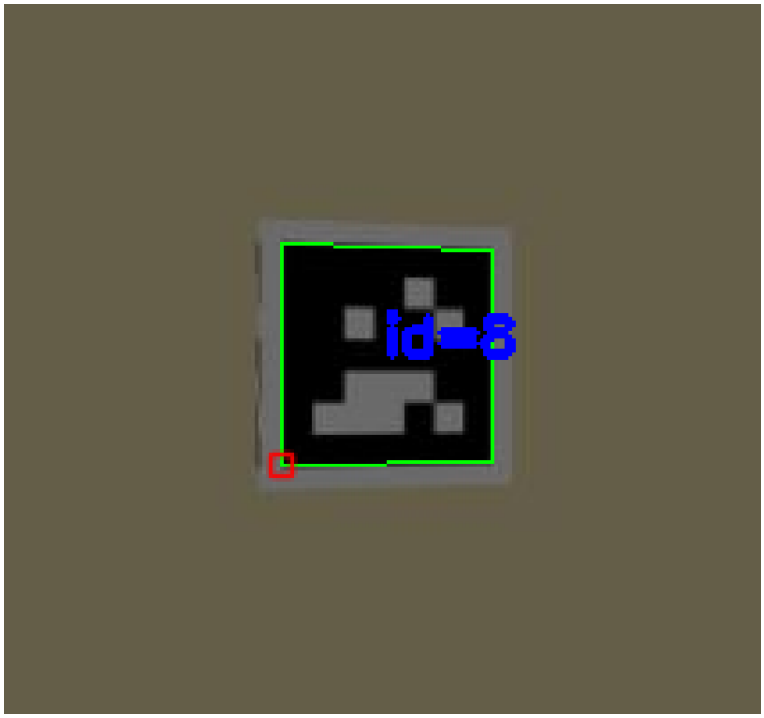


Figura 3.11 Detección de marca de aruco de identificador # 8.

En la Figura 3.12 se puede observar el movimiento del robot desde su punto de partida A hasta donde se encuentra la marca de aruco para la recarga en la marca B. Se puede ver que el robot se coloca inicialmente frente a la marca de aruco para luego moverse de reversa.

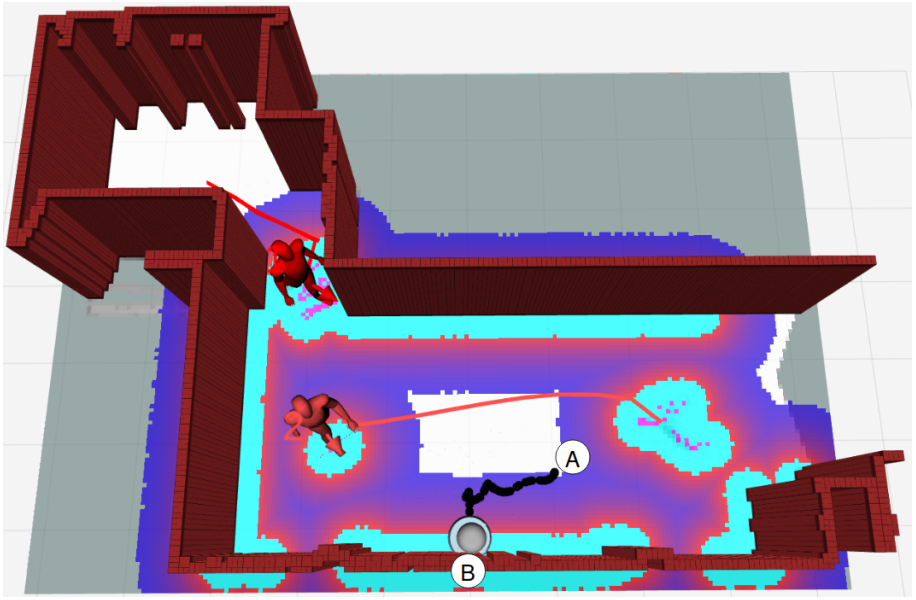


Figura 3.12 Movimiento para la acción de auto-docking en simulación.

3.2 Controlador Para Motores DC

3.2.1 Ziegler Nichols

En la Figura 3.13 se observa la respuesta obtenida según los valores que se obtuvieron para el controlador PID mediante este método en el Anexo 5.

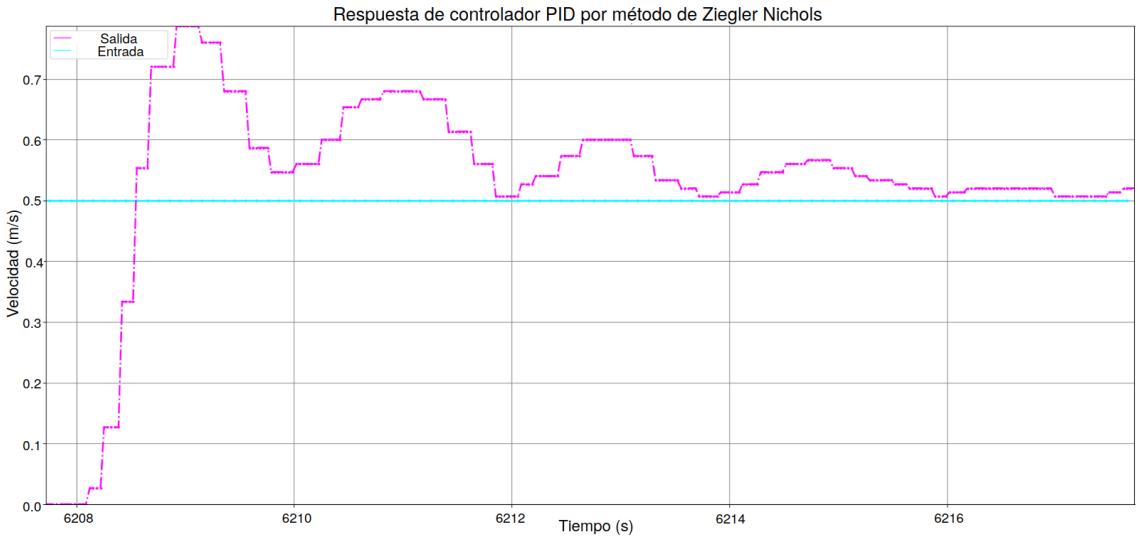


Figura 3.13 Controlador PI diseñado por método de Ziegler Nichols.

Se denota de que la respuesta no es muy buena, a pesar de que se estabiliza, toma mucho tiempo y tiene un overshoot muy alto. Además genera oscilaciones grandes en la salida, posteriormente se decidió seguir con el método heurístico.

3.2.2 Método heurístico

En la Figura 3.14 se muestra la mejor señal de salida como resultado del controlador tan solo usando la constante proporcional con un valor de 60. Se identifica que el tiempo muerto en la respuesta es de aproximadamente 0.5 segundos. Este tiempo es bastante aceptable considerando que las llantas tienen generalmente un movimiento lento.

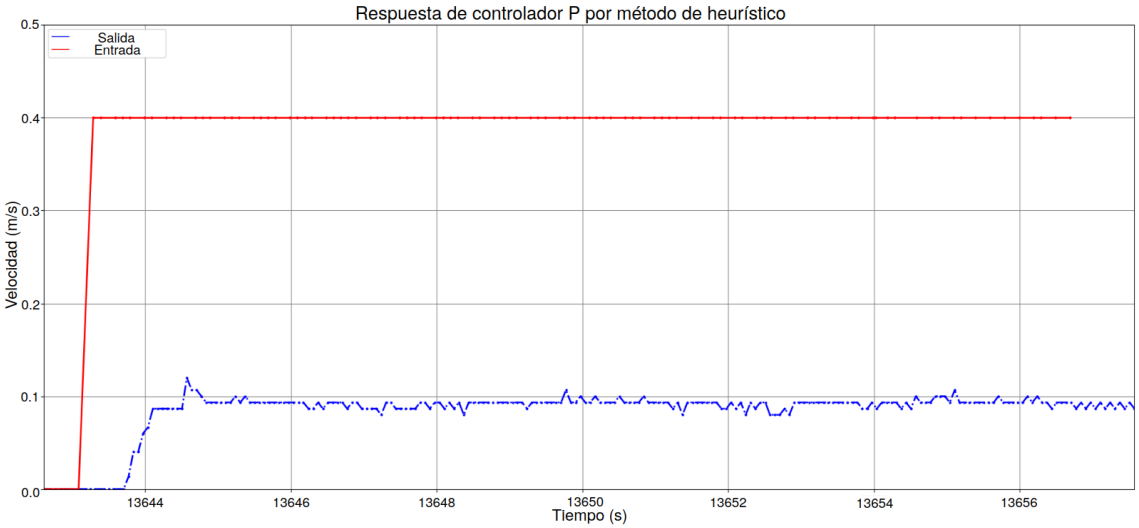


Figura 3.14 Controlador con solo constante proporcional de 60.

En la Figura 3.15 se observa la gráfica del controlador PI en la que se ha agregado la constante integral con el fin de eliminar el error de estado estacionario. Esta respuesta representa al mejor resultado mediante el método heurístico con un K_P y K_I de 65 y 160 respectivamente.

La gráfica demuestra una buena respuesta a la salida, el tiempo muerto que se dispone se da por el accionamiento mismo de los motores en la comunicación serial. Sin embargo, en la implementación se dan cambios de velocidades poco bruscos de manera que no representa problema en su funcionamiento.

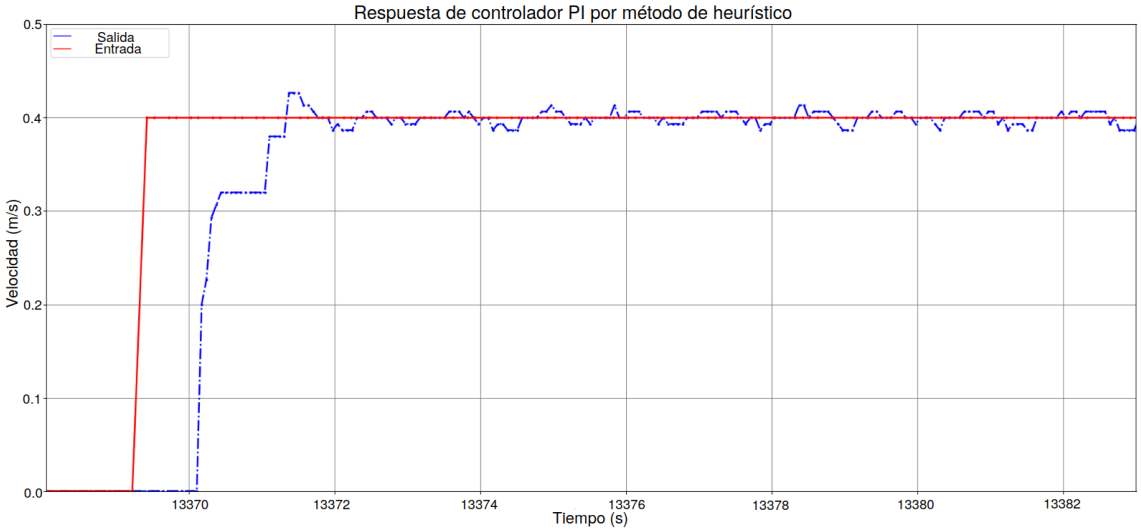


Figura 3.15 Controlador con solo constante proporcional de 65 e integral de 160.

No se implementó un controlador PID debido al ruido que se experimenta en las pruebas reales.

3.3 Sistema de auto-recarga

3.3.1 Análisis de esfuerzos

3.3.1.1 Soporte de portaelectrodos

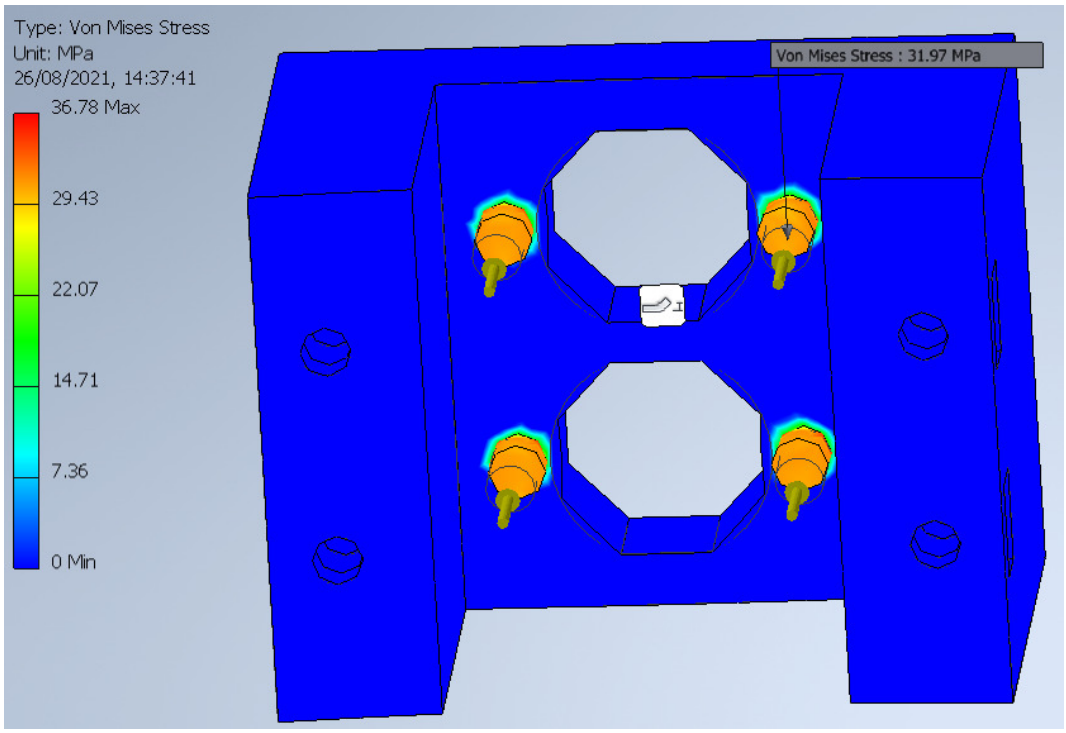


Figura 3.16 Esfuerzo de Von Mises para el soporte de portaelectrodos.

Como se observa en la Figura 3.16, el máximo esfuerzo desarrollado corresponde a 36.78MPa en compresión. Haciendo referencia a la Figura 2.33, podemos ver que el esfuerzo a la fluencia del PLA es de 77MPa. Por lo tanto no se muestra ningún problema por los esfuerzos experimentados.

En la Figura 3.17, se puede ver que el desplazamiento máximo en la pieza es de 0.05mm, como este es un desplazamientos hacia adentro de la pieza, es bastante insignificante sobre la pieza.

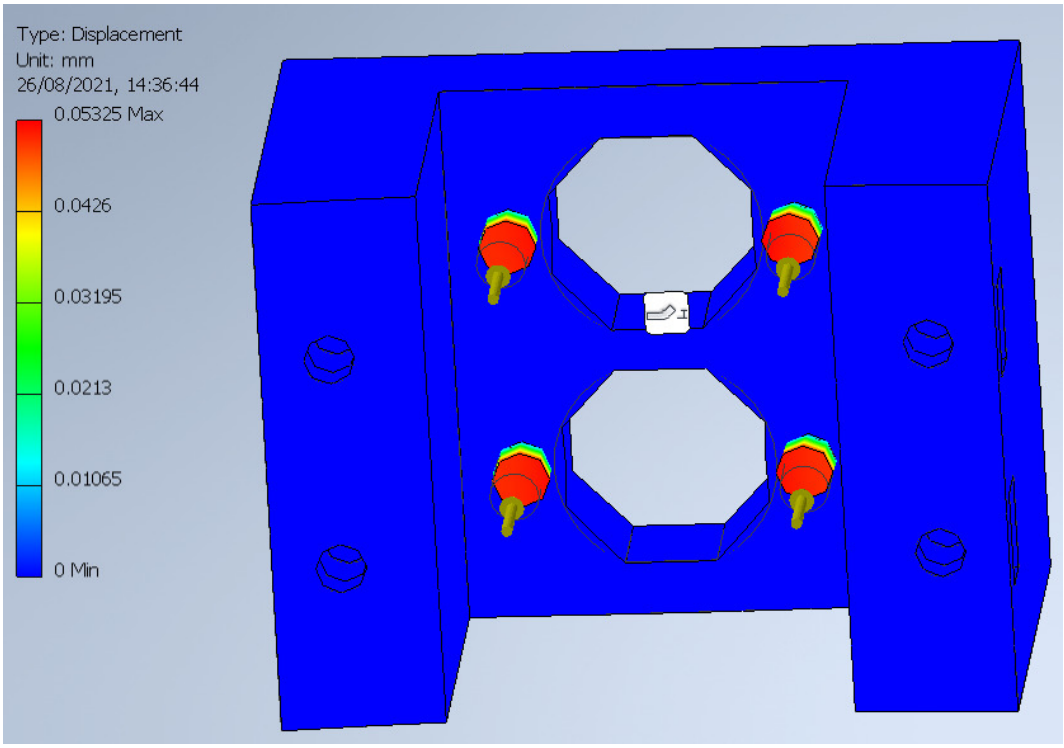


Figura 3.17 Desplazamiento en soporte de portaelectrodos.

Los dos resultados anteriores son corroborados por la Figura 3.18, que demuestra que el factor de seguridad mínimo en toda la pieza es de 2.09. Esto podría ser mayor dado que en la experimentación se usó finalmente una velocidad de 4cm/s para el movimiento de auto-docking.

