FABRICACIÓN DE UN ROBOT MÓVIL REPARTIDOR CON SISTEMA DE GENERACIÓN Y SEGUIMIENTO DE TRAYECTORIAS PARA EL CENTRO DE TELEINFORMÁTICA Y PRODUCCIÓN INDUSTRIAL DEL SENA DE POPAYÁN

Universidad del Cauca ®

Proyecto de grado

GABRIEL ALEJANDRO DIAZ FIERRO

Director
Ph.D Carlos Felipe Rengifo Rodas
Asesor de la empresa:
Msc. Julián Alexis Arana

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
INGENIERÍA EN AUTOMÁTICA INDUSTRIAL
POPAYÁN, 2025

Gabriel Alejandro Díaz Fierro

FABRICACIÓN DE UN ROBOT MÓVIL REPARTIDOR CON SISTEMA DE GENERACIÓN Y SEGUIMIENTO DE TRAYECTORIAS PARA EL CENTRO DE TELEINFORMÁTICA Y PRODUCCIÓN INDUSTRIAL DEL SENA DE POPAYÁN

Trabajo de grado en modalidad de práctica profesional presentado a la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca para obtener el título de:

Ingeniero en Automática Industrial

Popayán, 2025

Aunque a veces, en la vida,
pareciera que el esfuerzo no es recompensado,
lo único que conmueve para seguir adelante
es saber que la grandeza trasciende la existencia,
y quien ha de morir, deje su luz a la posteridad.

"Posteris Lvmen Moritvrvs Edat".

Dedicado con orgullo a mi madre,
Alejandra María
que con amor y esfuerzo
siempre estuvo presente.

Con especial gratitud al
Ph.D. Carlos Felipe Rengifo Rodas,
por compartir su conocimiento.
Alguien admirable.
Espero, algún día,
saber tanto como él.

Índice general

Índice de figuras

Lista de Tablas

Capítulo 1

Preliminares

1.1. Introducción

La robótica móvil ha experimentado una evolución significativa en los últimos años, permitiendo la automatización de diversas tareas repetitivas, como la logística y la entrega de paquetes en entornos cotidianos e industriales sin intervención humana. La localización de un robot móvil se representa comúnmente mediante un estado que define su postura, incluyendo su posición y orientación dentro de un mapa desconocido [2]. Para alcanzar esta capacidad, se han desarrollado múltiples métodos durante más de dos décadas. Una de las más destacadas es la localización y mapeo simultáneos *(SLAM, Simultaneous Localization And Mapping*), que permite a un robot construir un mapa del entorno mientras estima su propia ubicación dentro de él [3]. Este enfoque ha sido fundamental en aplicaciones de navegación autónoma [4] y ha sido objeto de análisis y desarrollo en estudios clásicos del problema *SLAM* [5].

Una de las variables más importantes dentro de *SLAM* para la robótica móvil es el análisis de la localización, el cual estima sus posturas en entornos conocidos utilizando los datos de un sensor o cualquier otro método, es decir, dentro de un mapa previamente reconstruido o dado, a ello se le denomina localización [6].

A diferencia de la localización, el mapeo es esencial para que un robot móvil encuentre su ubicación en el entorno y complete sus tareas de ruta en consecuencia. El mapa es un conjunto de características que describen el entorno, como paredes, obstáculos, puntos de referencia. Por tanto, uno de los métodos usados para completar el mapeo o reconstrucción del espacio desconocido es *Visual SLAM* mediante algoritmos de visión por computadora [7].

Para el mapeo, registro o reconstrucción es necesario una cámara o sensor con la capacidad de obtener datos del entorno de interés, la información resultante será una secuencia de imágenes de RGB-D. Con dichos fotogramas tridimensionales se logra realizar una reconstrucción completa, según lo propuesto por [8] y mejoradas con el registro rápido global [9] con gráfico de poses, función de optimización y cierre de bucle.

A fin de realizar *Visual SLAM*, son necesarias librerías de código abierto y multipropósito, pero suelen estar en su mayoría sobre sistemas GNU/Linux [10] y requieren un moderado manejo de programación debido a su operabilidad sobre ROS *(Robot Operating System)* [11].

En la actualidad, los siguientes algoritmos son ampliamente utilizados para *Visual SLAM: RANSAC (Random Sample Consensus)* importante para encontrar patrones en datos con alta cantidad de ruido o valores atípicos *(outliers)* [12] [13], *FPFH (Fast Point Feature Histograms)* para la búsqueda de características similares entre pares de imágenes [14] e *ICP (Iterative Closest Point)* [15] para obtener el emparejamiento de un conjunto de datos (3D o 2D) y lograr la transformación matricial de una imagen RGB-D fuente hacia una objetivo (Reconstrucción). En este sentido, Open3D, librería multiplataforma de licencia MIT, puede ser ejecutada sobre Windows, MAC OS y GNU/Linux [16], mientras que sus módulos son programables, según sea la necesidad, tanto en C++ como en Python.

Por otro lado, mientras *Visual SLAM* realiza la localización y reconstrucción sobre un espacio de trabajo y genera una nube de puntos tridimensional con coordenadas globales, es importante que un robot móvil las siga según su actividad.

Para conseguir el seguimiento de puntos mencionado con anterioridad, es necesario aplicar teoría de control y manipular el robot a través de su modelo físico, lo que permite seguir una trayectoria específica. Aunque el tema de modelos y tipos de robots que posibilitan lo anterior es extenso, resulta crucial la selección de un modelo adecuado en función de la maniobrabilidad y controlabilidad, las cuales tienden a ser inversamente proporcionales, según se indica en [1]. Además, la implementación de estrategias de control avanzadas, tal como se expone en [17], facilita la integración entre el modelo cinemático y la ejecución de trayectorias. A modo de ejemplo y para simplificar el análisis, se ha optado por la configuración omnidireccional de tres ruedas suecas a noventa grados propuesta por [1], la cual proporciona un modelo cinemático completo, incluyendo la matriz jacobiana directa e inversa necesarias para implementar un control básico. Adicionalmente, estudios en control robótico, como el presentado en [18], refuerzan la importancia de integrar la teoría de control en el diseño de sistemas robóticos.

Finalmente, una vez explicados los algoritmos y la teoría básica necesaria para que un robot móvil realice una tarea autónoma dentro de un espacio, se evidencia la dificultad técnica en el uso de estas tecnologías con fines académicos.

En esta situación, el Centro de Teleinformática y Producción Industrial del SENA Popayán percibe la necesidad de crear un robot móvil repartidor de pequeña escala junto a una herramienta software intuitiva, que genere trayectorias, las siga, funcione sobre *Windows* y además no requiera ROS ni habilidades en programación, ayudando así a lograr un aprendizaje confortable en su manipulación sin necesidad de conocer el trasfondo de la codificación, *Visual SLAM* o control cinemático.

1.2. Objetivos

1.2.1. Objetivo general

Construir un robot móvil repartidor junto con su herramienta software para reconstrucción, generación de trayectorias y su seguimiento en espacios tridimensionales estáticos del Centro de Teleinformática y Producción Industrial del SENA de Popayán.