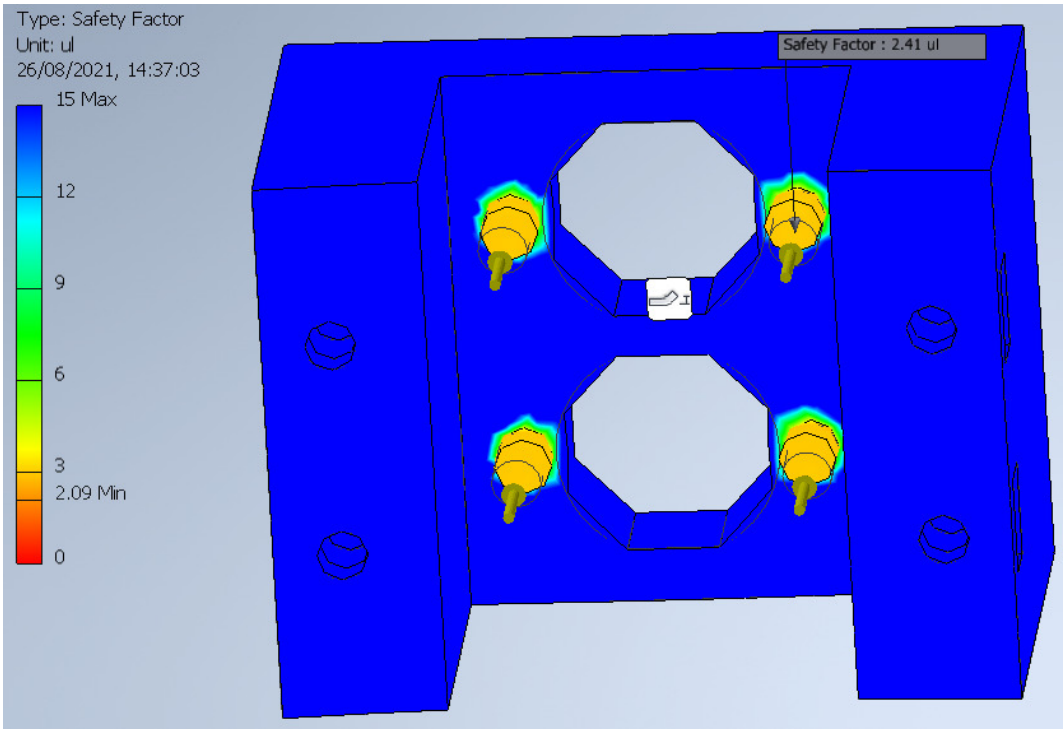


Figura 3.18 Factor de seguridad de soporte de portaelectrodos.

3.3.1.2 Portaelectrodos

En la Figura 3.19 podemos ver que el esfuerzo máximo en la pieza de portaelectrodos tiene el mayor esfuerzo en sus topes posteriores, igual a 59.12MPa. En este caso, el esfuerzo es muy cercano al límite.

A pesar de que para la simulación se usó una velocidad tres veces más alta, se encontró que esa parte de la pieza podría fallar. Esta podría ser reforzada con el uso de escuadras en los filos que redondean la pieza.

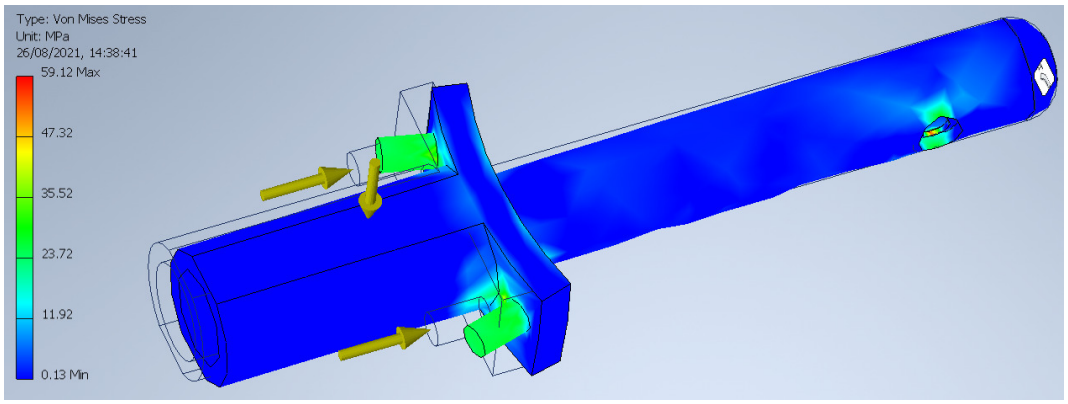


Figura 3.19 Esfuerzo de Von Mises para del portaelectrodo.

El resultado anterior es corroborado en la Figura 3.20, donde se observa un desplazamiento del 0.5mm. Esta es una deformación bastante significante. En el caso de que el robot ingrese a la estación de auto-recarga a la velocidad de 15cm/s, estas piezas podrían fallar.

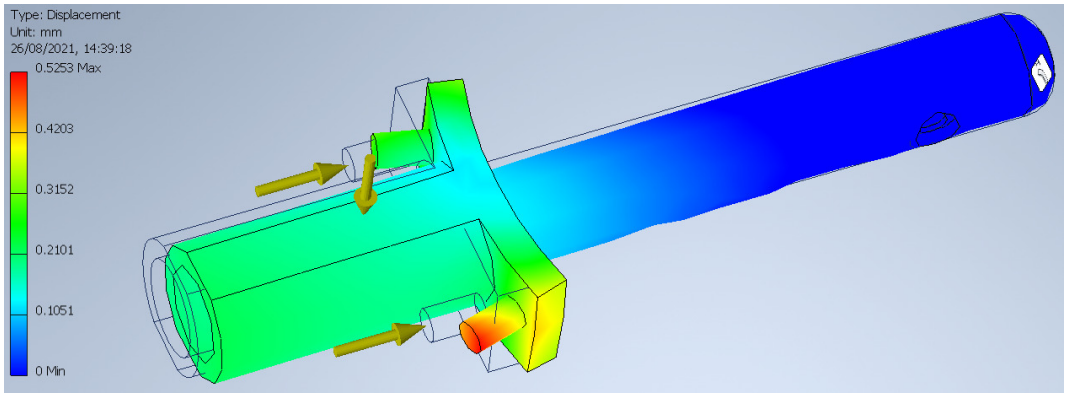


Figura 3.20 Desplazamiento en soporte del portaelectrodo.

Se dispone de un factor de seguridad bastante bajo de 1.3 en los soportes para los resortes como se observa en la Figura 3.21. Está claro que esta pieza necesita refuerzos y que podría romperse con altas velocidades.

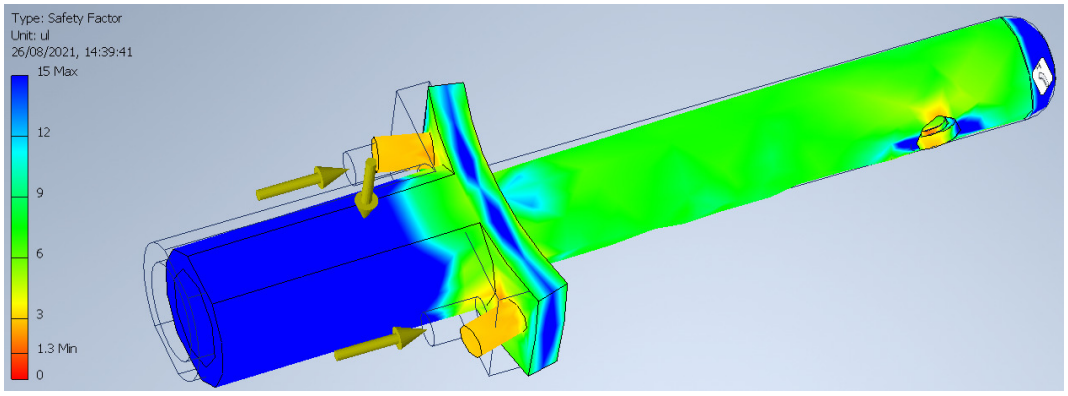


Figura 3.21 Factor de seguridad del portaelectrodo.

3.3.1.3 Base frontal

Como se presenta en la Figura 3.22, el esfuerzo máximo es de 324.5MPa principalmente en la zona de sujeción de los pernos para la pieza de soporte de portaelectrodos. Es aproximadamente cuatro veces mayor al que puede soportar el material. Definitivamente a tal velocidad esta pieza falla. Esto se debe a que en esa zona no hay ningún tipo de sujeción contra las paredes y el grosor es de tan solo 1.5mm.

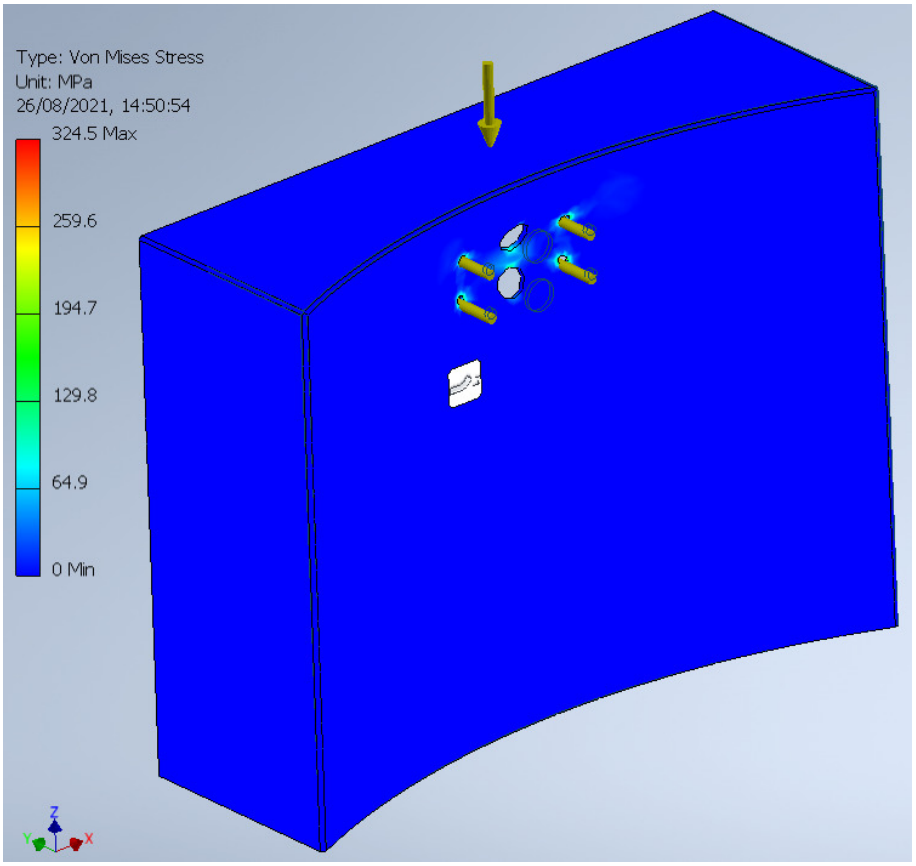


Figura 3.22 Esfuerzo de Von Mises para la base frontal de la estación de recarga.

Viendo la Figura 3.23, podemos notar que existe también una grave deformación en el material, de hasta 6.85mm. Este valor expresa que toda esta zona de la pieza podría romperse.

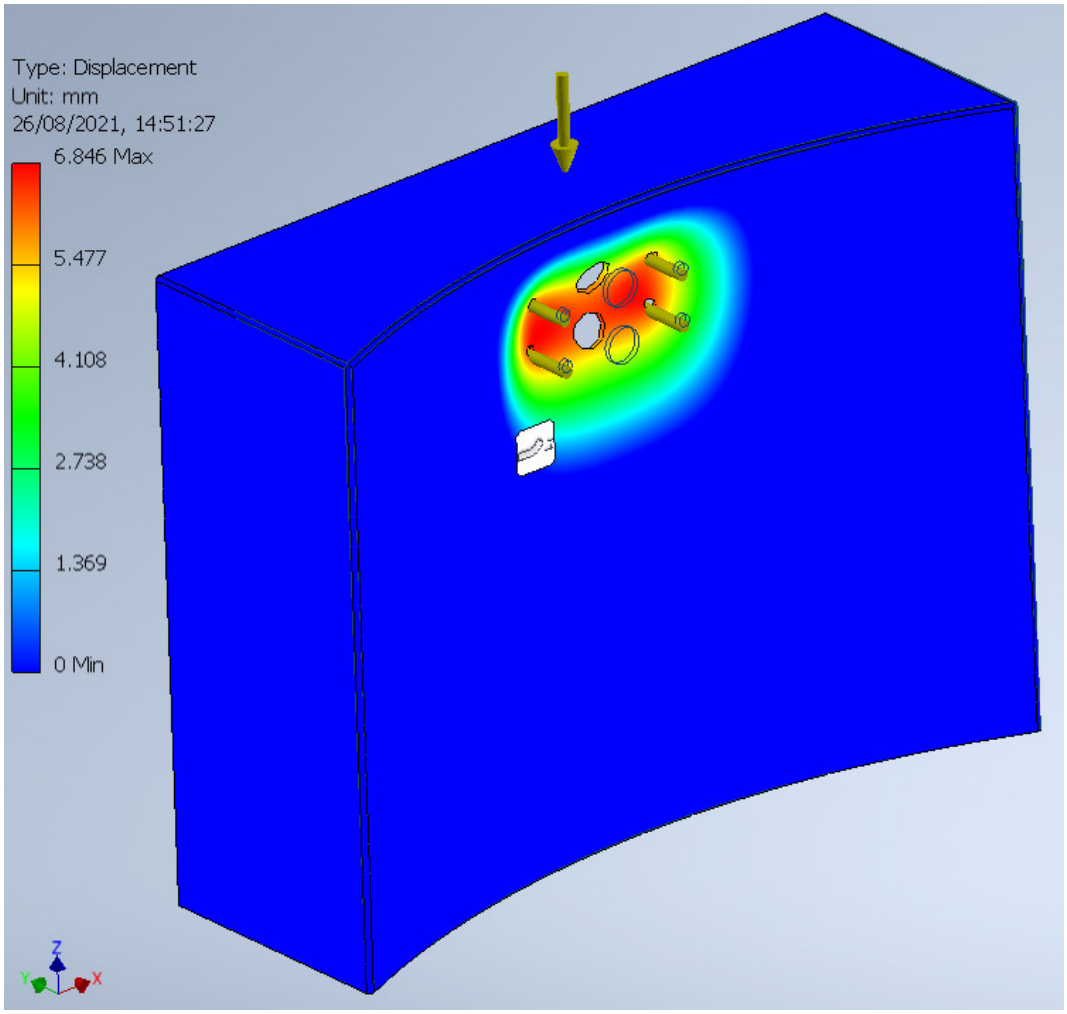


Figura 3.23 Desplazamiento en la base frontal de la estación de recarga.

En la Figura 3.24, el factor de seguridad mínimo en la estación es de 0.24. Uno de los cambios que se podrían realizar para aumentar el factor de seguridad, es aumentar el grosor de las paredes o al igual que en el portaelectrodos, agregarle soporte de escuadras que se sujeten contra las paredes.

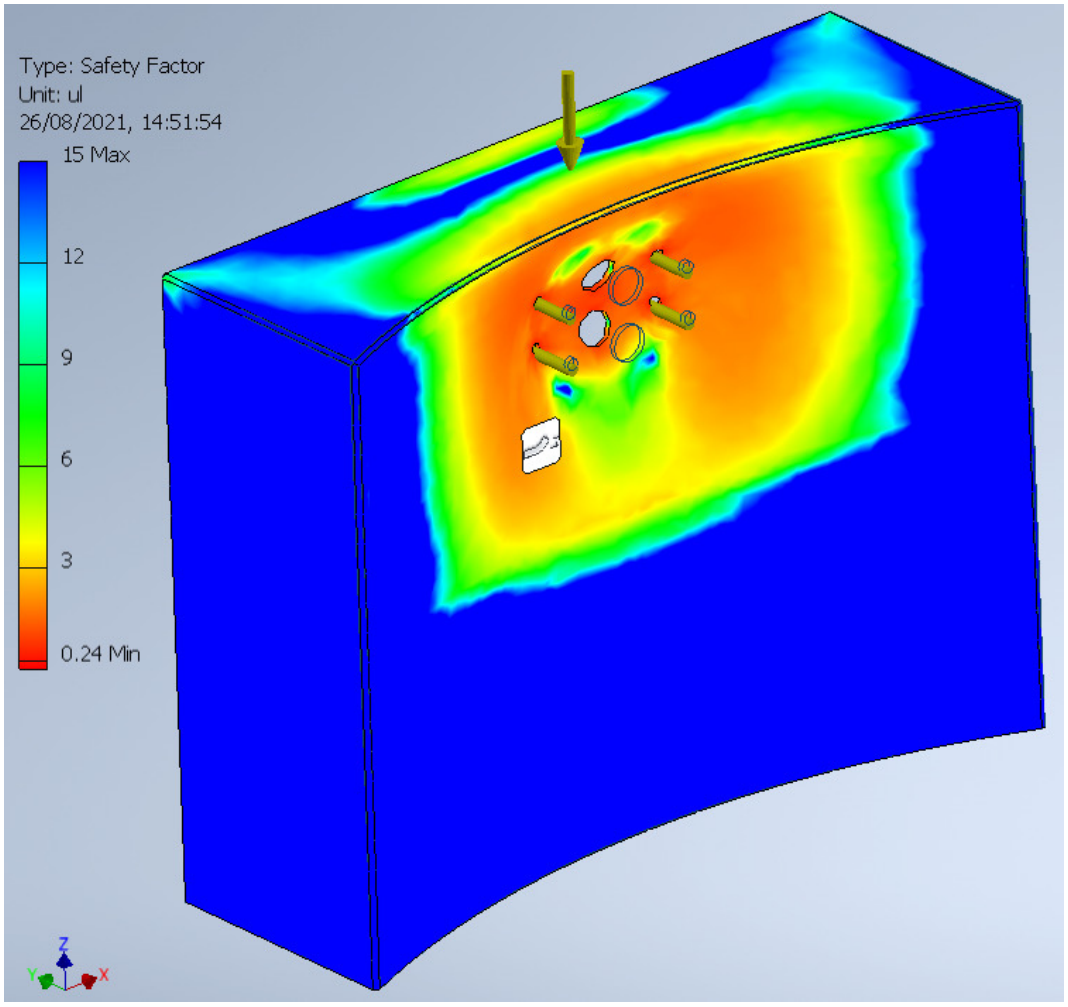


Figura 3.24 Factor de seguridad en la base frontal de la estación de recarga.

3.3.2 Razón de carga (baterias)

Se realizaron una serie de mediciones de voltaje para comparar con la medición digital del microcontrolador y obtener un valor de razón. En el Anexo 10 se puede observar todas las mediciones tomadas con lo que se obtuvo una razón de 0.0041. Usando este valor se obtuvo un error de $\pm 0.04V$ en la medición de voltaje del microcontrolador.

El error obtenido entre el valor de voltaje medido digitalmente por el microcontrolador y manualmente con un multímetro es bastante aceptable para una medición de voltaje de 12V.

3.4 Implementación de CoviBot

Las pruebas experimentales fueron realizadas en el pasillo exterior de la entrada del

Edificio 3A en ESPOL. En la Figura 3.25 se observa el espacio en el que se realizaron las pruebas con CoviBot.

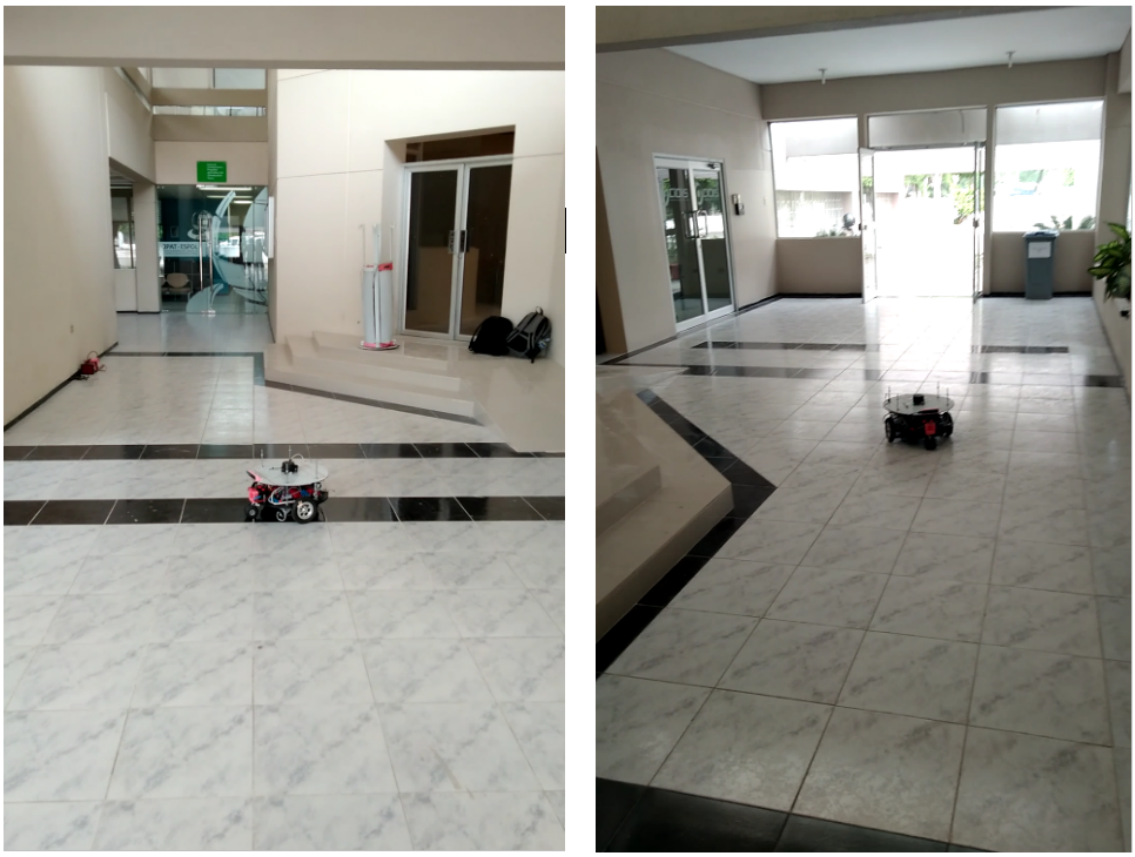


Figura 3.25 Escenario de pruebas en pasillo de entrada de Edificio 3A de ESPOL.

3.4.1 Mapeo

Inicialmente se generó un mapa de la zona de pruebas. En la Figura 3.26 se puede observar el mapa obtenido como resultado del mapeo.

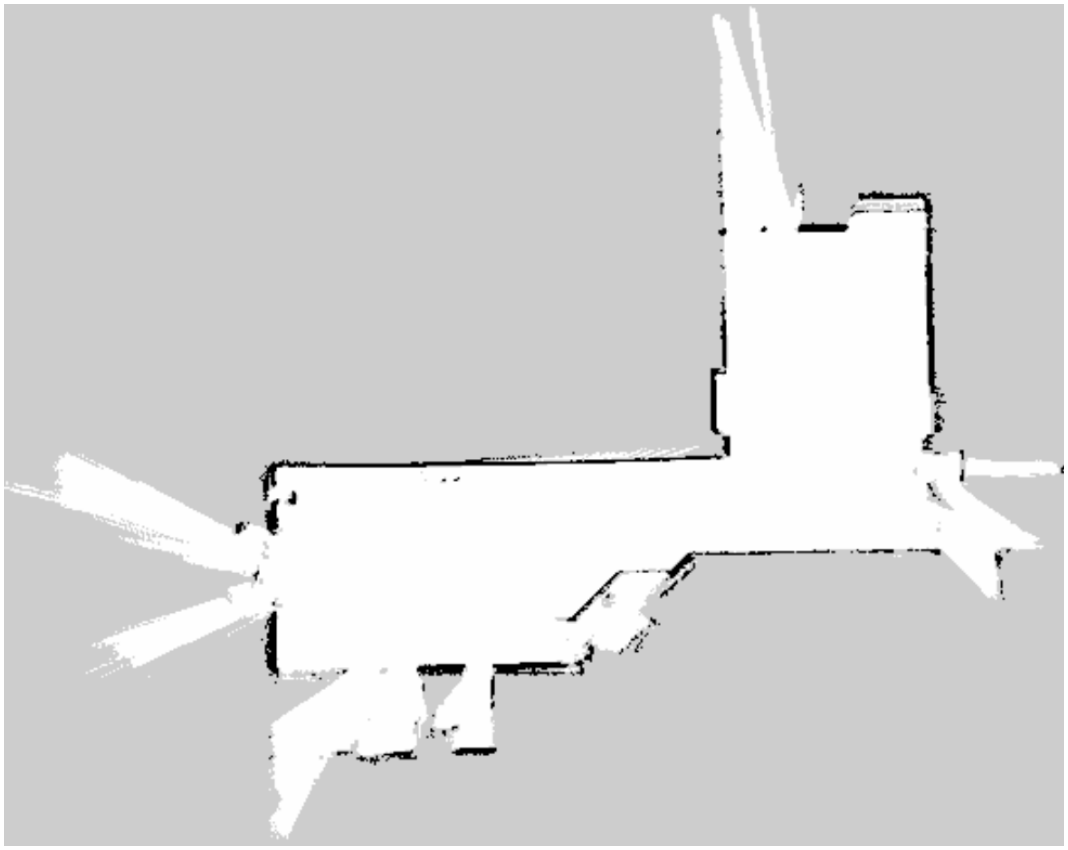


Figura 3.26 Mapa de pasillo exterior de entrada a edificio 3A de ESPOL.

A diferencia del mapa obtenido en simulación, en la Figura 3.26, se puede ver como el experimental tiene pequeñas fallas en su contorno y que las esquinas no se encuentran completamente definidas. Sin embargo, el grosor de las paredes es muy nítido lo cual aporta mucho.

En general se obtuvo un mapa satisfactorio. Las pequeñas fallas que se presentaron se deben al uso de un lidar de bajo costo y además del ruido adicional que se dispone en un ambiente real.

3.4.2 Trayectoria del movimiento de el robot

Para las pruebas experimentales de navegación se puso en frente del robot a una persona y se indujo a que el robot se mueva con el fin de evadir a la persona como se observa en la Figura 3.27.

Figura 3.27 Método de pruebas experimental de navegación usado.

En la Figura 3.28 se observa la trayectoria (línea de color azul) tomada por el robot usando el planeador. Se identifica que la trayectoria es suave, bien definida y con movimientos bien claros. Se mueve directamente desde su punto de inicio A hasta su punto final B y que además evade al obstáculo.

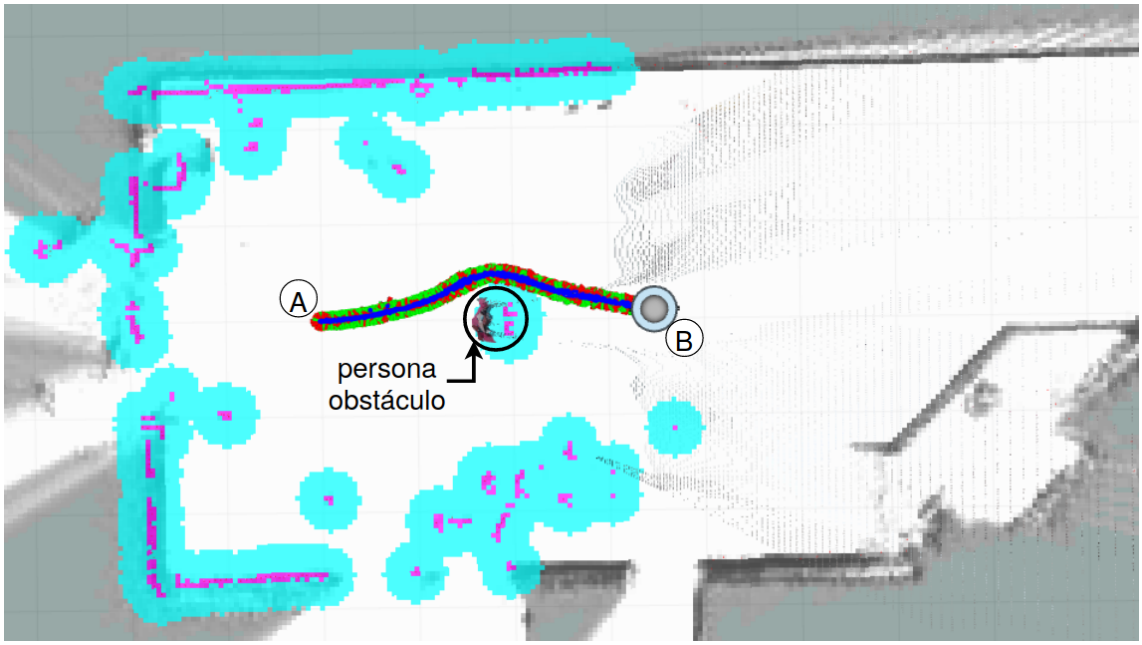


Figura 3.28 Trayectoria del robot en movimiento usando DWA Planner.

Se realizó la misma prueba usando el modelo de fuerzas sociales con el que se obtuvo la Figura 3.29.

A diferencia del planner, se puede ver que en este caso la trayectoria es poco eficiente, se ve un poco errática y los movimientos no son suaves. Esto se debe principalmente a que se implementó el SFM en un robot diferencial.

Al ser un robot de este tipo, es complicado generar un movimiento que vaya acorde a la salida del SFM dado que un robot diferencial no puede moverse en el eje Y.

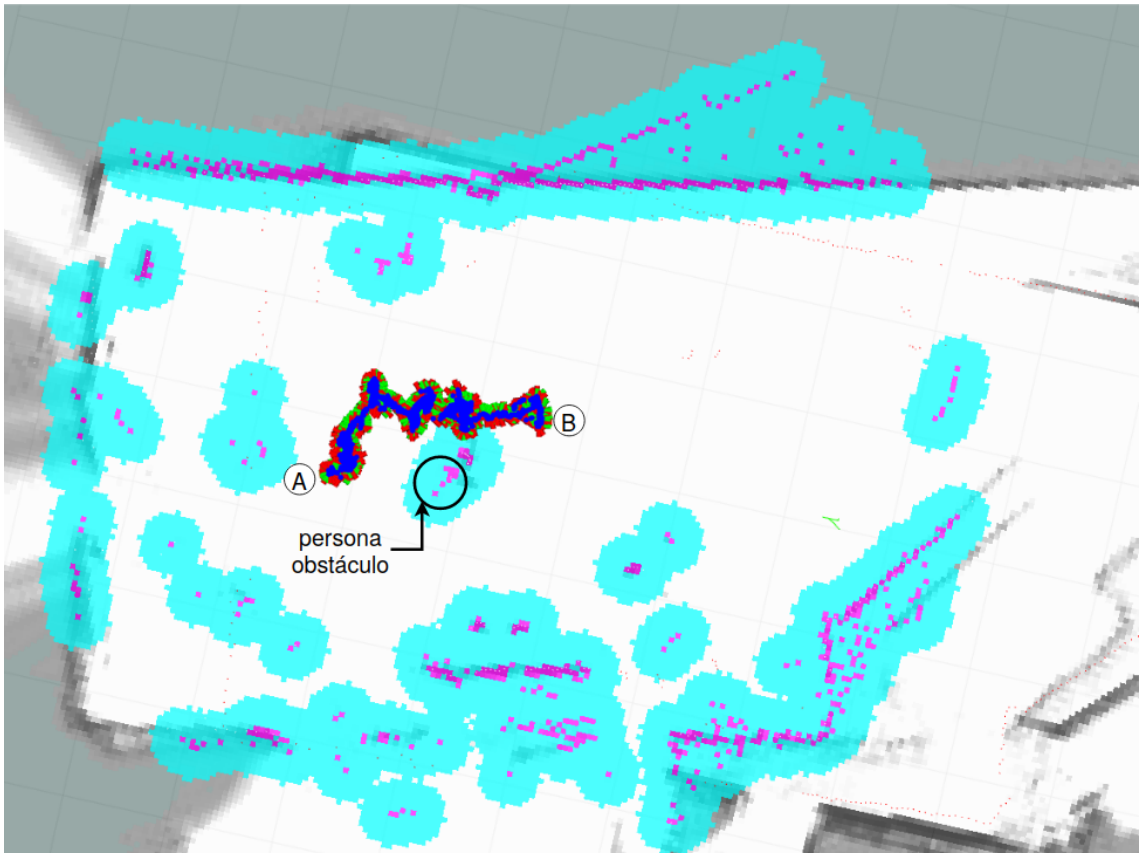


Figura 3.29 Trayectoria del robot en movimiento usando el modelo de fuerzas sociales.

3.4.3 Medición del Índice Social Individual

En la Figura 3.30 se observa la medida del SII para el caso del planeador. Se muestra como en un momento de la trayectoria el valor se eleva más de 0.5. Expresa que en un momento el robot se acercó demasiado a la persona y afectó a su confort. Se debe principalmente a que los planner se mueven en base a la distancia del robot a obstáculos sin considerar factores sociales.

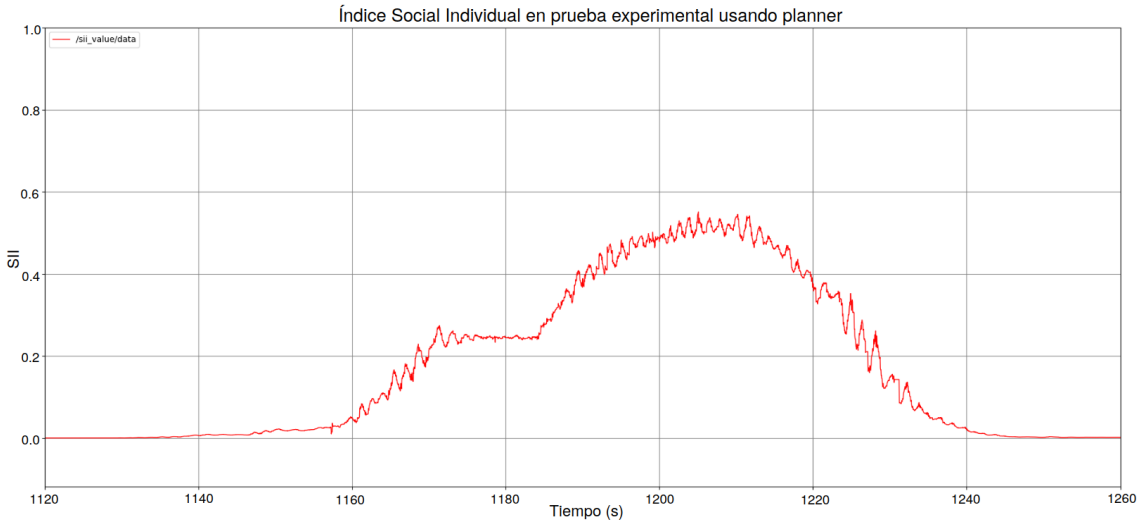


Figura 3.30 Índice Social Individual por el movimiento del planner.

En la Figura 3.31 se expresa la medida del SII cuando el robot se mueve usando el SFM. Podemos identificar que el valor cambia bruscamente y no tiene curvas suaves como en el caso del planeador. Este efecto se hace presente debido a su trayectoria errática por ser un robot diferencial. Sin embargo, a través de todo el movimiento del robot, este no pasa del valor límite para mantener el confort de la persona. En toda su trayectoria el SII es menor a 0.5.

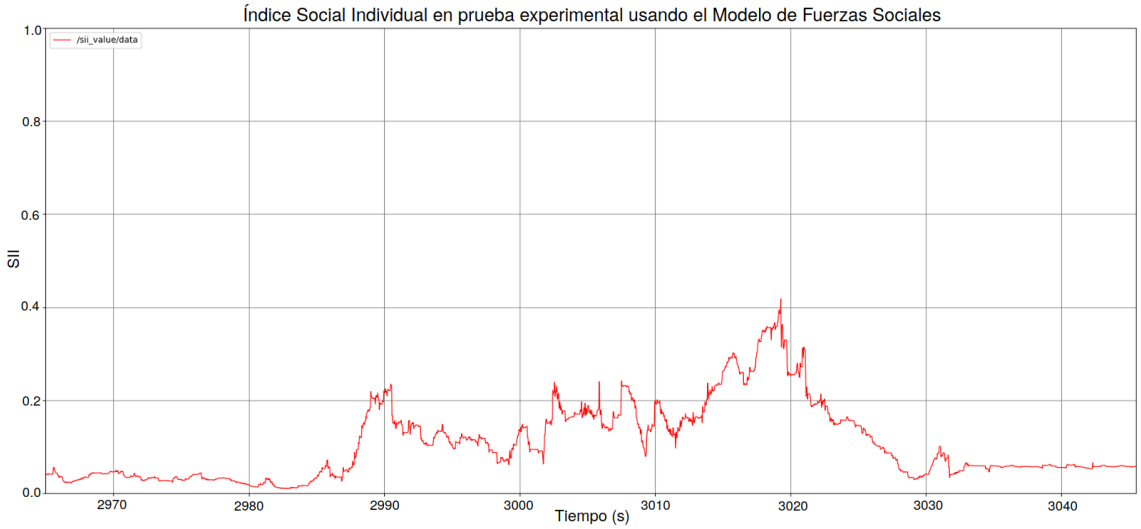


Figura 3.31 Índice Social Individual por el movimiento del Modelo De Fuerzas Sociales.