1.2.2. Objetivos específicos

- Fabricar un robot móvil repartidor omnidireccional de tres ruedas suecas a noventa grados que entregue un paquete de un punto hacia otro.
- Realizar la reconstrucción, generación de trayectorias y seguimiento las mismas en entornos 3D mediante el uso de sensores de profundidad y algoritmos de visión por computadora.
- Programar la interfaz gráfica con visualizador de reconstrucción, generación de trayectorias y su respectivo seguimiento en espacios 3D estáticos para controlar el robot móvil repartidor.

1.3. Estructura del documento

Este documento se distribuye de la siguiente manera: En el Capítulo 1, se aborda la introducción, proporcionando el contexto que enmarca la aplicación del trabajo de grado, identificando la problemática central del Centro de Teleinformática y Producción Industrial del SENA de Popayán y formulando la correspondiente solución. Además, se presentan las teorías necesarias que guiarán el desarrollo de la solución, estableciendo así las metas específicas que se pretenden alcanzar. En el Capítulo 2, se lleva a cabo un análisis de las investigaciones y desarrollos previos relacionados con el tema de estudio. En el Capítulo 3 se describe detalladamente el desarrollo del proyecto y en el Capítulo 4 la validación de la solución presentada y el análisis de los resultados obtenidos, conclusiones y posibles trabajos futuros.

Capítulo 2

2.1. Estado del arte

En este capítulo se examina el robot omnidireccional de tres ruedas, su modelo y las tecnologías más utilizadas para permitir su autonomía en interiores. Además, se destaca la relevancia de los algoritmos de visión por computadora y las plataformas de código abierto disponibles para implementar *Visual SLAM*. La exploración de estas tecnologías permite apreciar los avances recientes en el campo, resaltando sus aplicaciones y lo que contribuye a una comprensión más integral de las capacidades actuales de los robots móviles de tres ruedas para la entrega de paquetes.

2.1.1. Control en robots omnidireccionales de tres ruedas

La investigación reciente en robots omnidireccionales de tres ruedas ha abordado diversos aspectos que resaltan su versatilidad y aplicabilidad en distintos contextos. Por ejemplo,[19] se enfoca en el diseño y control, resaltando no solo la alta maniobrabilidad y eficacia en entornos industriales y logísticos, sino también la implementación de estrategias de control robustas que aseguran un desempeño estable en condiciones variables. De manera similar, [20] profundiza en el modelado cinemático y el control para la navegación en interiores, destacando la integración de estos elementos para optimizar la precisión y estabilidad en entornos complejos, y subrayando la importancia de una distribución adecuada de fuerzas para mejorar el rendimiento dinámico. Asimismo, [21] orienta su trabajo hacia la mejora de la interacción humano-robot, no solo para aplicaciones de servicio y asistencia, sino también mediante la integración de algoritmos de visión por computadora que facilitan la detección y el seguimiento de seres humanos en entornos colaborativos. Para lograr lo anterior, los robots incorporan tecnologías de control, programación, visión por computadora, inteligencia artificial, percepción e instrumentación que facilita la interacción con su entorno y la realización de tareas complejas sin la intervención directa de los humanos [22]. La robótica móvil se considera un dominio emergente de la alta tecnología que involucra realidades de amplia complejidad que puede tener diferentes aplicaciones industriales y ofrecer soluciones tecnológicas innovadoras, tales como la entrega de paquetes y logística.

Por otro lado, en la robótica educativa se tiene como ejemplo la plataforma robótica de tres ruedas omnidireccional propuesta por [23] denominada (ROBOTONT, *Open-source*

and ROS-supported omnidirectional mobile robot for education and research) que integra SLAM 2D, mapeo 3D, AR tag tracking (seguimiento de marcadores AR), para que el robot pueda identificar, localizar y seguir objetos o puntos de interés en su entorno utilizando marcadores visuales impresos (como códigos QR especiales), operación remota y manipulación con gestos.

Para lograr lo anterior, la disciplina de control es una de las mas importantes en entornos colaborativos, pues permite ajustar dinámicamente la respuesta del sistema ante variaciones en el entorno con el uso de sensores para la recolección y fusión de datos, que consiste en combinar información proveniente de múltiples sensores con el fin de obtener una representación más precisa, robusta y coherente del entorno[24].

También, la estimación precisa de los parámetros de la cinemática de un robot móvil omnidireccional es fundamental para mejorar la exactitud de su localización y navegación autónoma. En este contexto, se han propuesto distintas metodologías para mitigar los errores sistemáticos derivados de imperfecciones en la configuración física del robot. Una de estas metodologías es la presentada en [25], donde se introducen los Parámetros Cinemáticos Efectivos *(EKPs, Effective Kinetic Parameters*) para ajustar la matriz cinemática del robot mediante la minimización de una función de error basada en pruebas controladas, lo que permitió mejorar sustancialmente el seguimiento de trayectorias tanto en simulación como en pruebas experimentales. Por otro lado, en [26] se plantea una estrategia de calibración no paramétrica utilizando algoritmos genéticos, los cuales optimizan la matriz cinemática inversa del robot comparando trayectorias reales con las esperadas, logrando una mejora significativa en la odometría con una reducción promedio del error del 82 %. Estos enfoques destacan la importancia de una calibración precisa para aumentar la fiabilidad del robot en tareas críticas como la entrega autónoma de paquetes en entornos interiores.

2.1.2. Algoritmos más utilizados para *SLAM*

La información del entorno puede ser capturada en tres dimensiones mediante sensores como cámaras 3D para la representar detalladamente el entorno. Esta capacidad facilita el emparejamiento entre pares de datos, como capturas de profundidad tomadas en diferentes instantes, permitiendo transformar una captura 3D de origen hacia otra de destino. El emparejamiento de capturas 3D mencionadas anteriormente son posibles gracias a la combinación de algoritmos variados, el algoritmo ICP es uno de los más importantes, usado para el registro global de una secuencia de fotogramas RGB-D [15] junto con otros algoritmos para extraer antes sus características.

Es importante destacar que, para llevar a cabo la transformación entre un conjunto de datos A y un conjunto de datos B, los algoritmos de descripción de características juegan un papel fundamental. Estos algoritmos permiten identificar y representar puntos clave únicos dentro de cada conjunto de datos tridimensionales, generando descriptores que capturan propiedades geométricas o espaciales distintivas. A través de estos descriptores, es

posible establecer correspondencias confiables entre los dos conjuntos, lo cual es un paso esencial para tareas como la alineación, el registro o la fusión de datos en sistemas de percepción 3D. En este sentido, los algoritmos utilizados tanto para la extracción de características como para el emparejamiento entre descriptores son elementos clave a resaltar.