3.4.4 Ejecución de auto-recarga y auto-docking

Se realizaron pruebas para verificar el accionamiento del auto-docking y auto-recarga. En la Figura 3.32 se puede observar una de las pruebas de como es

el ingreso de el robot a su estación.

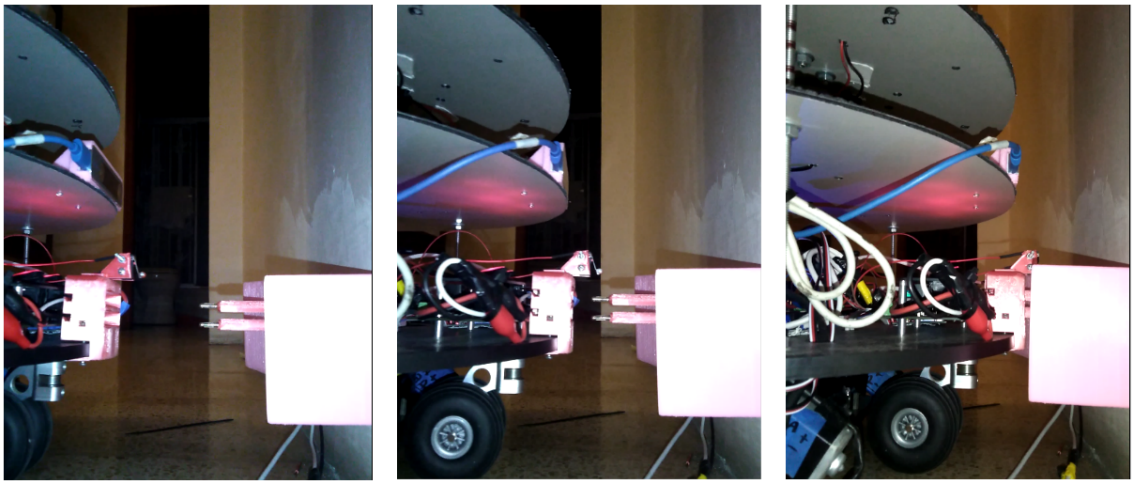


Figura 3.32 Secuencia de entrada a estación de auto-recarga.

También se hicieron también pruebas del proceso en la entrada del Edificio 3A, en la Figura 3.33 se puede observar en la imagen izquierda como inicia el robot, y en la derecha como se encuentra este justo antes de entrar a la estación de recarga.

Figura 3.33 Prueba de auto recarga en el Edificio 3A.

En la Figura 3.34 se presenta el estado final del robot al ingresar a la estación de recarga. No en todas las ocasiones se logró el ingreso del robot a la estación. Existe un margen de error debido a que en el retroceso del robot, no se tiene un control específico, simplemente se mueve el robot hacia atrás hasta que el switch trasero

se activa.

Figura 3.34 CoviBot completamente ingresado en la estación de recarga.

3.5 Acción de desinfección

Se llevó a cabo la desinfección en un espacio dentro de unas oficinas del CIDIS. En esta ocasión se tomaron las respectivas medidas de seguridad. Se encendieron las lámparas UV-C tomando precauciones de distancia. En la Figura 3.35 se observa al robot encendido en su acción de desinfección activa.

Figura 3.35 Acción de desinfección de oficina dentro del CIDIS.

En el proceso de desinfección se presentó una falla encendiendo las cuatro lámparas de CoviBot simultáneamente. Como se observa en la Tabla 2.16 se puede ver que con todas las lámparas el consumo es de 13.64A. Este alto consumo de corriente, provoca que el voltaje de las baterías recaiga y se active un paro de emergencia del inversor.

Se logró integrar exitosamente tan solo dos lámparas UV-C en el robot. Para poder encender tres o cuatro lámparas, se necesita de la integración de otra batería para aumentar la capacidad de carga que se pueda entregar.

3.6 Análisis de costos

Como consideración en el costo del proyecto se consideraron los distintos componentes de hardware usado y las horas de trabajo implementadas.

En la Tabla 3.1 se muestra el costo del hardware del robot de desinfección.

Tabla 3.1 Costos de componentes físicos de robot de desinfección.

Componentes Robot de Desinfección				
Equipo - Hardware	**Unidades**	**Costo Unitario**	**Costo Total**	**Proveedor**
Plataforma Arlo Parallax	1	$700.00	$700.00	Parallax
Jetson Nano 2GB	1	$60.00	$60.00	Nvidia
RPLidar A1	1	$100.00	$100.00	SlamTech
Cámara D435	1	$189.00	$189.00	Intel
Cámara T265	1	$209.00	$209.00	Intel
Step Down 5V 5A	2	$30.00	$60.00	N/A
ESP32	1	$12.00	$12.00	Espressif
Inversor 12VDC – 110VAC	1	$25.00	$25.00	Suredom
Lámparas UV-C	4	$11.00	$44.00	N/A
Marcos lámparas UV-C	4	$3.00	$12.00	N/A
Plancha de aluminio compuesto	1	$24.00	$24.00	MegaMetales
Pie de amigo	4	$0.75	$3.00	Ferretería Reinoso
Misceláneos de conexión eléctrica y elementos mecánicos	1	$5.00	$5.00	N/A
Guías de estructura - impresión 3D	8	$45.00	$360.00	N/A
Varilla de roscada Galvanizada 1/4 *1 m	1	$0.60	$0.60	Ferretería Reinoso
Puerto de auto recarga - Impresión 3D	1	$25.00	$25.00	N/A
Soporte Cámara T265 – Impresión 3D	1	$15.00	$15.00	N/A
Total			$1,843.60	

Se consideró entre los costos los componentes de la estación de auto-recarga como se especifica en la Tabla 3.2.

Tabla 3.2 Costo de actividades como mano de obra.

Componentes de estación de auto-recarga				
Equipo\Hardware	**Unidades**	**Costo Unitario**	**Costo Total**	**Fabricante/Proveedor**
ESP32	1	$12.00	$12.00	Espressif
Relé 5V	1	$1.68	$1.68	N/A
Impresiones 3D	1	$70.00	$70.00	Impresora Propia
Misceláneos de conexión eléctrica y elementos mecánicos	1	$2.50	$2.50	N/A
Cargador de pulso de baterías	1	$20.00	$20.00	AYBE
Total			$106.18	

Cómo últimos costo, en la Tabla 3.3 se pueden observar los costos por mano de obra basado en las horas implementadas para la solución.

Tabla 3.3 Costo de actividades como mano de obra.

Mano de obra			
Área/Actividad	**Horas**	**Costo por hora**	**Costo**
Programación	45	$4.00	$180.00
Ensamble	24	$3.00	$72.00
Diseño controlador	20	$3.50	$70.00
Simulación en Gazebo	22	$4.50	$99.00
Implementación de arquitectura	15	$3.50	$52.50
Diseño de base de auto-recarga	13	$3.00	$39.00
Total	139		$512.50

El costo total del robot es de $2,452.00. Dado que está destinado al uso como robot de servicio, para sus capacidades y su necesidad, el precio es bastante accesible. Los gastos mayores corresponden a la plataforma base de Arlo Parallax y los sensores como el lidar y cámaras que son indispensables.

A pesar de que también hay un gran gasto en lo que son impresiones 3D, estas son justificadas debido a que se construyó solo una unidad. Las piezas necesitadas eran muy específicas, elementos que no se pueden encontrar fácilmente en el mercado.

Se piensa que el robot será rentado por $350 mensuales. Considerando que una desinfección profesional puede llegar a costar hasta $140 por hora, CoviBot es capaz de ahorrar hasta un 70 % en desinfección e incluso en costos operacionales de desinfección manual.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Se logró obtener un diseño funcional y la implementación prototipo de un robot móvil autónomo de desinfección de espacios cerrados a través del uso de lámparas UV-C. Se pudo llevar a cabo la simulación del robot con su estructura mecánica principal en la cual se implementaron y probaron la mayor cantidad de nodos en ROS. Posteriormente se construyó efectivamente y se pudieron realizar las correspondientes pruebas experimentales que corroboraron las capacidades del proyecto.

Según los resultados obtenidos, el robot demuestra cumplir con los requerimientos como su peso máximo, con un valor muy satisfactorio actual de 41 libras. Además, se logró implementar un controlador para los motores. En esta etapa el método de Ziegler Nichols demostró resultados poco aceptables, pero se pudo obtener una muy buena respuesta con el diseño de un controlador PI usando un método heurístico. De esta manera se le dio la capacidad al robot de moverse perfectamente. A pesar de que en el control de los motores se presentó un poco de retraso en la señal de salida, esta no ha afectado al resultado dado que las velocidades del robot son generalmente bajas.

También se diseñó exitosamente la estación de auto-recarga. Esta pudo ser implementada y probada junto al robot. Se obtuvieron análisis de esfuerzos que revelaron que se necesitan modificaciones en la mecánica y estructura de la estación de recarga. En la experimentación esta funcionó como se esperaba debido a que se usó una velocidad mucho menor para el auto-docking. Esta característica permite que el robot pueda ser recargado a través del ingreso a la estación de recarga y la conexión de sus puertos traseros.

Se realizaron pruebas en las cuales se comprobaron la arquitectura de navegación del robot y se demostró su capacidad de autonomía en la navegación. CoviBot evidenció tener la capacidad de evadir obstáculos con ambos métodos de navegación. Para este

caso, la navegación del planner manifestó ser mucho más eficiente en movimiento, mientras que el Modelo de Fuerzas Sociales demostró mantener el confort de las personas pero con movimientos poco eficientes. Cabe recalcar que también se encontró que el Modelo de Fuerzas Sociales no debe ser usado directamente en un robot diferencial ya que este no es capaz de seguir a un vector de velocidad muy oscilante proveniente del centro del robot.

Finalmente, el robot pudo ser armado y probado con éxito. Se cumplieron los objetivos. El robot fue diseñado en base a los requerimientos y se pudo implementar toda la arquitectura electrónica y de control de software para permitirle ser autónomo. Esto se comprobó tanto en simulación como en la experimentación donde también se realizó procesos de SLAM. La arquitectura de software fue verificada a través de las distintas pruebas en las cuales se identificó un correcto funcionamiento entre los nodos implementados para las acciones. Por otro lado, a pesar de que la arquitectura de electrónica presentó falla en el encendido de todas las lámparas UV-C debido a la falta de carga, esta no mostró anomalías externas en actuadores y sensores.

4.2 Recomendaciones

Como trabajos futuros, se recomienda principalmente que se le agregue al robot una batería adicional para que pueda funcionar con las cuatro lámparas UV-C y soporte la caída de voltaje de cuando estas son encendidas. Esta batería puede ser agregada en la zona intermedia entre las lámparas donde se dispone de una gran cantidad de espacio. Con una batería adicional igual a las usadas en la solución sería suficiente para su funcionamiento.

También se debe de considerar el cambio de la base del robot a una de tipo holonómica. Un robot holonómico es capaz de funcionar mucho mejor acorde al SFM y causaría movimientos un poco más eficientes mientras respete el confort de las personas. Esto se debe a que este tipo de robots tiene una cinemática con mayores grados de libertad y puede moverse en el eje Y. Aunque si se implementa un sistema holonómico se debe corroborar el consumo de corriente ya que se tendría que adicionar motores en el sistema.

Para generar un mejor movimiento de auto-docking, se recomienda poner la cámara

D435i en la parte posterior del robot. De esta manera sería posible controlar mucho más la entrada del robot a la estación de recarga. Es posible corregir la posición del robot evaluando constantemente la transformada de posición de la marca de Aruco. Asimismo se recomienda incluir más cámaras de profundidad alrededor del robot. Con tres cámaras adicionales, el robot tendría un rango mucho más amplio de detección de personas y sería mucho más seguro para generar los paros de emergencia.

Por último, de ser posible, todas las piezas deben de ser cambiadas por un material más resistente como polietileno. De lo contrario, es necesario aumentar soportes o refuerzos en las piezas de la base frontal del robot y los portaelectrodos. Aluminio maquinado también sería una gran opción dado que es ligero.

BIBLIOGRAFÍA

[1] N. Akbulaev, I. Mammadov, and V. Aliyev, *Economic Impact of COVID-19*. 1 ed., Londres, MA, 2020. [E-book] Disponible en: CEPR Press.

[2] S. Tian, N. Hu, J. Lou, K. Chen, X. Kang, Z. Xiang, H. Chen, D. Wang, N. Liu, D. Liu, G. Chen, Y. Zhang, D. Li, J. Li, H. Lian, S. Niu, L. Zhang, and J. Zhang, "Characteristics of COVID-19 infection in Beijing," *Journal of Infection*, vol. 80, no. 4, pp. 401–406, 2020.

[3] Gobierno de México, "Limpieza y desinfección de espacios comunitarios durante la pandemia por SARS-CoV-2," *Gobierno de México*, pp. 1–13, 2020.

[4] B.-C. Natali, Z.-T. Miriam, F.-c. Fabricio, and C.-l. Katherine, "Ultraviolet light for disinfection in health areas , in front of covid-19 . Literature review .," *Revista OACTIVA UC Cuenca*, vol. 5, no. 3, pp. 107–114, 2020.

[5] T. Shimabukuro, "Allergic reactions including anaphylaxis after receipt of the first dose of Pfizer-BioNTech COVID-19 vaccine — United States, December 14–23, 2020," *American Journal of Transplantation*, vol. 21, no. 3, pp. 1332–1337, 2021.

[6] C. de Comercio de Bogotá, "PROTOCOLO DE BIOSEGURIDAD FRENTE A LA PREVENCIÓN AL CONTAGIO POR CORONAVIRUS COVID-19," *Cámara de Comercio de Bogotá*, 2021. [En línea]. Disponible en: http://ccb.org.co [Accedido: 20-my-2021.

[7] T. Greenhalgh, J. L. Jimenez, K. A. Prather, Z. Tufekci, D. Fisman, and R. Schooley, "Ten scientific reasons in support of airborne transmission of SARS-CoV-2," *The Lancet*, vol. 397, no. 10285, pp. 1603–1605, 2021.

[8] P. Megantoro, H. Setiadi, and B. A. Pramudita, "All-terrain mobile robot disinfectant sprayer to decrease the spread of COVID-19 in open area," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 3, pp. 2090–2100, 2021.

[9] C. Otálora, “Desarrollo de un Sistema de Monitoreo y Planificación de Procesos de Desinfección para Ambientes COVID-19,” tesis ingenieril, Universidad del Rosario, Bogotá, 2020.

[10] M. Cardona, F. Cortez, A. Palacios, and K. Cerros, “Mobile robots application against covid-19 pandemic,” *2020 Ieee Andescon, Andescon 2020*, 2020.

[11] E. Freund and P. Kaever, “Autonomous mobile robots,” *IFAC Proceedings Series*, no. 10, pp. 255–260, 1989.

[12] E. Romaine, “Types and applications of autonomous mobile robots (amrs).” Disponible en: https://www.conveyco.com/types-and-applications-of-amrs/, Agosto 2020.

[13] A. Bradshaw, “Sensors for mobile robots,” *Measurement and Control*, vol. 23, no. 2, pp. 48–52, 1990.

[14] M. Köseoğlu, O. M. Çelik, and Ö. Pektaş, “Design of an autonomous mobile robot based on ROS,” *IDAP 2017 - International Artificial Intelligence and Data Processing Symposium*, 2017.

[15] J. Zhang and S. Singh, “Low-drift and real-time lidar odometry and mapping,” *Autonomous Robots*, vol. 41, no. 2, pp. 401–416, 2017.

[16] B. Langmann, K. Hartmann, and O. Loffeld, “Depth camera technology comparison and performance evaluation,” *ICPRAM 2012 - Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, vol. 2, pp. 438–444, 2012.

[17] S. K. Malu and J. Majumdar, “Kinematics, Localization and Control of Differential Drive Mobile Robot Global Journal of Researches in Engineering: H Kinematics, Localization and Control of Differential Drive Mobile Robot,” *Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Inc*, vol. 14, no. 1, 2014.

[18] C. A. Ganzaroli, D. F. De Carvalho, R. N. Dias, M. R. Reis, A. J. Alves, J. L. Domingos, and W. P. Calixto, “Heuristic and deterministic strategies applied on

cascade PI controller tuning for speed control of a DC motor," *CHILECON 2015 - 2015 IEEE Chilean Conference on Electrical, Electronics Engineering, Information and Communication Technologies, Proceedings of IEEE Chilecon 2015*, pp. 101–106, 2016.

[19] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The SLAM problem: A survey," *Frontiers in Artificial Intelligence and Applications*, vol. 184, no. 1, pp. 363–371, 2008.

[20] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, 2016.

[21] N. Yu and B. Zhang, "An Improved Hector SLAM Algorithm based on Information Fusion for Mobile Robot," *Proceedings of 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems, CCIS 2018*, pp. 279–284, 2019.

[22] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1271–1278, 2016.

[23] R. Yagfarov, M. Ivanou, and I. Afanasyev, "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth," *2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018*, no. November, pp. 1979–1983, 2018.

[24] S. Wang, Y. Li, Y. Sun, X. Li, N. Sun, X. Zhang, and N. Yu, "A localization and navigation method with ORB-SLAM for indoor service mobile robots," *2016 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2016*, pp. 443–447, 2016.

[25] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[26] M. Keller, F. Hoffmann, T. Bertram, C. Hass, and A. Seewald, *Planning of optimal collision avoidance trajectories with timed elastic bands*, vol. 19. IFAC, 2014.

[27] J. Cheng, H. Cheng, M. Q. Meng, and H. Zhang, “Autonomous Navigation by Mobile Robots in Human Environments: A Survey,” in *2018 IEEE International Conference on Robotics and Biomimetics, ROBIO 2018*, 2018.

[28] R. Campa and R. Campa, “The rise of social robots : a review of the recent literature,” *Journal of Evolution and Technology*, 2016.

[29] J. Rios-Martinez, A. Spalanzani, and C. Laugier, “From Proxemics Theory to Socially-Aware Navigation: A Survey,” *International Journal of Social Robotics*, 2015.

[30] K. Charalampous, I. Kostavelis, and A. Gasteratos, “Recent trends in social aware robot navigation: A survey,” 2017.

[31] T. Hall, “The hidden dimension: man’s use of space in public and private,” *The Bodley Head Ltd*, 1966.