Finalmente, para contextualizar, en la Tabla 2.1 se presentan los algoritmos más utilizados para la extracción de características y el emparejamiento entre conjuntos de datos tridimensionales. Estos algoritmos son fundamentales en procesos de percepción espacial, como la generación de mapas 3D de entornos desconocidos, ya que permiten identificar puntos clave representativos y establecer correspondencias entre diferentes nubes de puntos. Esta capacidad es esencial en aplicaciones como la localización y mapeo simultáneo *(SLAM)*, donde se requiere construir un modelo preciso del entorno a partir de observaciones parciales y sucesivas.

Algoritmo	Año	Referencia
RANSAC	1981	Random Sample Consensus (RANSAC) [27]
ICP	2001	Iterative Closest Point [15]
SURF	2006	Speed-up Robust Features [28]
SIFT	2009	Scale Invariant feature transform [29]
RIFT	2010	Rotation Invariant Feature Transform [30]
SUSAN	2010	Smallest Univalue Segment Assimilating Nucleus [31]
RSD	2012	Radius-based Surface Descriptor [32]
SHOT	2014	Signature of Histogram of Orientation [33]
3DSC	2023	3D Shape Context [34]

Tabla 2.1: Algoritmos para extracción de características y correspondencias de conjunto de datos

2.1.3. Plataformas de código abierto para *SLAM*

La utilización de las plataformas de código abierto existentes presenta diversas estrategias para abordar las dos problemáticas fundamentales de *SLAM*: el mapeo y la estimación de la trayectoria. El primero se encarga de capturar y reconstruir el entorno utilizando sensores, como cámaras RGB-D o LIDAR [35], mientras que la odometría visual se enfoca en estimar la trayectoria del robot en función de la secuencia de nubes de puntos adquiridas durante el movimiento [36]. Sin embargo, una gran parte de estas soluciones se encuentran estrechamente integradas al ecosistema *ROS*, el cual, aunque robusto y ampliamente utilizado en investigación, está diseñado principalmente para correr sobre distribuciones *Linux*, especialmente *Ubuntu*.

Esta dependencia limita la accesibilidad y adopción de tecnologías de *SLAM* en otros entornos operativos como *Windows*, que sigue siendo ampliamente utilizado tanto en el

ámbito académico como en aplicaciones comerciales. En este sentido, se vuelve necesario investigar enfoques más accesibles, intuitivos y compatibles con sistemas *Windows*, sin comprometer la precisión y eficiencia del sistema de localización y mapeo.

Una alternativa destacada en este contexto es *Open3D*, una biblioteca moderna y de código abierto orientada al procesamiento de datos 3D, que incluye herramientas eficientes para *Visual SLAM*, registro de nubes de puntos, segmentación y reconstrucción [16]. A diferencia de otras plataformas más acopladas a ROS, *Open3D* permite ejecutar *pipelines SLAM* de manera independiente, tanto en *Linux* como en *Windows*, facilitando su integración en entornos más diversos y reduciendo la curva de entrada para desarrolladores no familiarizados con ROS. Esta flexibilidad lo convierte en una herramienta ideal para el desarrollo de aplicaciones de entrega autónoma en interiores, donde la portabilidad y la facilidad de implementación son aspectos clave.

La Tabla 2.2 muestra las plataformas de SLAM de código abierto de última generación que integran aspectos clave como ROS y además tienen soporte por parte de sus desarrolladores.

Tabla 2.2: Plataformas código abierto de última generación

Framework	**Documentation**	**Support**	**Usage examples on popular datasets**	**Docker**	**ROS**
ORB-SLAM2	github	-	+	-	+
MapLab	github	+	+	-	+
LDSO	github	+	+	-	-
VINS-Mono	github	+	+	-	+
VINS-Fusion	github	+	+	-	+
OpenVSLAM	web-page, github	+	+	+	+
Basalt	github	+	+	+	-
Kimera	github	+	EuRoC only	+	+
OpenVINS	web-page, github	+	+	-	+
ORB-SLAM3	github	+	+	-	+
DRE	github	-	-	-	+

Capítulo 3

Materiales y Métodos

En este capítulo son abordados los materiales y métodos empleados en el desarrollo de este proyecto, se proporcionará una descripción de los recursos utilizados, así como de las estrategias y procedimientos implementados. Este análisis permitió comprender la base experimental y técnica sobre la cual se sustentan los resultados obtenidos, brindando una visión integral del enfoque metodológico empleado en la ejecución de la práctica.

3.1. Materiales

Esta sección describe los elementos utilizados para la construcción y puesta en marcha del robot móvil propuesto. Se detallan tanto los componentes electrónicos como mecánicos y de software, necesarios para garantizar el correcto funcionamiento del sistema. Cada componente fue seleccionado en función de los requerimientos del robot y priorizando la disponibilidad de los componentes en el Centro de Teleinformática y Producción Industrial del SENA, sede Popayán. Asimismo, se considera la facilidad de integración de dichos dispositivos para lograr una plataforma funcional.

3.1.1. Justificación de los componentes

Los componentes electrónicos utilizados en el desarrollo del robot ya se encontraban previamente adquiridos por el SENA, lo cual hizo innecesario diseñar circuitos impresos (PCB) personalizados. Debido a esto, no se realizaron comparaciones con otras opciones disponibles en el mercado, y los elementos disponibles fueron integrados directamente en la implementación del sistema robótico.

3.1.2. Diagrama de bloques

La arquitectura hardware del sistema se representa primero mediante un diagrama de conexiones general presentado en la Figura 3.1 que muestra la interacción entre los distintos módulos del robot. El diagrama general de componentes mencionado es fundamental porque proporciona una visión global del sistema antes de entrar en detalles. Este esquema ayuda a comprender cómo se integran los distintos componentes, permitiendo identificar

claramente las funciones de cada parte y su rol dentro del sistema completo.