[32] H. Kivrak, F. Cakmak, H. Kose, and S. Yavuz, “Social navigation framework for assistive robots in human inhabited unknown environments,” *Engineering Science and Technology, an International Journal*, no. xxxx, 2020.

[33] H. Kivrak, F. Cakmak, H. Kose, and S. Yavuz, “A multilevel mapping based pedestrian model for social robot navigation tasks in unknown human environments,” in *International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, 2020.

[34] H. Kivrak, F. Cakmak, H. Kose, and S. Yavuz, “Socially Aware Robot Navigation Using the Collision Prediction Based Pedestrian Model,” no. May 2019, 2018.

[35] G. Song, H. Wang, J. Zhang, and T. Meng, “Automatic docking system for recharging home surveillance robots,” *IEEE Transactions on Consumer Electronics*, vol. 57, no. 2, pp. 428–435, 2011.

[36] K. L. Su, Y. L. Liao, S. P. Lin, and S. F. Lin, “An interactive auto-recharging system for mobile robots,” *International Journal of Automation and Smart Technology*, vol. 4, no. 1, pp. 43–53, 2014.

[37] M. Rao and M. Shivakumar, “IR Based Auto-Recharging System for Autonomous Mobile Robot,” *Journal of Robotics and Control (JRC)*, vol. 2, no. 4, 2021.

[38] S. Daniel, J. Younghun, S. Jorge, and L. Marcus, “Automatic docking.” [En línea] Disponible en: https://wiki.ros.org/kobuki/Tutorials/Automatic%20Docking. [Accedido: 20-Jul-2021], 2021.

[39] L. Frenzel, *CONTEMPORARY ELECTRONICS: FUNDAMENTALS, DEVICES, CIRCUITS, AND SYSTEMS*. New York, NY, USA: McGraw-Hill, 2014.

[40] Parallax, “Arlo complete robot system.” [En línea] Disponible en https://www.parallax.com/product/arlo-complete-robot-system/ [Accedido: 20-Jul-2021].

[41] Parallax, “Motor mount and wheel kit - aluminum.” [En línea] Disponible en https://www.parallax.com/product/motor-mount-wheel-kit-aluminum/ [Accedido: 20-Jul-2021].

[42] Parallax, “36-position quadrature encoder set.” [En línea] Disponible en `https://www.parallax.com/product/36-position-quadrature-encoder-set/` [Accedido: 20-Jul-2021].

[43] Parallax, “Dhb-10 dual h-bridge 10 amp motor controller (28231).” [En línea] Disponible en: https://www.parallax.com/package/dhb-10-motor-controller-product-guide/?ind=1602803488988filename=28231-DHB-10-Motor-Controller-Guide-v1.0.pdfwpdmdl=4750refresh=60dfa91cdfcba1625270556 [Accedido: 20-Jul-2021].

[44] L. Shanghai Slamtec Co., “RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner,” pp. 1–16, 2020.

[45] Intel, “Intel® realsensetm product family d400 series.” [En línea] Disponible en: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf [Accedido: 20-Jul-2021].

[46] Intel, “Intel® realsense™ tracking camera.” [En línea] Disponible en: https://www.intelrealsense.com/wp-content/uploads/2019/09/. [Accedido: 20-Jul-2021].

[47] “T8 germicidal lamp uvc.” [En línea] Disponible en: http://www.ch-lighting.com/productinfo/387308.html?templateId=1133605, [Accedido: 20-Jul-2021].

[48] NVIDIA, “Jetson nano 2gb developer kit.” [En línea] https://developer.nvidia.com/embedded/jetson-nano-2gb-developer-kit [Accedido: 20-jul-2021.

[49] ESPRESSIF, “Esp32wroom32d and esp32wroom32u.” [En línea] Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf [Accedido: 20-jul-2021], 2021 [Online].

[50] H. Yoshida, H. Fujimoto, D. Kawano, Y. Goto, M. Tsuchimoto, and K. Sato, “Range extension autonomous driving for electric vehicles based on optimal velocity trajectory and driving braking force distribution considering road gradient information,” *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, no. Figure 1, pp. 4754–4759, 2015.

[51] A. Mahtani, L. Sánchez, E. Fernández, and A. Martinez, *Effective Robotics Programming with ROS*. Birminghan, UK: Packt Publishin Ltd., 2016.

[52] S. Silva Mendoza, D. F. Paillacho Chiluiza, D. Soque León, M. Guerra Pintado, and J. S. Paillacho Corredores, “Autonomous Intelligent Navigation for Mobile Robots in Closed Environments,” *Applied Technologies: Second International Conference, ICAT 2020*, vol. 1388, pp. 391–402, 2021.

[53] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, “2D SLAM quality evaluation methods,” *Conference of Open Innovation Association, FRUCT*, pp. 120–126, 2018.

[54] M. Filipenko and I. Afanasyev, “Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment,” *9th International Conference on Intelligent Systems 2018: Theory, Research and Innovation in Applications, IS 2018 - Proceedings*, pp. 400–407, 2018.

[55] C. Henkel, A. Bubeck, and W. Xu, “Energy Efficient Dynamic Window Approach for Local Path Planning in Mobile Service Robotics,” *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 32–37, 2016.

[56] M. Moussaïd, D. Helbing, S. Garnier, A. Johansson, M. Combe, and G. Theraulaz, “Experimental study of the behavioural mechanisms underlying self-organization in

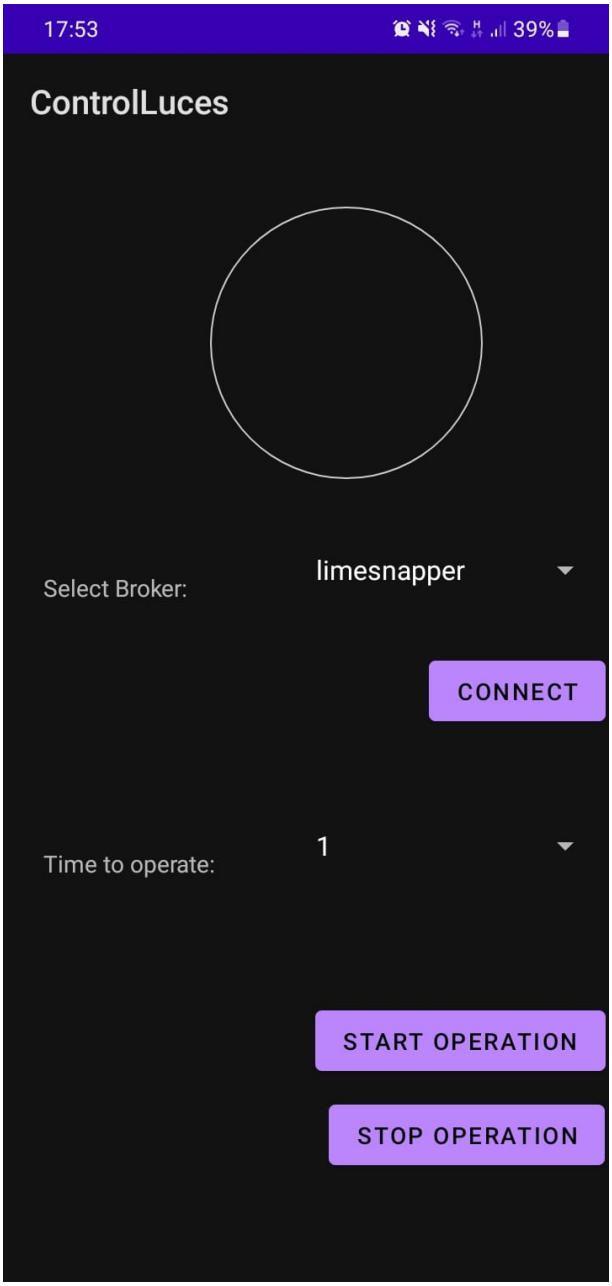
human crowds," *Proceedings of the Royal Society B: Biological Sciences*, vol. 276, no. 1668, pp. 2755–2762, 2009.

[57] T. Fraichard and V. Levesy, "From Crowd Simulation to Robot Navigation in Crowds," *IEEE Robotics and Automation Letters*, 2020.

[58] P. Żur, "Finite Elements Analysis of PLA 3D-printed Elements and Shape Optimization," *European Journal of Engineering Science and Technology*, no. January, 2019.

ANEXOS

ANEXO 1.- CAPTURA DE APLICACIÓN PARA CONTROL DE LÁMPARAS UV-C



ANEXO 2.- ARCHIVO DE CONFIGURACIÓN .LUA PARA CARTOGRAPHER ROS

```
-- Copyright 2016 The Cartographer Authors
--
-- Licensed under the Apache License, Version 2.0 (the "License");
-- you may not use this file except in compliance with the License.
-- You may obtain a copy of the License at
--
--      http://www.apache.org/licenses/LICENSE-2.0
--
-- Unless required by applicable law or agreed to in writing, software
-- distributed under the License is distributed on an "AS IS" BASIS,
-- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
-- See the License for the specific language governing permissions and
-- limitations under the License.

include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_footprint",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_odometry = false,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
```

```
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

MAP_BUILDER.use_trajectory_builder_2d = true

TRAJECTORY_BUILDER_2D.min_range = 0.15
TRAJECTORY_BUILDER_2D.max_range = 12.0
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 3.
TRAJECTORY_BUILDER_2D.use_imu_data = false
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.1)

POSE_GRAPH.constraint_builder.min_score = 0.65
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.7

return options
```

ANEXO 3.- NODO PARA CONTROL DE VELOCIDAD DE LOS MOTORES USANDO PID

```python
#!/usr/bin/env python3
from simple_pid import PID
from arlo_control_msgs.msg import WheelsPower
import rospy
from std_msgs.msg import Float32, Int16
import time


You, 2 months ago | 1 author (You)
class PidVelocity:
    def __init__(self):
        rospy.init_node("pid_velocity")
        self.Kp = rospy.get_param("~Kp", 10)
        self.Ki = rospy.get_param("~Ki", 10)
        self.Kd = rospy.get_param("~Kd", 0.0001)
        self.ouput_max = rospy.get_param("~out_max", 127)
        self.ouput_min = rospy.get_param("~out_min", -127)
        self.ticks_per_meter = rospy.get_param("ticks_meter", 20)
        self.rate = rospy.get_param("~rate", 30)
        self.timeout_ticks = rospy.get_param("~timeout_ticks", 4)

        self.cur_vel = 0
        self.target_vel = 0
        self.rate = rospy.Rate(self.rate)

        self.power_msg = Float32()
        self.wheel_vel_msg = Float32()

        self.time_tick = time.time()

        # subscribers
        rospy.Subscriber("wheel", Int16, self.wheelCallback)
        rospy.Subscriber("wheel_vtarget", Float32, self.targetCallback)
```

```
33
34          # publishers
35          self.wheel_vel_pub = rospy.Publisher("wheel_vel", Float32, queue_size=10)
36          self.wheel_power_pub = rospy.Publisher("motor_cmd", Float32, queue_size=10)
37
38          self.wheels_power_pub = rospy.Publisher(
39              "wheels_power", WheelsPower, queue_size=10
40          )
41
42          ## pid definition
43          self.pid = PID(self.Kp, self.Ki, self.Kd, setpoint=0)
44          self.pid.output_limits = (self.ouput_min, self.ouput_max)
45          self.pid.sample_time = 1 / 15
46          rospy.on_shutdown(self.cleanup)
47          # self.pid.proportional_on_measurement = True
48
49      def cleanup(self):
50          wheel_power_stop = WheelsPower()
51          wheel_power_stop.left_power = 0
52          wheel_power_stop.right_power = 0
53
54          self.wheels_power_pub.publish(wheel_power_stop)
55
56      def wheelCallback(self, msg):
57          self.cur_vel = float(msg.data) / float(self.ticks_per_meter)
58
59      def targetCallback(self, msg):
60          self.target_vel = msg.data
61          self.pid.setpoint = self.target_vel
62          self.time_tick = time.time()
63
```

```
64      def start(self):
65          while not rospy.is_shutdown():
66              if time.time() - self.time_tick > self.timeout_ticks:
67                  self.power_msg.data = 0
68                  self.wheel_power_pub.publish(self.power_msg)
69              else:
70                  cur_output = self.pid(self.cur_vel)
71                  self.power_msg.data = cur_output
72                  self.wheel_vel_msg.data = self.cur_vel
73                  self.wheel_vel_pub.publish(self.wheel_vel_msg)
74                  self.wheel_power_pub.publish(self.power_msg)
75                  print("pid computed", cur_output)
76
77              self.rate.sleep()
78
79
80  if __name__ == "__main__":
81      pid_controller = PidVelocity()
82      pid_controller.start()
```

ANEXO 4.- CONFIGURACIÓN DE ESCENARIO PARA SIMULACIÓN DE AGENTES

```xml
<?xml version="1.0" ?>
<scenario>
  <obstacle x1="1.075" x2="1.025" y1="6.375" y2="6.425"/>
  <obstacle x1="1.125" x2="1.075" y1="6.375" y2="6.425"/>
  <obstacle x1="1.175" x2="1.125" y1="6.375" y2="6.425"/>
  <obstacle x1="1.225" x2="1.175" y1="6.375" y2="6.425"/>
  <obstacle x1="1.275" x2="1.225" y1="6.375" y2="6.425"/>
  <obstacle x1="1.325" x2="1.275" y1="6.375" y2="6.425"/>
  <obstacle x1="1.375" x2="1.325" y1="6.375" y2="6.425"/>
  <obstacle x1="1.425" x2="1.375" y1="6.375" y2="6.425"/>
  <obstacle x1="1.475" x2="1.425" y1="6.375" y2="6.425"/>
  <obstacle x1="1.525" x2="1.475" y1="6.375" y2="6.425"/>
  <obstacle x1="1.575" x2="1.525" y1="6.375" y2="6.425"/>
  <obstacle x1="1.625" x2="1.575" y1="6.375" y2="6.425"/>
```

```
772     <obstacle x1="8.175" x2="8.125" y1="0.375" y2="0.425"/>
773     <obstacle x1="8.225" x2="8.175" y1="0.375" y2="0.425"/>
774
775
776     <!-- WAYPOINTS -->
777     <!-- comedor # 1 x=3 y=26 -->
778     <waypoint id="comedor" x="-4" y="0.8" r="0.5"/>
779     <!-- office # 2 x=11 y=28 -->
780     <waypoint id="sala" x="1" y="0.8" r="0.5"/>
781
782     <waypoint id="entretenimiento" x="3" y="-2.5" r="0.5"/>
783
784     <!--robot-->
785     <agent x="0" y="0" n="1" dx="0" dy="0" type="2">
786       <addwaypoint id="comedor"/>
787       <addwaypoint id="sala"/>
788     </agent>
789
790     <!-- definicion de agentes -->
791     <agent x="-8" y="0" n="1" dx="0" dy="0" type="1">
792       <addwaypoint id="comedor"/>
793       <addwaypoint id="sala"/>
794       <addwaypoint id="entretenimiento"/>
795     </agent>
796
797     <agent x="-7" y="0" n="1" dx="0" dy="0" type="1">
798       <addwaypoint id="comedor"/>
799       <addwaypoint id="sala"/>
800     </agent>
801
802 </scenario>
```

ANEXO 5.- CÁLCULOS DE CONSTANTES DE CONTROLADOR PID SEGÚN MÉTODO DE ZIEGLER NICHOLS

Según la Figura 2.28 se determina que:

$$P_\sigma = (5410 - 5408.65) = 1.35$$

Y se conoce antes que:

$$K_\sigma = 190$$

Según la Figura 1.4, entonces obtenemos que:

$$K_p = 190 * 0.6 = 114.00$$

$$K_i = 114/(1.35/2) = 168.88$$

$$K_d = 114 * ((1.35)/8) = 19.24$$

ANEXO 6.- CÁLCULOS PARA DIVISOR DE VOLTAJE

Se conoce como entrada un valor de 16V y se busca reducirlo a un aproximado de 3.3V para que pueda ser leído por un microcontrolador con entrada analógica de 3.3V máximo. Usando la Ecuación 1.8 se tomó como condición lo siguiente:

$$R_1 = 18kOhms$$

De esta manera se cálculo el valor de R_2:

$$3.3 = R_2/(R_1 + R_2) * 16$$

$$R_2 \simeq 4.7kOhms$$

Ambos valores de resistencias son comerciales.

ANEXO 7.- PROGRAMACIÓN EN MICROPYTHON DE MICROCONTROLADOR DE ESTACIÓN DE AUTO-RECARGA

```
import utime
import gc
from machine import Pin
import network
from umqtt.robust2 import MQTTClient

gc.enable()

def do_connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        wlan.connect("NETLIFE-Silva", "SASM3141")
        while not wlan.isconnected():
            pass
    # print("network config:", wlan.ifconfig())

do_connect()

utime.sleep(20)

shiftr_token = b"v4BmtoHRLVPHcfc4"
clientID = b"ESP32_charger"
usuario_ = b"paintbug935"
topic_time = b"tiempoActivacion"
topic_stop = b"stop"
topic_cargador = b"accionarCargador"

```

```
client = MQTTClient(
    clientID,
    b"paintbug935.cloud.shiftr.io",
    1883,
    user=usuario_,
    password=shiftr_token,
)  # creacion de objeto

client.DEBUG = True
# Information whether we store unsent messages with the flag QoS==0 in the queue.
client.KEEP_QOS0 = False
client.NO_QUEUE_DUPS = True
# Limit the number of unsent messages in the queue.
client.MSG_QUEUE_MAX = 1

relay_pin = Pin(22, Pin.OUT)

def charger_activate_callback(topic, msg, retained, duplicate):
    message = msg
    message = str(message.decode("utf-8"))
    if message == "1":
        relay_pin.on()
    else:
        relay_pin.off()

client.set_callback(charger_activate_callback)
client.connect(clean_session=False)  # conexion a broker
client.subscribe(topic_cargador)

```

```
while True:
    while True:
        try:
            if not client.is_conn_issue():
                client.check_msg()
                client.send_queue()
            else:
                client.reconnect()
                client.resubscribe()
        except:
            pass
```

ANEXO 8.- PROGRAMACIÓN EN MICROPYTHON DE MICROCONTROLADOR DE PLATAFORMA ROBÓTICA

```
import uros
from arlo_control_msgs import WheelsEncoders, WheelsPower  # rosserial messages
from arlorobot import *
import utime
import gc
from machine import Pin, ADC
import network
from umqtt.robust2 import MQTTClient
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("NETLIFE-Silva", "SASM3141")
        while not wlan.isconnected():
            pass
    # print("network config:", wlan.ifconfig())
      sasilva, a month ago • feat:rosgraphs made
do_connect()

utime.sleep(20)
You, a month ago | 2 authors (sasilva and others)
class ArlorobotMove(object):
    def __init__(self):

        # analog read
        self.voltage_read = ADC(Pin(34))
        self.voltage_read.atten(ADC.ATTN_11DB)

```

```
        # mqtt variables light control
        self.lUVC = Pin(15, Pin.OUT)  # 15 uv
        self.lRed = Pin(18, Pin.OUT)  #  18
        self.lGreen = Pin(25, Pin.OUT)
        self.lBlue = Pin(22, Pin.OUT)  # 22
        self.message = ""
        self.is_waiting = False

        self.time_to_work = None
        self.callback_time = 0

        self.connect_sub = 0
        self.mqtt_client_error = False
        self.mqtt_reconnecting = False

        self.initial_status_lights()
        # utime.sleep(1)

        # conexion a broker

        self.shiftr_token = b"v4BmtoHRLVPHcfc4"
        self.clientID = b"ESP32"
        self.usuario_ = b"paintbug935"
        self.topic_time = b"tiempoActivacion"
        self.topic_stop = b"stop"

        self.client = MQTTClient(
            self.clientID,
            b"paintbug935.cloud.shiftr.io",
            1883,
            user=self.usuario_,
            password=self.shiftr_token,
        )  # creacion de objeto

```

```
68          self.client.DEBUG = True
69          # Information whether we store unsent messages with the flag QoS==0 in the queue.
70          self.client.KEEP_QOS0 = False
71          self.client.NO_QUEUE_DUPS = True
72          # Limit the number of unsent messages in the queue.
73          self.client.MSG_QUEUE_MAX = 1
74
75          self.client.set_callback(self.uv_light_callback)
76          self.client.connect(clean_session=False)  # conexion a broker
77          self.client.subscribe(self.topic_time)
78          self.client.subscribe(self.topic_stop)
79
80          utime.sleep(1)
81
82          self.timer = machine.Timer(0)
83          self.timer.init(
84              period=60000,
85              mode=machine.Timer.PERIODIC,
86              callback=self.battery_voltage_callback,
87          )
88
89          # constants
90          # self.WHEEL_ENCODER_TOPIC = "/wheels_encoder"
91          self.WHEEL_POWER_TOPIC = "/wheels_power"
92          self.WHEEL_VELOCITY_TOPIC = "/wheels_vel"
93          self.BATTERY_TOPIC = "/battery_voltage"
94          self.PRESENCE_STOP_TOPIC = "/presence"
95          self.PIR_VALUE_TOPIC = "/pir_value"
96
```

```
97          self.arlobot = ArloRobot(serial_id=2, baudrate=19200, tx=17, rx=16, pace=2)
98          self.node = uros.NodeHandle(1, 115200, tx=1, rx=3)
99          # self.wheels_encoders = WheelsEncoders()
100         self.wheels_velocity = WheelsEncoders()
101         self.left_power = 0
102         self.right_power = 0
103
104         self.last_left_power = 0
105         self.last_right_power = 0
106
107         self.pir_value_msg = Bool()
108
109         self.battery_voltage_msg = Float32()
110
111         self.pir_pin = machine.Pin(35, machine.Pin.IN)
112
113         _thread.start_new_thread(self.mqtt_handler, ())
114
115     # funciones de control de luces
116     def initial_status_lights(self):
117         if not self.is_waiting:
118             self.lUVC.value(1)
119             self.lRed.value(1)
120             self.lGreen.value(0)
121             self.lBlue.value(1)
122
```

```
123     def last_status_lights(self):
124         self.is_waiting = True
125         cur_time = utime.ticks_ms()
126         while utime.ticks_ms() - cur_time < 4000:
127             self.lUVC.value(1)
128             self.lRed.value(0)
129             self.lGreen.value(0)
130             self.lBlue.value(1)
131         self.lUVC.value(1)
132         self.lRed.value(1)
133         self.lGreen.value(0)
134         self.lBlue.value(1)
135         self.is_waiting = False
136         return
137 
138     def light_controller_active(self):
139         self.lUVC.value(0)
140         self.lRed.value(0)
141         self.lGreen.value(1)
142         self.lBlue.value(1)
143 
144     # callbacks
145 
146     def uv_light_callback(self, topic, msg, retained, duplicate):
147         self.message = msg
148         self.message = str(self.message.decode("utf-8"))
149         self.callback_time = utime.ticks_ms()
150         if self.message == "stop":
151             self.is_waiting = True
152             self.time_to_work = None
153             _thread.start_new_thread(self.last_status_lights, ())
154         else:
155             self.time_to_work = int(self.message) * 1000
156 
```

```
157     def wheel_power_callback(self, msg):
158         self.left_power = msg.left_power
159         self.right_power = msg.right_power
160         self.left_power = int(self.left_power)
161         self.right_power = int(self.right_power)
162 
163         if (
164             self.last_left_power != self.left_power
165             or self.last_right_power != self.right_power
166         ):
167             self.arlobot.go(self.left_power, self.right_power)
168             self.last_left_power = self.left_power
169             self.last_right_power = self.right_power
170 
171     def presence_callback(self, msg):
172         msg = msg.data
173         if msg:
174             self.is_waiting = True
175             self.time_to_work = None
176             _thread.start_new_thread(self.last_status_lights, ())
177 
178     def battery_voltage_callback(self, timer):
179         voltaje_lectura = self.voltage_read.read()
180         self.connect_sub = utime.ticks_ms()
181         self.battery_voltage_msg.data = float(voltaje_lectura)
182         # print(self.battery_voltage_msg.data)
183         self.node.publish(self.BATTERY_TOPIC, self.battery_voltage_msg)
184 
```

```
185     def mqtt_handler(self):
186         while True:
187             try:
188                 if not self.client.is_conn_issue():
189                     self.client.check_msg()
190                     self.client.send_queue()
191                 else:
192                     self.client.reconnect()
193                     self.client.resubscribe()
194             except:
195                 pass
196 
```

```
197     def start(self):  # threads init and running
198         self.arlobot.clear_counts()
199         self.node.subscribe(
200             self.WHEEL_POWER_TOPIC, WheelsPower, self.wheel_power_callback
201         )
202         self.node.subscribe(self.PRESENCE_STOP_TOPIC, Bool, self.presence_callback)
203         while True:
204             self.pir_value_msg.data = self.pir_pin.value()
205             self.node.publish(self.PIR_VALUE_TOPIC, self.pir_value_msg)
206             if self.time_to_work is not None:
207                 try:
208                     if utime.ticks_ms() - self.callback_time > self.time_to_work:
209                         self.is_waiting = True
210                         self.time_to_work = None
211                         _thread.start_new_thread(self.last_status_lights, ())
212                     else:
213                         self.light_controller_active()
214                 except Exception as e:
215                     pass
216             else:
217                 self.initial_status_lights()
218             try:
219                 wheel_vel = self.arlobot.read_speeds()
220                 left = int(wheel_vel[0])
221                 right = int(wheel_vel[1])
222                 if isinstance(left, int) and isinstance(right, int):
223                     self.wheels_velocity.left_encoder = left
224                     self.wheels_velocity.right_encoder = right
225                     self.node.publish(
226                         self.WHEEL_VELOCITY_TOPIC, self.wheels_velocity
227                     )  # wheel speed value published
228             except Exception as e:
229                 pass
```

```
231 my_arlobot = ArlorobotMove()
232 my_arlobot.start()
```

ANEXO 9.- CÁLCULO DE CORRIENTE USADA POR LAS LÁMPARAS UV-C

Conociendo que cada una de las lámparas son de 40W y que las baterías son de 12V. Todas las lámparas se encuentran conectadas en paralelo. Podemos calcular la corriente consumida por las lamparas de la siguiente forma:

$$Corriente = \frac{Potencia}{Voltaje}$$

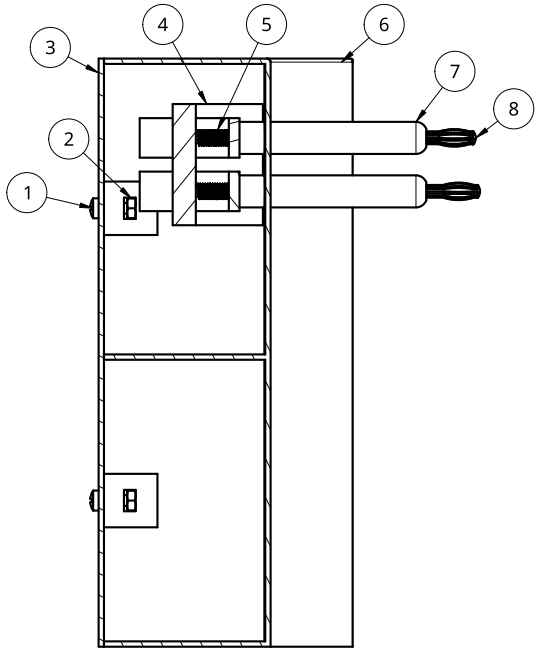
$$Corriente = \frac{40}{12} = 3.33A$$

Por lo tanto, la corriente consumida por cada lámpara es de 3.33A.

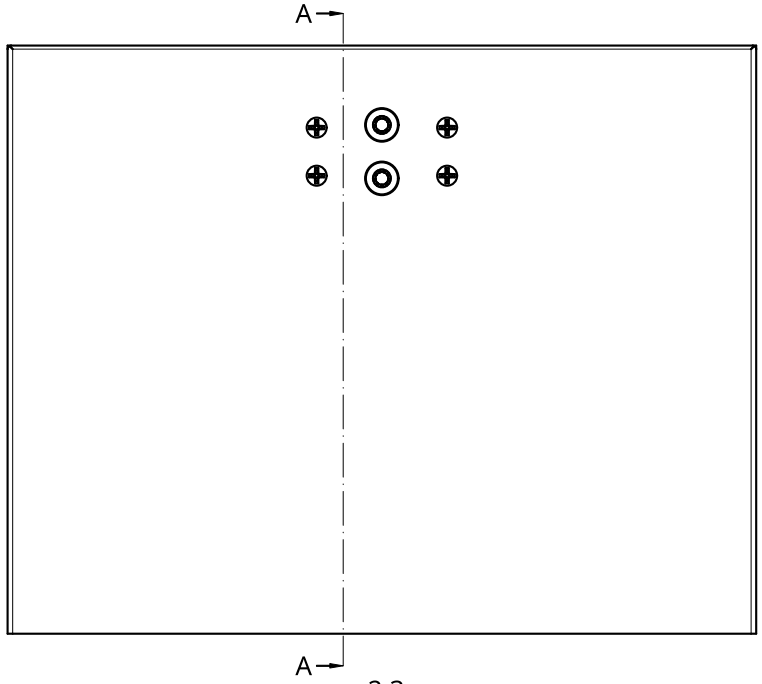
ANEXO 10.- MEDICIÓN DE VOLTAJE PARA RAZÓN DIGITAL DE BATERÍAS

Dato Digital de Microcontrolador	Voltaje Medido (V)	Razón
3009	12.22	0.004061149883682
2953	12.11	0.004100914324416
2891	12.09	0.004181943964026
2895	11.89	0.004107081174439
2903	11.81	0.004068205304857
2624	11.06	0.00421493902439
3101	12.62	0.004069654950016
3061	12.55	0.004099967330938
3025	12.45	0.004115702479339
3002	12.42	0.004137241838774
2981	12.27	0.004116068433412
2942	12.18	0.004140040788579
2941	12.13	0.004124447466848
2925	12.08	0.004129914529915
2890	11.91	0.004121107266436
2867	11.88	0.004143704220439
2675	11.21	0.004190654205607
	Promedio	0.004124866893301

ANEXO 11.- PLANOS DE ESTACIÓN DE RECARGA



A - A
2:3



2:3

No. DE PIEZA	DENOMINACIÓN	No. DE NORMA/DIBUJO	MATERIAL	No. DE ORDEN	No. DEL MODELO/SEMIPRODUCTO	PESO g/PIEZA	OBSERVACIONES
2	Puerto macho conector banana	PZ # 8	Nickel	8	--	5.24	
2	Portaelectrodos	PZ # 7	Ácido Poliláctico	7	–	5.00	
1	Base frontal	PZ # 6	Ácido Poliláctico	6	--	174.00	
4	Resortes	PZ # 5	Acero de resorte	5	3mmx8mm	0.025	
1	Soporte de portaelectrodos	PZ # 4	Ácido Poliláctico	4	–	13.00	
1	Tapa trasera	PZ # 3	Ácido Poliláctico	3	–	66.00	
8	Tuerca hexagonal	PZ # 2	Acero al carbono	2	M3x0.5	0.39	
8	Perno de cabeza plana	PZ # 1	Acero al carbono	1	M3x0.5x10	1.18	

TOLERANCIAS
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MILIMETROS
ACABADO SUPERFICIAL: N7
TOLERANCIAS:
LINEAL: 0.1
ANGULAR: 0.1

PESO
286g

MATERIALES
VER LISTA DE MATERIALES

	FECHA	NOMBRE
DIB.	27/7/21	STEVEN SILVA
REV.		
APROB.		

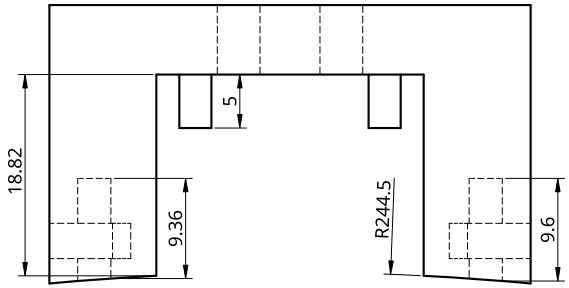
DENOMINACIÓN
ESTACIÓN DE AUTO-RECARGA
PLANO CONJUNTO

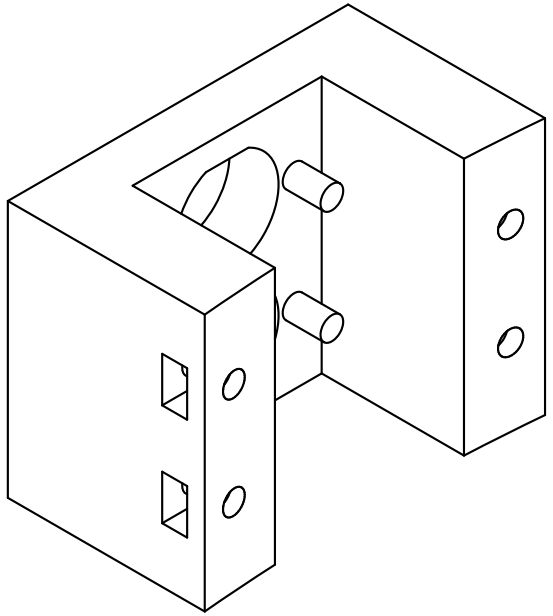
ESCALA
2:3

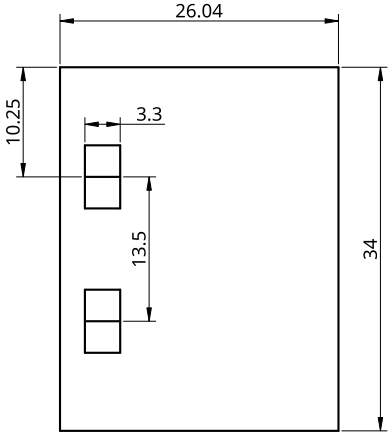
NÚMERO DEL DIBUJO
MCT-21-01

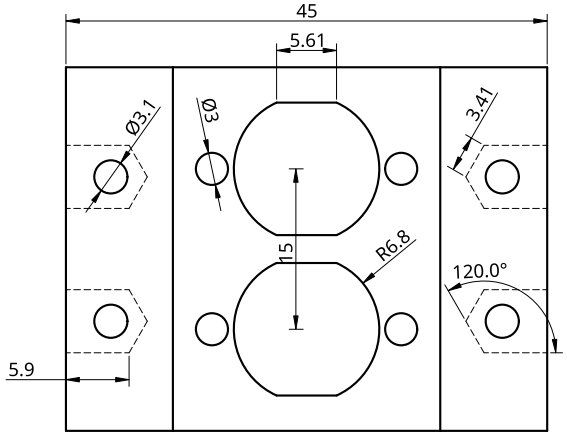
SUSTITUYE A:

EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE

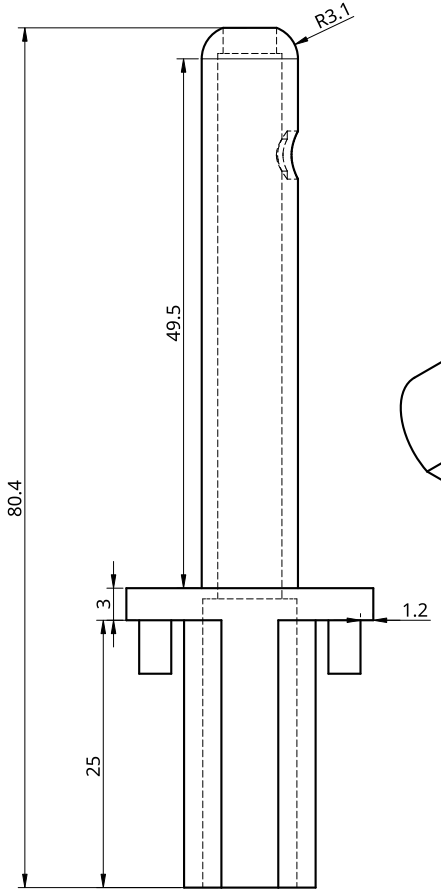


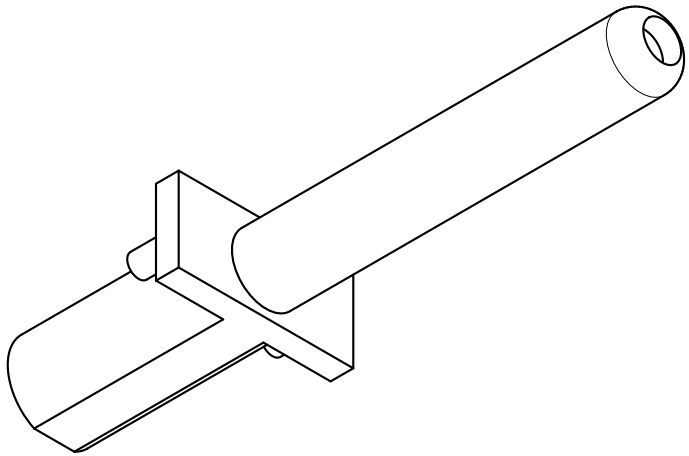


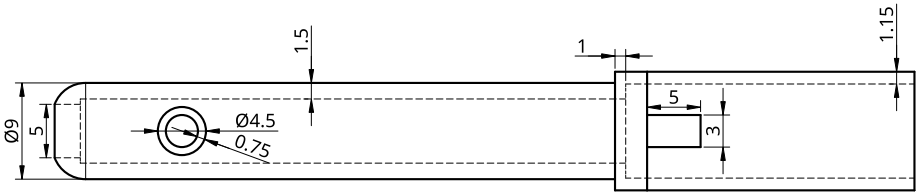


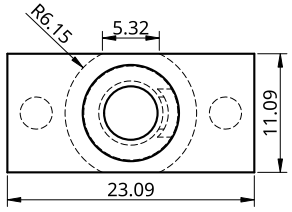


				TOLERANCIAS SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1	PESO 13g	MATERIALES ÁCIDO POLILÁCTICO	
					FECHA	NOMBRE	DENOMINACIÓN SOPORTE DE PORTAELECTRODOS	ESCALA 2:1
				DIB.	28/7/21	SILVA STEVEN		
				REV.	-/-/14			
				APROB.	-/-/14			
							NÚMERO DEL DIBUJO MCT-21-02	
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:	









EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE

TOLERANCIAS

SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MILIMETROS
ACABADO SUPERFICIAL: N7
TOLERANCIAS:
LINEAL: 0.1
ANGULAR: 0.1

PESO: 5g

MATERIALES: ÁCIDO POLILÁCTICO

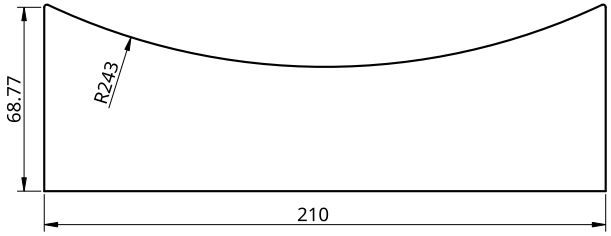
	FECHA	NOMBRE
DIB.	28/7/21	SILVA STEVEN
REV.	-/-/14	
APROB.	-/-/14	

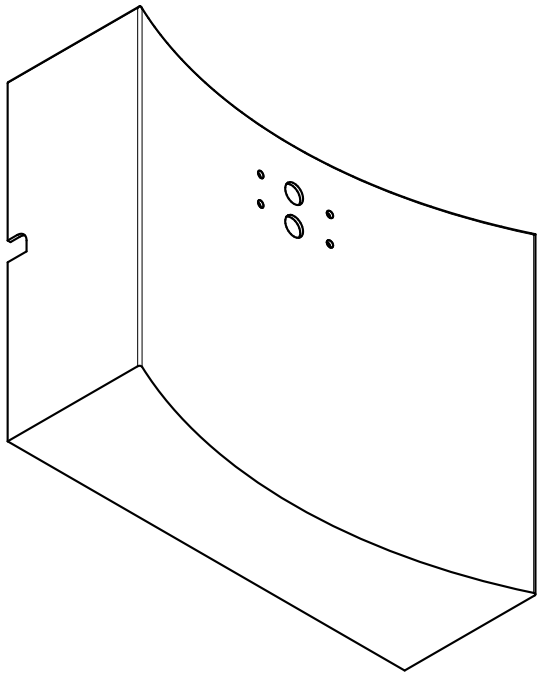
DENOMINACIÓN: PORTAELECTRODOS

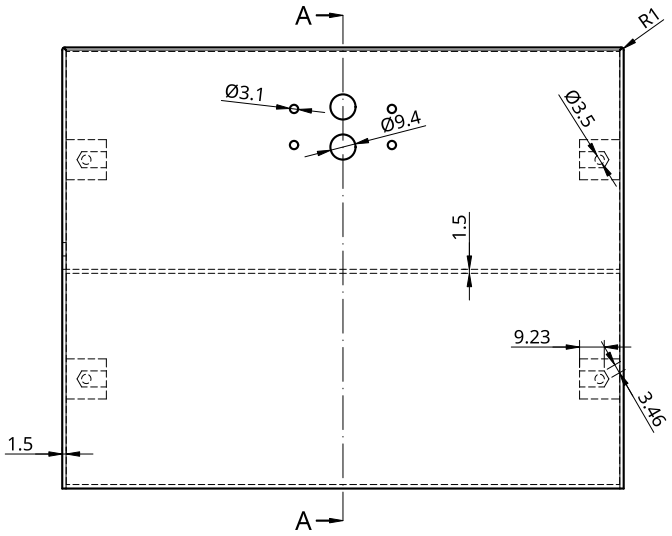
ESCALA: 2:1

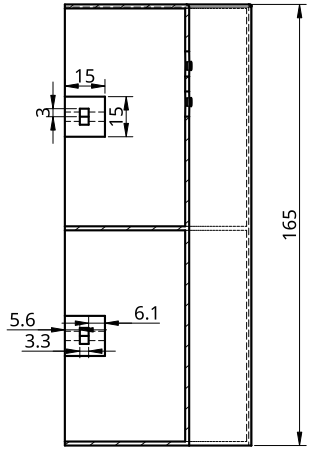
NÚMERO DEL DIBUJO: MCT-21-03

SUSTITUYE A:



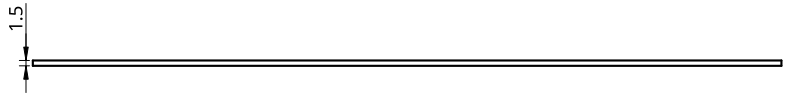


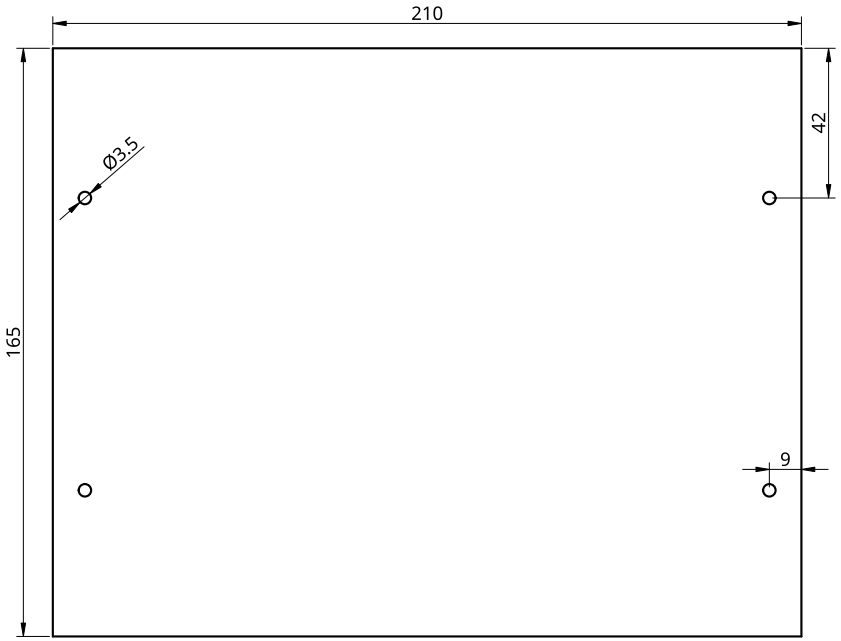


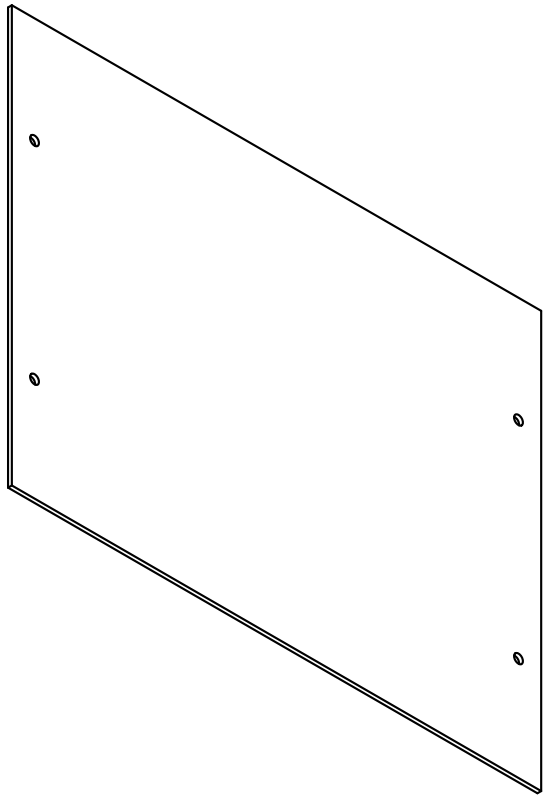


SECCIÓN A - A

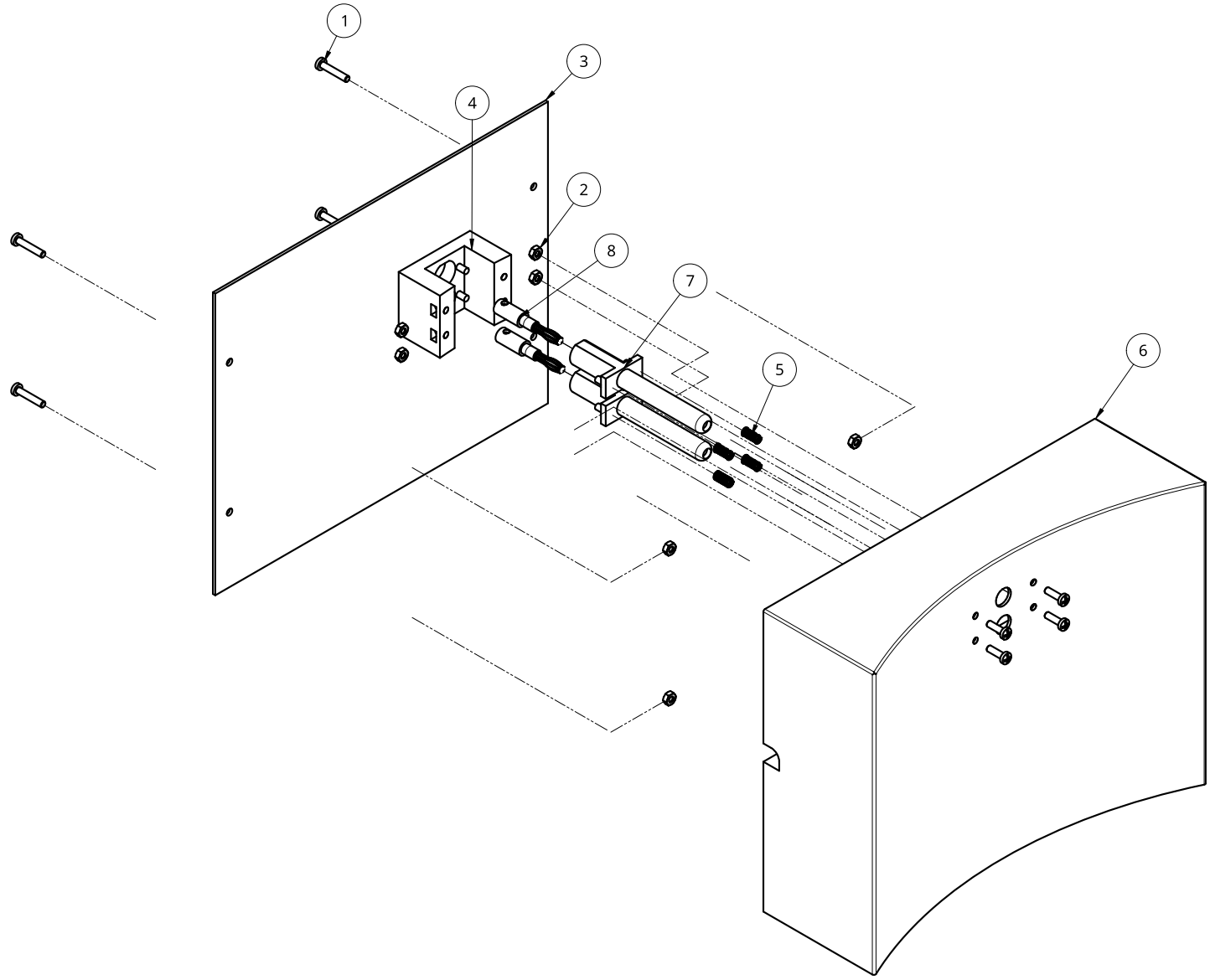
				TOLERANCIAS		PESO	MATERIALES
				SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1		174g	ÁCIDO POLILÁCTICO
					FECHA	NOMBRE	DENOMINACIÓN / ESCALA
				DIB.	28/7/21	SILVA STEVEN	BASE FRONTAL / 1:2
				REV.	-/-/14		
				APROB.	-/-/14		
							NÚMERO DEL DIBUJO: MCT-21-04
EDI-CIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:







				TOLERANCIAS		PESO	MATERIALES
				SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1		66g	ÁCIDO POLILÁCTICO
					FECHA	NOMBRE	DENOMINACIÓN / ESCALA
				DIB.	28/7/21	SILVA STEVEN	TAPA TRASERA — 2:3
				REV.	-/-/14		
				APROB.	-/-/14		
							NÚMERO DEL DIBUJO: MCT-21-05
EDI-CIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:



				TOLERANCIAS	PESO	MATERIALES
				SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1	286g	ÁCIDO POLILÁCTICO
					FECHA	NOMBRE
				DIB.	28/7/21	SILVA STEVEN
				REV.	-/-/14	
				APROB.	-/-/14	
EDI-CIÓN	MODIFICACIÓN	FECHA	NOMBRE			

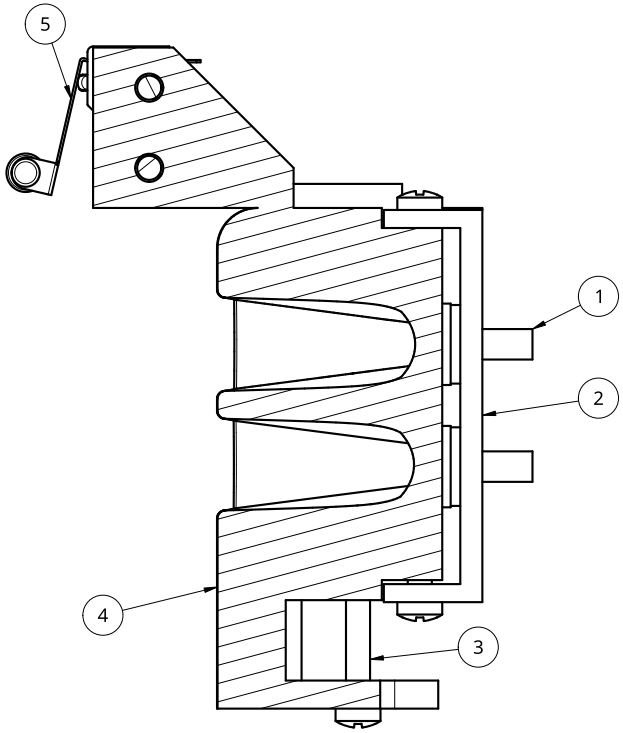
DENOMINACIÓN
ESTACIÓN DE AUTO-RECARGA
VISTA ISOMÉTRICA EXPLOSIONADA

ESCALA
1:2

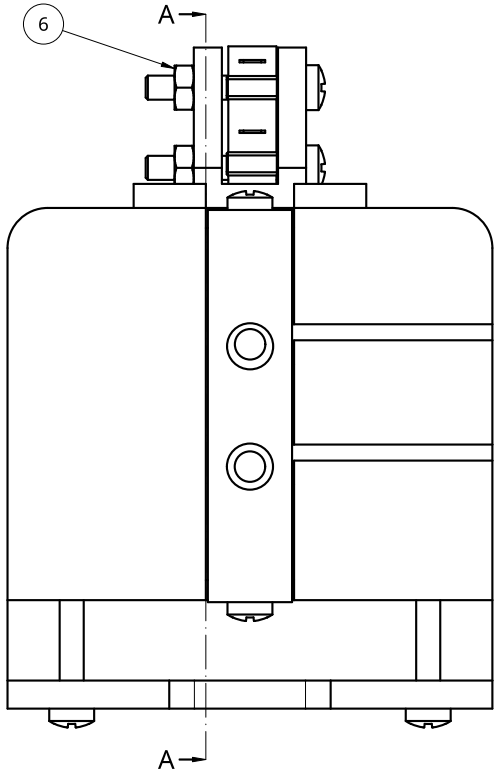
NÚMERO DEL DIBUJO
MCT-21-00

SUSTITUYE A:

ANEXO 12.- PLANOS DE PUERTO DE RECARGA



SECCIÓN A - A



6	Tuerca hexagonal	PZ # 6	Acero al carbono	6	M3x0.5	0.39	
1	Switch de limite	PZ # 5	Plastico	5	--	2.00	
1	Base de puerto de recarga	PZ # 4	Ácido Poliláctico	4	--	43.00	
6	Pernos de cabeza plana	PZ # 3	Acero al carbono	3	M3x0.5x10	1.18	
1	Abrazadera de puertos	PZ # 2	Ácido Poliláctico	2	–	2.00	
2	Puerto hembra de conector banana	PZ # 1	Nickel	1	5mm diametro	3.78	
No. DE PIEZA	DENOMINACIÓN	No. DE NORMA/DIBUJO	MATERIAL	No. DE ORDEN	No. DEL MODELO/SEMIPRODUCTO	PESO g/PIEZA	OBSERVACIONES

EDI-CIÓN	MODIFICACIÓN	FECHA	NOMBRE

TOLERANCIAS
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MILIMETROS
ACABADO SUPERFICIAL: N7
TOLERANCIAS:
LINEAL: 0.1
ANGULAR: 0.1