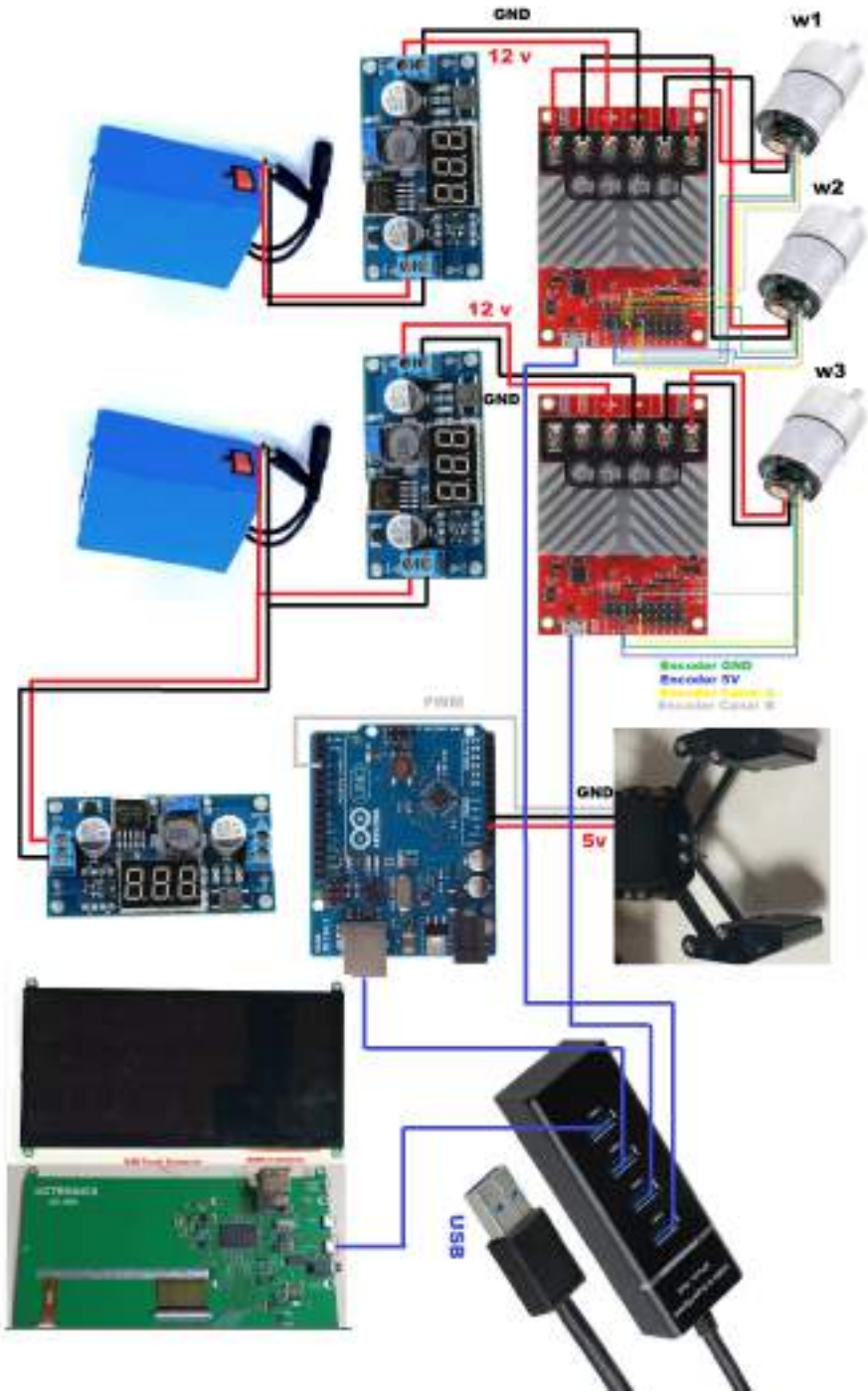


Figura 3.1: Diagrama general de conexiones

3.1.3. Sensores

Cámara RGB-D Intel RealSense D435i

Para la captura de profundidad se optó por una cámara Intel RealSense D435i presentada en la Figura 3.2, debido a la accesibilidad de presupuesto. Esta cámara es uno de los componentes principales de este proyecto, pues es un dispositivo RGB-D que destaca por su diseño compacto, alta resolución, amplio rango de detección y capacidad de visión estereoscópica que se utiliza para la captura de datos tridimensionales. Estas características permiten una mayor precisión en el seguimiento y la adquisición de datos convirtiendo la misma en una herramienta importante para el mapeo 3D y necesario para realizar *Visual SLAM*, alineándose perfectamente con los requerimientos del presente proyecto. En la Tabla 3.1 se listan las características principales de este dispositivo.

Figura 3.2: Cámara RGB-D Intel Realsense D435i

Tabla 3.1: Características de la Cámara RGB-D Intel Realsense D435i

Cámara Intel RealSense D435i
Procesador de visión Intel RealSense™ D4
Hasta 1280 x 720 de resolución de profundidad estéreo
Hasta 1920 x 1080 de resolución de RGB
Campo de visión diagonal de profundidad de más de 90°
Dos sensores de obturación global para una transmisión de profundidad de hasta 90 FPS
Alcance de 0.2 m a más de 3.0 m (varía según las condiciones de iluminación)
Unidad de medición inercial (IMU) para datos de 6 grados de libertad (6DoF)
Filtro de paso de IR

Encoder de cuadratura

El motor Pololu 4752 cuenta con un *encoder* de cuadratura integrado en la Figura 3.3, el cual permite medir de manera precisa la velocidad y la posición del eje del motor. Este *encoder* funciona mediante sensores de efecto *Hall* y genera 64 pulsos por cada vuelta

completa del eje del motor, antes de la reducción de la caja de engranajes. Al combinarse con una caja reductora, la resolución se incrementa significativamente, permitiendo obtener 1920 pulsos por vuelta del eje de salida, lo cual es ideal para aplicaciones que requieren un control fino del movimiento. El *encoder* emite dos señales desfasadas (canales A y B) que permiten no solo contar los pulsos, sino también identificar la dirección del giro. Esta información es fundamental para implementar sistemas de control en lazo cerrado.

Figura 3.3: *Encoder* de cuadratura integrado en Pololu 4752

3.1.4. Procesamiento

Arduino UNO

Se utilizó una placa Arduino UNO presentada en la Figura 3.4, debido a que las funciones asignadas a este microcontrolador eran básicas y específicas, actuando únicamente como un módulo esclavo encargado de accionar la pinza del robot. La comunicación entre el Arduino UNO y el computador principal se realizó a través de un canal serial por cable, sin necesidad de implementar ningún tipo de conexión inalámbrica, ya que no se requería autonomía en la comunicación desde el microcontrolador. Esta configuración resultó suficiente y eficiente para las tareas designadas, permitiendo una integración sencilla con el sistema.

La placa Arduino Uno utiliza comunicación serial para el control del efector final del robot, específicamente una pinza accionada mediante un servomotor. Esta placa microcontroladora es ampliamente utilizada por su facilidad de programación, versatilidad y compatibilidad con una amplia variedad de dispositivos electrónicos. En este caso, el Arduino Uno permitió controlar de forma independiente la apertura y cierre de la pinza, lo cual es crucial para la tarea de manipulación durante la entrega de objetos de un punto hacia otro, uno de los objetivos principales de la práctica.

Cabe resaltar que este tipo de actuadores, como los servomotores, requieren señales PWM que no pueden ser generadas directamente desde una computadora personal (PC). Por ello, el uso del Arduino se vuelve esencial, ya que actúa como un intermediario entre el sistema de alto nivel (por ejemplo, el software de control en la PC, Python) y los dispositivos físicos. Además, el Arduino puede ser reutilizado para controlar otros periféricos

adicionales en el futuro, como sensores auxiliares o actuadores, gracias a su capacidad de expansión mediante pines digitales y analógicos. Esta modularidad lo convierte en una herramienta valiosa dentro de sistemas robóticos escalables.

Figura 3.4: Arduino UNO

Unidad Central de Procesamiento (CPU)

Durante el desarrollo del sistema, fue necesario implementar transmisión de datos en streaming 3D y 2D para la visualización del entorno en tiempo real utilizando la cámara Intel RealSense, así como integrar el control de movimiento a través de los controladores RoboClaw 2x15A. Inicialmente, se empleó una Raspberry Pi 4B con 8GB de RAM para ejecutar estas tareas de manera simultánea; sin embargo, se evidenció un retardo considerable en el procesamiento, lo cual resultaba inadecuado para las exigencias de navegación del robot omnidireccional de tres ruedas, que requiere una respuesta rápida. Ante esta limitación, se optó por reutilizar un equipo computacional previamente disponible que estaba fuera de operación debido a su antigüedad. Se obtuvo la placa y batería del mismo para reducir espacio y menor masa de carga para el robot. Cuenta con una placa de procesador Intel Core i7-4710MQ de cuarta generación y 8GB de RAM DDR3 junto al sistema operativo *Windows* 10 sobre un disco sólido de 1000GB. Este dispositivo demostró ser capaz de soportar la carga computacional requerida sin inconvenientes, funcionando eficientemente como módulo servidor incrustado dentro del robot, permitiendo así una operación fluida tanto para el stream de Datos 3D gracias a su conexión con la cámara Intel RealSense a través del puerto 3.0 disponible en la placa, obteniendo el máximo provecho de las características dadas por el fabricante y también para el control en tiempo real del robot con los controladores *RoboClaw* a 115200 baudios. Para la visualización del stream 2D y 3D se utiliza una pantalla tactil UCTRONICS UC-595 de resolución 1920x1080 conectada al puerto HDMI de la placa.

La computadora descrita previamente y presentada en la Figura 3.5, se utiliza como *(CPU)* en el robot omnidireccional de tres ruedas, permitiendo el funcionamiento en tiempo real del stream para que el cliente lea la información y ejecute los algoritmos de la cámara Intel RealSense D435i para el mapeo 3D del entorno, así como el control preciso de movimiento mediante comunicación serial con el controlador *RoboClaw* 2x15A. Esta capacidad de procesamiento local garantiza una operación autónoma eficiente sin depender de sistemas externos.

Figura 3.5: Computadora del robot

Controlador *RoboClaw* 2x15A

El controlador *RoboClaw* 2x15A presentado en la Figura 3.6, fue empleado para la gestión de los motores del robot móvil. Este controlador es ampliamente utilizado en aplicaciones de robótica gracias a su capacidad para manejar dos motores de corriente continua con retroalimentación, ofreciendo control preciso de posición, velocidad y corriente. Entre sus características se encuentra su compatibilidad con múltiples interfaces de comunicación, como UART, USB, I2C y PWM, facilitando su integración con microcontroladores y plataformas de desarrollo. Además, incorpora controladores PID configurables, lo que permite una regulación eficiente del movimiento. Su capacidad de hasta 15 A por canal lo hace adecuado para robots de tamaño medio, proporcionando un rendimiento confiable en tareas de navegación y movilidad. Gracias a estas características, el *RoboClaw* 2x15A representó una solución robusta y accesible para la implementación del sistema de control del robot desarrollado en este proyecto.

Cabe resaltar que, el *RoboClaw* 2X15A es un controlador de motores DC diseñado por *Basicmicro*(anteriormente *Ion Motion Control*), capaz de controlar dos motores de

corriente continua (DC) por canal, por esta razón se utilizaron dos unidades para tres motores. Una de sus principales ventajas es el soporte para *encoders* de cuadratura, lo que permite implementar lazos cerrados de control para una mayor precisión en el movimiento. Además, el *RoboClaw* puede funcionar en varios modos de operación: Control por señal PWM, control serial TTL o USB, ideal para comunicarse con microcontroladores o computadoras, control RC (Radio Control) y modo analógico. Otra característica destacable es que el *RoboClaw* 2x15A incluye protección contra sobrecorriente, sobrecalentamiento y bajo voltaje, lo cual lo hace robusto para entornos exigentes. El controlador puede manejar voltajes desde 6 V hasta 34 V, lo que lo hace compatible con diversas fuentes de energía. También incluye un regulador interno BEC que puede alimentar dispositivos externos a 5V, como sensores o microcontroladores.

Figura 3.6: Controlador para motores, *RoboClaw* 2x15A

3.1.5. Actuadores

Motor Pololu-4752

El Pololu 4752 presentado en la Figura 3.7, es un motor de corriente continua (DC) con escobillas, diseñado para operar a 12 V. Incorpora una caja de engranajes metálica con una relación de reducción de 30:1, lo que le permite alcanzar una velocidad sin carga de aproximadamente 330 revoluciones por minuto (RPM) y un torque de hasta 1.37 newton-metros (Nm). Sus dimensiones son: 67.6 milímetros de largo y 34 milímetros de diámetro. Además, este motor cuenta con un *encoder* de cuadratura integrado que proporciona una resolución de 64 cuentas por revolución (CPR) del eje del motor, lo que se traduce en 1920 CPR en el eje de salida de la caja de engranajes. Este *encoder* facilita un control preciso de la velocidad y posición del motor. El eje de salida tiene una longitud de 16 mm y un diámetro de 6 mm con forma de "D", lo que facilita su acoplamiento a diversos

dispositivos mecánicos. El motor presenta un consumo de corriente sin carga de 200 mA y puede llegar hasta 5.5A en condiciones de bloqueo.

Figura 3.7: Motor Pololu 4752 con *encoder* de cuadratura

3.1.6. Transmisión de información

Enrutador TP-Link Archer C1200

El enrutador TP-Link Archer C1200 presentado en la Figura 3.8, es un dispositivo de doble banda que ofrece velocidades combinadas de hasta 1200 Mbps, distribuidas en 300 Mbps para la banda de 2.4 GHz y 867 Mbps para la de 5 GHz, lo que permite una conexión estable tanto para navegación básica como para tareas más exigentes como streaming en alta definición. Cuenta con tres antenas externas para una mejor cobertura inalámbrica, cuatro puertos LAN para conexiones por cable de alta velocidad.

Figura 3.8: Enrutador TP-Link Archer C1200

Concentrador USB

El concentrador USB (también llamado *Hub USB*) presentado en la Figura 3.9, sirve para ampliar la cantidad de puertos USB disponibles en la computadora del robot dada su limitación de puertos. Es especialmente útil porque permite conectar los siguientes dispositivos: *RoboClaw*2x15A x 2, la pantalla táctil y el Arduino UNO.

Figura 3.9: Concentrador USB

3.1.7. Despliegue de información

Pantalla tactíl UCTRONICS UC-595

La UCTRONICS UC-595 presentada en la Figura 3.10, es una pantalla táctil LCD IPS capacitiva de 7 pulgadas, diseñada para integrarse con otros dispositivos. Ofrece una resolución de hasta 1920×1080 píxeles con auto-escalado, también tiene capacidad multi-táctil de 5 puntos y el controlador táctil capacitivo *plug-and-play* a través de USB facilitan una interacción fluida y sencilla.