PESO
64g

MATERIALES
VER LISTA DE MATERIALES

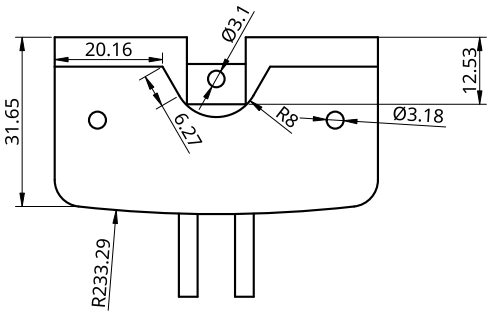
	FECHA	NOMBRE
DIB.	25/7/21	STEVEN SILVA
REV.	-/-/14	
APROB.	-/-/14	

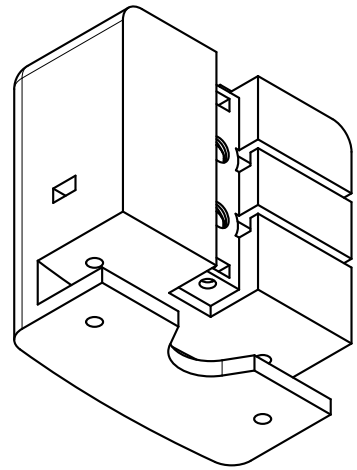
DENOMINACIÓN
PUERTO PARA AUTO-RECARGA
PLANO CONJUNTO

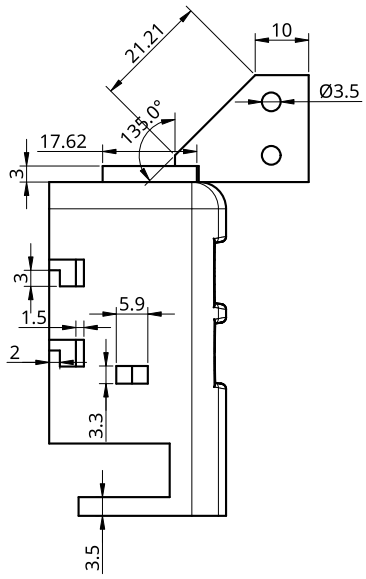
ESCALA
3:2

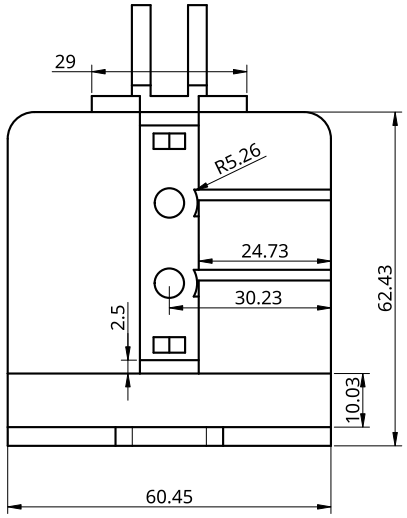
NÚMERO DEL DIBUJO
MCC-21-01

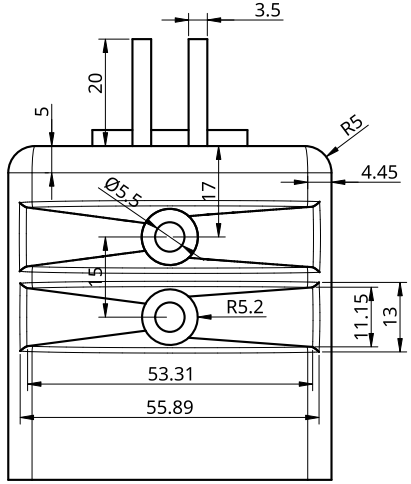
SUSTITUYE A:



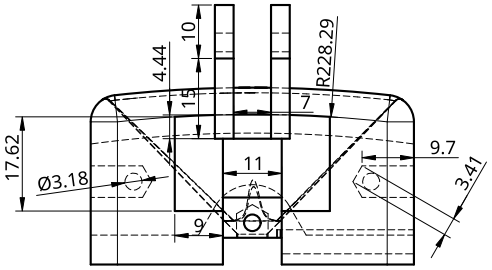




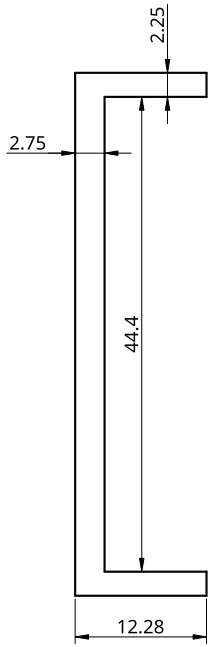


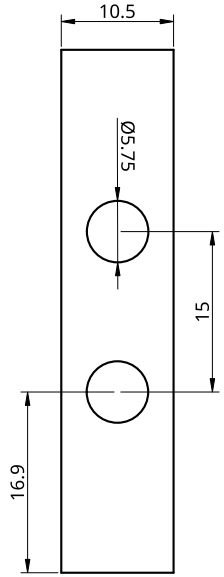


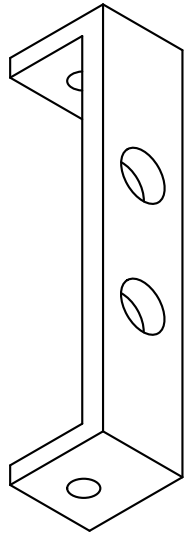
Posterior



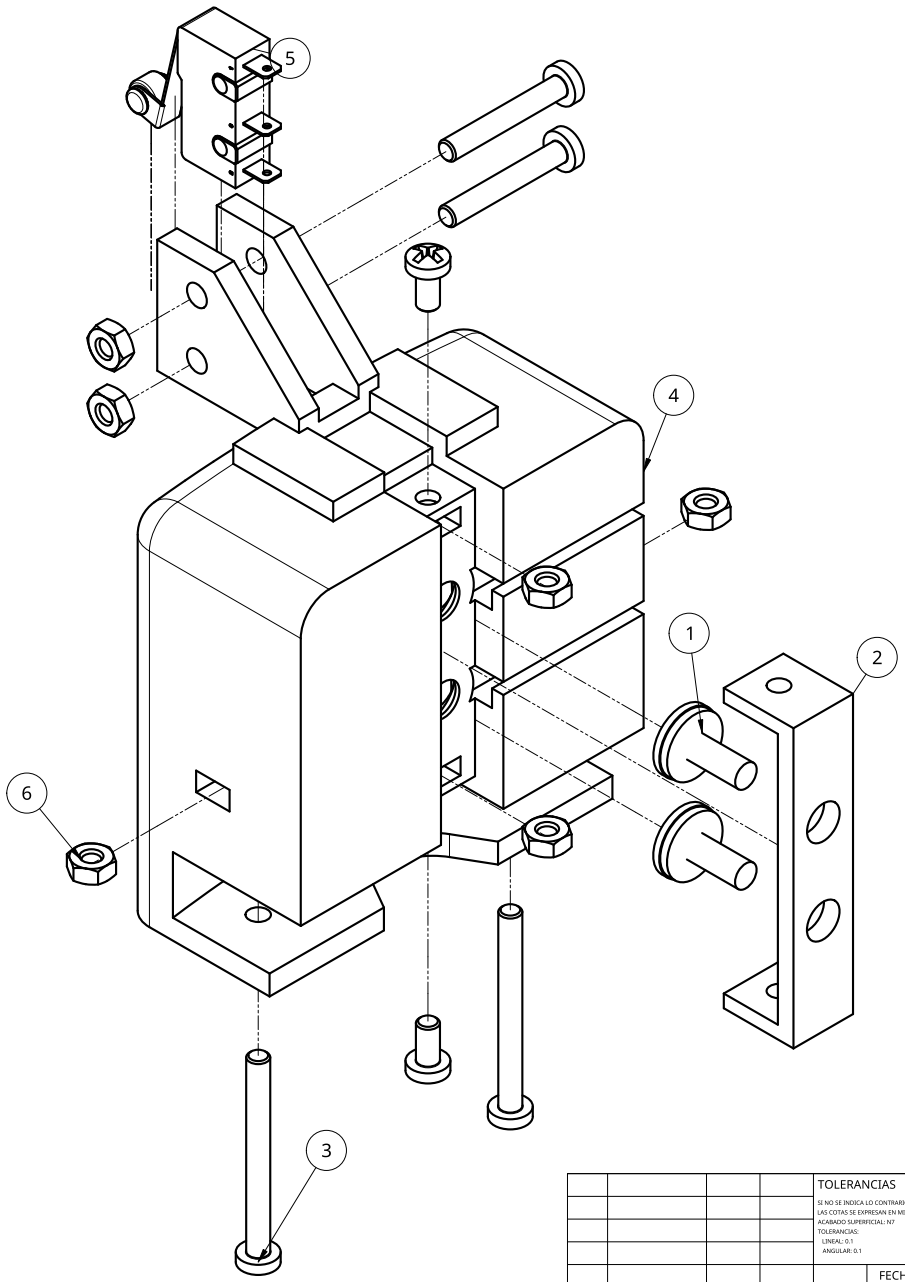
				TOLERANCIAS		PESO	MATERIALES
				SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1		43g	ÁCIDO POLILÁCTICO
					FECHA	NOMBRE	DENOMINACIÓN / ESCALA
				DIB.	25/7/21	STEVEN SILVA	BASE DE PUERTO PARA AUTO-RECARGA — 1:1
				REV.	-/-/14		
				APROB.	-/-/14		
							NÚMERO DEL DIBUJO: MCC-21-02
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:





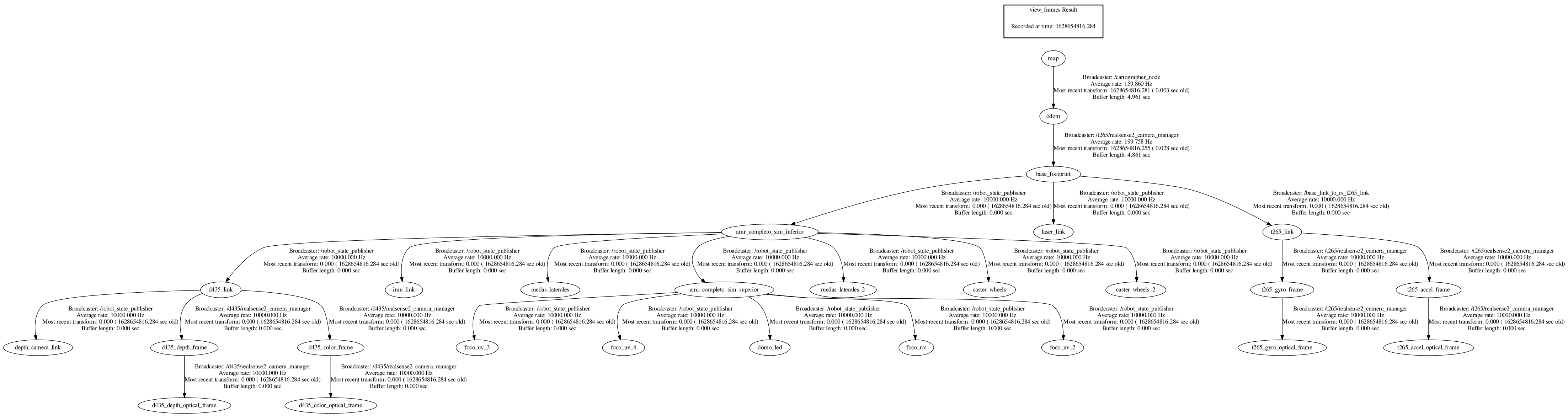


				TOLERANCIAS		PESO	MATERIALES
				SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1		2g	ÁCIDO POLILÁCTICO
					FECHA	NOMBRE	DENOMINACIÓN: ABRAZADERA DE PUERTOS
				DIB.	25/7/21	STEVEN SILVA	ESCALA: 2:1
				REV.	-/-/14		
				APROB.	-/-/14		
							NÚMERO DEL DIBUJO: MCC-21-03
EDICIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

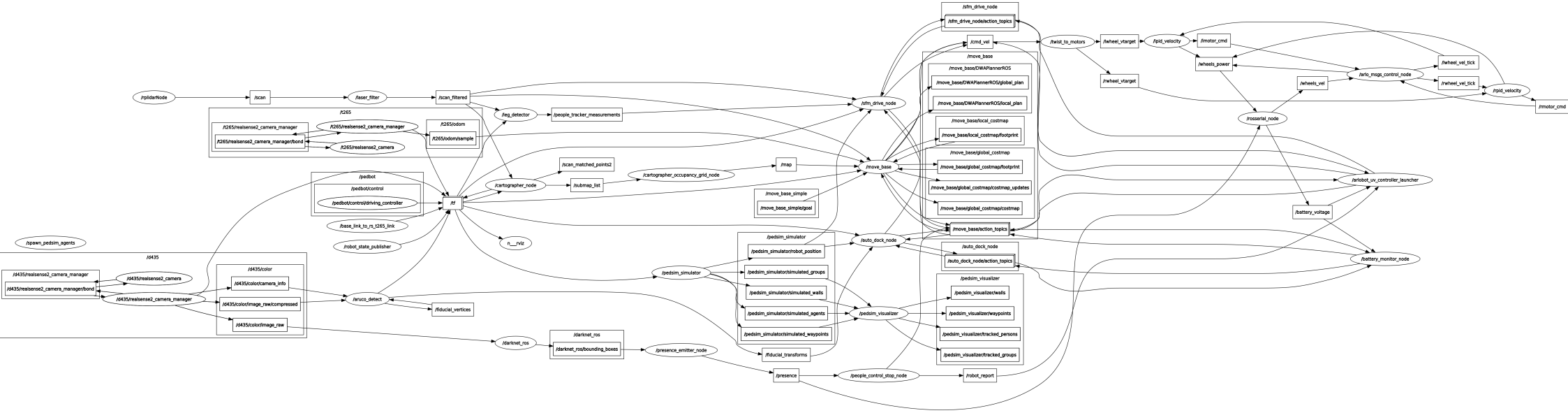


				TOLERANCIAS		PESO	MATERIALES
				SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MILIMETROS ACABADO SUPERFICIAL: N7 TOLERANCIAS: LINEAL: 0.1 ANGULAR: 0.1		64g	ÁCIDO POLILÁCTICO
					FECHA	NOMBRE	DENOMINACIÓN / ESCALA
				DIB.	25/7/21	STEVEN SILVA	PUERTO DE AUTO-RECARGA VISTA ISOMÉTRICA EXPLOTADA — 3:2
				REV.	-/-/14		
				APROB.	-/-/14		
							NÚMERO DEL DIBUJO: MCC-21-00
EDI-CIÓN	MODIFICACIÓN	FECHA	NOMBRE				SUSTITUYE A:

ANEXO 13.- TRANSFORMADAS DE ESLABONES DE ROBOT EN ROS


view_frames Result
Recorded at time: 1628654816.284
map
Broadcaster: /cartographer_node
Average rate: 159.860 Hz
Most recent transform: 1628654816.281 (0.003 sec old)
Buffer length: 4.961 sec
odom
Broadcaster: /t265/realsense2_camera_manager
Average rate: 199.758 Hz
Most recent transform: 1628654816.255 (0.028 sec old)
Buffer length: 4.861 sec
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1628654816.284 sec old)
Buffer length: 0.000 sec
Broadcaster: /base_link_to_rs_t265_link
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1628654816.284 sec old)
Buffer length: 0.000 sec
amr_completo_sim_inferior
laser_link
t265_link
d435_link
imu_link
ruedas_laterales
amr_completo_sim_superior
ruedas_laterales_2
caster_wheels
caster_wheels_2
t265_gyro_frame
t265_accel_frame
depth_camera_link
d435_depth_frame
d435_color_frame
foco_uv_3
foco_uv_4
domo_led
foco_uv
foco_uv_2
t265_gyro_optical_frame
t265_accel_optical_frame
d435_depth_optical_frame
d435_color_optical_frame

ANEXO 14.- GRÁFICO DE NODOS EN ROS


/rplidarNode
/scan
/laser_filter
/scan_filtered
/leg_detector
/people_tracker_measurements
/f265
/f265/realsense2_camera_manager
/f265/realsense2_camera_manager/bond
/f265/realsense2_camera_manager
/f265/realsense2_camera
/f265/odom
/f265/odom/sample
/pedbot
/pedbot/control
/pedbot/control/driving_controller
/base_link_to_rs_t265_link
/robot_state_publisher
/tf
/cartographer_node
/scan_matched_points2
/submap_list
/cartographer_occupancy_grid_node
/map
/move_base_simple
/move_base_simple/goal
/move_base
/sfm_drive_node
/sfm_drive_node/action_topics
/cmd_vel
/move_base
/move_base/DWAPlannerROS
/move_base/DWAPlannerROS/global_plan
/move_base/DWAPlannerROS/local_plan
/move_base/local_costmap
/move_base/local_costmap/footprint
/move_base/global_costmap
/move_base/global_costmap/footprint
/move_base/global_costmap/costmap_updates
/move_base/global_costmap/costmap
/move_base/action_topics
/twist_to_motors
/wheel_vtarget
/wheel_vtarget
/pid_velocity
/motor_cmd
/wheels_power
/rosserial_node
/arlo_msgs_control_node
/wheel_vel_tick
/wheel_vel_tick
/wheels_vel
/pid_velocity
/rmotor_cmd
/arlobot_uv_controller_launcher
/battery_voltage
/battery_monitor_node
/spawn_pedsim_agents
/d435
/d435/realsense2_camera_manager
/d435/realsense2_camera_manager/bond
/d435/realsense2_camera
/d435/realsense2_camera_manager
/d435/color
/d435/color/camera_info
/d435/color/image_raw/compressed
/d435/color/image_raw
/aruco_detect
/fiducial_vertices
/darknet_ros
/darknet_ros
/darknet_ros/bounding_boxes
/presence_emitter_node
/pedsim_simulator
/pedsim_simulator/robot_position
/pedsim_simulator/simulated_groups
/pedsim_simulator/simulated_walls
/pedsim_simulator/simulated_agents
/pedsim_simulator/simulated_waypoints
/fiducial_transforms
/pedsim_simulator
/pedsim_visualizer
/pedsim_visualizer/walls
/pedsim_visualizer/waypoints
/pedsim_visualizer/tracked_persons
/pedsim_visualizer/tracked_groups
/pedsim_visualizer
/auto_dock_node
/auto_dock_node
/auto_dock_node/action_topics
/presence
/people_control_stop_node
/robot_report
n__rviz