Por otro lado, la pantalla es esencial para visualizar los datos 2D y 3D de la cámara Intel Realsense D435i en tiempo real y manipular con el táctil el sistema operativo *Windows* sin requerir un ratón (*mouse*). Su diseño portátil y el soporte de acrílico incluido permiten una instalación flexible en la estructura del robot, facilitando el monitoreo y control de las operaciones con el sistema operativo *Windows* 10. La alimentación se realiza a través del puerto USB, eliminando la necesidad de una fuente de alimentación adicional.

Figura 3.10: Pantalla Táctil UCTRONICS UC-595

3.1.8. Estructura mecánica

Pinza

El robot cuenta con una pinza metálica presentada en la Figura 3.11, operada por un servomotor a 5V, la cual permite agarrar y soltar paquetes de forma controlada por pulsos PWM con Arduino UNO, facilitando su traslado desde un punto A hacia un punto B.

Figura 3.11: Pinza controlada por servomotor

Ruedas suecas dobles a noventa grados

Las ruedas suecas dobles a noventa grados presentadas en la Figura 3.12, desempeñan un papel fundamental en la capacidad de desplazamiento del robot. Estas ruedas están fabricadas en aluminio, lo que les proporciona alta resistencia mecánica y durabilidad, ideales para soportar el peso del robot y las exigencias de la movilidad constante.

Cada rueda está compuesta por dos capas y cada capa incorpora 5 rodillos de caucho dispuestos de forma perpendicular de 90° al eje del motor y 31 mm de radio. Estos rodillos permiten que la rueda no solo gire sobre su eje principal, sino que también se deslice lateralmente con facilidad. Este principio de funcionamiento es lo que permite al robot realizar movimientos en cualquier dirección (adelante, atrás, lateral o diagonal), e incluso girar sobre su propio eje, sin necesidad de cambiar la orientación de las ruedas.

Figura 3.12: Ruedas suecas de noventa grados

Chasis

Dada la disponibilidad de materiales como MDF y acrílico presentados en la Figura 3.13, de 5 mm en el *SENA* de Popayán, se optó por utilizarlos en la construcción del chasis del robot. El MDF fue seleccionado para la capa inferior debido a su buena resistencia mecánica, mientras que el acrílico, utilizado en la capa superior, aporta una mejor presentación estética y permite una visualización clara de los componentes internos del sistema.

(a) MDF 5mm (b) Acrílico 5mm

Figura 3.13: Material MDF y acrílico para el chasis del robot

Tornillería

Para el ensamblaje de las piezas del robot se utilizaron tres tipos de tornillos: M3, M4 y M5, todos de rosca fina y con una longitud de 25 mm presentados en la Figura 3.14a. Estos tornillos cuentan con cabeza hexagonal, lo que facilita su manipulación con herramientas estándar y permite aplicar el par de apriete necesario sin dañar las piezas. Además, se emplearon tuercas de seguridad, también conocidas como tuercas con inserto de nylon presentados en la Figura 3.14b, las cuales previenen el aflojamiento por vibración, una característica esencial en sistemas móviles como robots. El uso de esta tornillería asegura una unión firme, duradera y fácilmente desmontable en caso de mantenimiento o modificaciones.

(a) Tornillos M3, M4 y M5 (b) Tuerca de seguridad

Figura 3.14: Tornillería empleada

3.1.9. Sistema de alimentación

Regulador de voltaje LM2596

Durante las pruebas, se observó que al operar el motor a altas velocidades, se generaban picos de voltaje que excedían los niveles tolerados del controlador *RoboClaw* 2X15A, provocando el apagado por protección. Para solucionar este inconveniente, se incorporó un regulador de voltaje LM2596 con led de siete segmentos presentado en la Figura 3.15, para apreciar el nivel de carga de la batería. Su uso estabilizó la alimentación del sistema, permitiendo un funcionamiento seguro y continuo del controlador bajo condiciones de alta demanda.

Figura 3.15: Regulador de voltaje LM2596

Baterias

Las baterías presentadas en la Figura 3.16, son de 12000 mAh a 12 V y proporcionan la alimentación que requieren los motores como también de los controladores *RoboClaw* de 5 V a través de los reguladores mencionados anteriormente. Se resalta el uso de dos baterías, una por cada *RoboClaw*.

Figura 3.16: Baterías 12 V a 12000 mAh

3.2. Métodos

En esta sección se detallan los procedimientos técnicos utilizados para la construcción e implementación del robot móvil omnidireccional. Se abordan aspectos como el diseño de la plataforma, la configuración del sistema de control y la integración de los componentes. Además, se describe el desarrollo del software y la arquitectura de comunicaciones entre el robot y el sistema remoto.

3.2.1. Control cinemático

El control cinemático en robots móviles se ocupa de establecer cómo los comandos de velocidad que se envían a cada actuador se traducen en la posición y orientación del robot. Este enfoque es clave para convertir esos comandos de velocidad en movimientos reales en el espacio, asegurando que el robot siga con precisión las trayectorias que se han planificado. Según el autor [1], el manejo cinemático se basa en fórmulas matemáticas que relacionan las velocidades de las ruedas con la velocidad lineal y angular del robot. Se destacan en recuadros negros las contribuciones específicas de cada llanta a las velocidades lineales y angulares del sistema, las cuales se integran en el modelo cinemático general del robot, representado en el recuadro verde de la Figura 3.17. Los términos trigonométricos dependen de la orientación del robot y su relación con las ruedas. En esta configuración, las ruedas están dispuestas con 120° de separación, formando un triángulo equilátero.

Terminología del modelo cinemático

Todos los términos mencionados se evidencian sobre el modelo cinemático del robot móvil omnidireccional de tres ruedas de la Figura 3.17.

1. $\frac{\pi}{6}$: Ángulos de orientación de las ruedas con respecto a los vectores unitarios que indican la dirección de rodadura de cada rueda. Como las ruedas están colocadas a ángulos de 120° una respecto de otra, comenzando desde el eje X_B se genera un ángulo $\phi = \frac{\pi}{6}$.

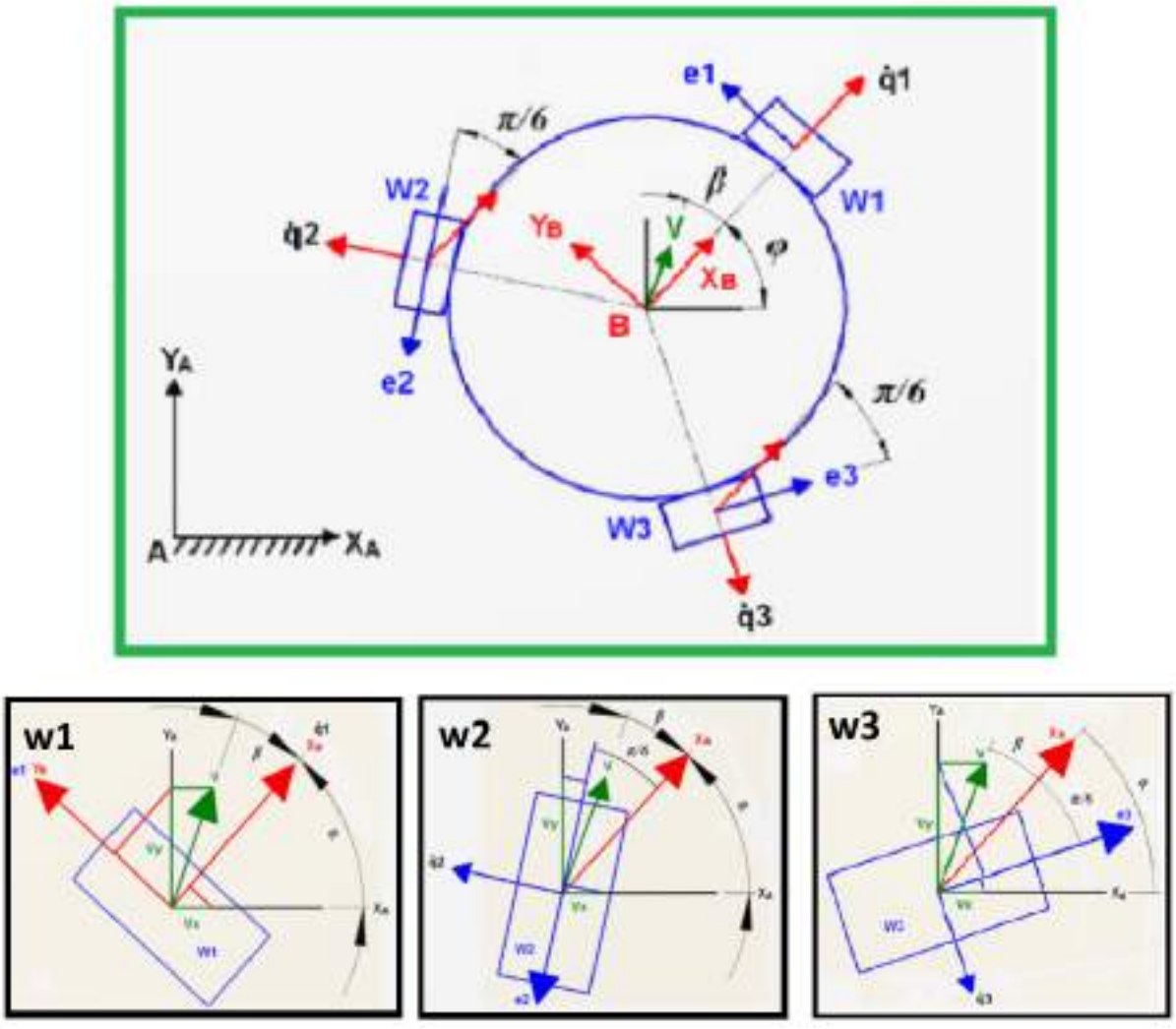


Figura 3.17: Modelo cinemático de un robot móvil omnidireccional de tres ruedas, tomado de [1]

2. X_A, Y_A: Sistema de coordenadas global o del mundo. Es un marco de referencia fijo respecto al entorno, usado para describir la posición y orientación absoluta del robot. Las trayectorias o metas suelen estar definidas en este sistema.

3. X_B, Y_B: Sistema de coordenadas local o del robot. Está centrado en el punto medio del robot (punto B) y se mueve junto con él.

4. $\dot{q}_1$, $\dot{q}_2$, $\dot{q}_3$: Son las velocidades angulares de las tres ruedas del robot W1, W2 y W3 respectivamente. Son medidas en radianes por segundo que representan la velocidad de giro individual generado mediante sensores, en este caso, los *encoders* de cuadratura.

5. $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$: Son los vectores unitarios que indican la dirección de rodadura de cada rueda. Es decir, representan la dirección en la que cada rueda genera velocidad lineal sobre el suelo cuando gira.

6. R: Es la constante del radio de las ruedas del robot. Se usa para convertir la velocidad angular de las ruedas en velocidad lineal en el punto de contacto con el suelo.

7. L: Es otra constante e indica la distancia desde el centro del robot hasta el punto de contacto de cada rueda con el suelo.

8. V_x, V_y: Son los componentes de la velocidad lineal del robot en el marco de referencia global (X_A, Y_A).

9. ω: Es el componente de velocidad angular del robot y representa la derivada de la orientación con respecto al tiempo, es decir, la rapidez de giro sobre su propio eje.

10. φ: Ángulo de orientación del robot con respecto al marco global. Define la rotación del marco local del robot (X_B, Y_B) respecto al marco global (X_A, Y_A).

Modelo cinemático inverso

El modelo cinemático inverso, es una expresión matemática que utiliza la matriz jacobiana J para convertir la velocidad lineal (expresada en el sistema de coordenadas global y la velocidad angular del robot) a las velocidades angulares que se necesitan en cada una de las ruedas para poder realizar una trayectoria determinada. Con ella, para valores de velocidad globales como consignas *Vx, Vy,* ω la matriz proporcionará las velocidades angulares requeridas en cada motor que permitan al robot moverse desde un punto inicial conocido hasta un punto final deseado.

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} -\sin(\varphi) & \cos(\varphi) & L \\ -\cos\left(\frac{\pi}{6}+\varphi\right) & -\sin\left(\frac{\pi}{6}+\varphi\right) & L \\ \cos\left(\varphi-\frac{\pi}{6}\right) & \sin\left(\varphi-\frac{\pi}{6}\right) & L \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} \tag{3.1}$$

Modelo cinemático directo

La matriz jacobiana inversa J^{-1} permite obtener la velocidad del robot en el sistema de coordenadas global y la velocidad angular del robot en función de las velocidades angulares de cada una de las ruedas. Esto implica que, al integrar con respecto al tiempo las velocidades de sistema de coordenadas global, se obtiene la posición X, Y y orientación φ del robot, permitiendo conocer su ubicación dentro del espacio. Cabe destacar que para lograr lo mencionado con anterioridad, el modelo cinemático directo requiere la lectura de las velocidades angulares $\dot{q}_1$, $\dot{q}_2$, $\dot{q}_3$ de las ruedas.

$$\begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} = \frac{R}{3} \begin{bmatrix} -2\sin(\varphi) & \sin(\varphi) - \sqrt{3}\cos(\varphi) & \sin(\varphi) + \sqrt{3}\cos(\varphi) \\ 2\cos(\varphi) & -\sqrt{3}\sin(\varphi) - \cos(\varphi) & \sqrt{3}\sin(\varphi) - \cos(\varphi) \\ \frac{1}{L} & \frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \tag{3.2}$$

3.2.2. Diseño del robot móvil omnidireccional

SolidWorks es un software de diseño asistido por computadora (*CAD*) presentado en la Figura 3.18 fue desarrollado por la empresa *Dassault Systèmes* y ampliamente utilizado en ingeniería mecánica para modelado 3D, simulaciones y generación de planos. Para el diseño del robot de tres ruedas se utilizó la versión de prueba 2020 para escritorio, la cual proporciona herramientas completas para el modelado de piezas y ensambles, permitiendo crear modelos con alto nivel de detalle y precisión. Durante el proceso de diseño, se modelaron las estructuras del robot, considerando la integración del chasis y motores.

Figura 3.18: Software CAD empleado, SolidWorks

Diseño

La Figura 3.19 muestra una vista inferior del diseño de la base del robot móvil omnidireccional de tres ruedas, elaborado en *SolidWorks*. Las líneas rojas representan el radio del robot L, definido como la distancia desde el centro geométrico del chasis que corresponde al punto de control del robot hasta el punto de contacto de cada rueda con el suelo. Este radio tiene una longitud de 16 cm, lo que implica un diámetro total de 32 cm. Esta medida es fundamental para los cálculos cinemáticos del robot, ya que determinan la orientación y la velocidad lineal del mismo en función cada rueda, como se evidencia en la Figura de

la ecuación 3.1.

Por otro lado, las líneas azules subrayan un aspecto clave del diseño: la disposición equidistante de las ruedas, separadas 120° entre sí. Esta configuración es característica de los robots omnidireccionales de tres ruedas y fue lograda con precisión gracias al modelado en *SolidWorks*, permitiendo asegurar una dinámica simétrica y estable. Este desfase angular de 120° es crucial para el correcto funcionamiento del modelo cinemático del robot, ya que permite que cualquier vector de velocidad del centro del robot pueda ser descompuesto adecuadamente en contribuciones de cada rueda.

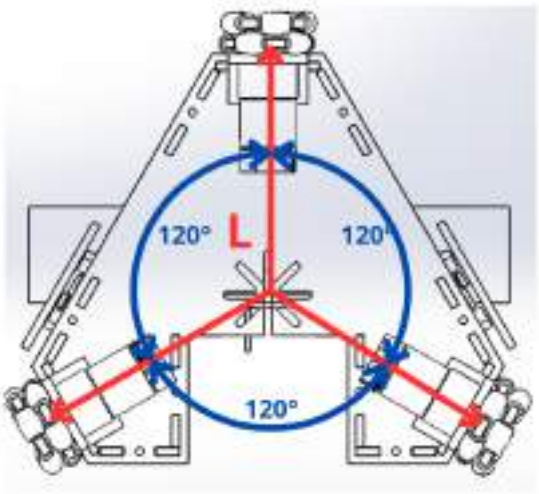


Figura 3.19: Vista inferior del diseño de la base del robot con distribución triangular.

Ensamble

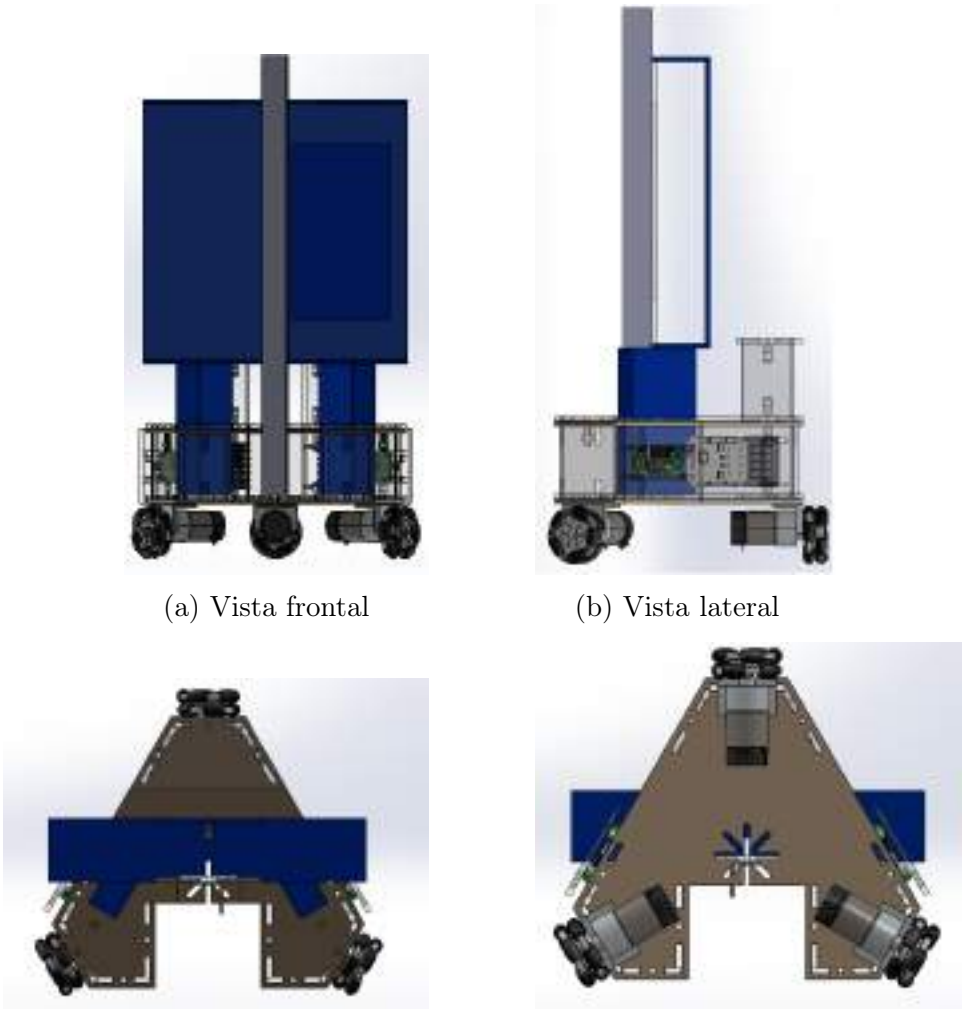
La fabricación del chasis del robot se llevó a cabo utilizando materiales de 5 mm de espesor: MDF para la capa inferior y acrílico para la capa superior. Debido a la precisión requerida en el corte de las piezas y la necesidad de ensamblaje exacto, se utilizó una cortadora láser de alta precisión. Este proceso de corte se realizó en las instalaciones del *Tecnoparque* perteneciente al *SENA*, ubicado en la ciudad de Popayán. El uso de esta tecnología permitió obtener cortes limpios, precisos y reproducibles, garantizando que las piezas diseñadas en *SolidWorks* encajaran perfectamente durante el proceso de ensamble físico.

Antes de realizar ensamble del robot también se llevó una simulación dentro de la plataforma de *SolidWorks*, donde se realizó una vista previa detallada de la unión de todas las piezas y electrónica, asegurando la compatibilidad entre los componentes antes de la fabricación física. El chasis del robot se compone de dos capas: una inferior fabricada en MDF de 5 mm mostrada en la Figura 3.13a para la resistencia mecánica de la estructura y una superior en acrílico como se evidenció en la Figura 3.13b con el mismo espesor para un acabado estético y refuerzo adicional. Además, se incorpora una capa intermedia también

en acrílico, cuya finalidad es unir la tapa superior e inferior y permitir la visualización de la electrónica instalada en el interior del robot.

Las capas fueron unidas mediante nervios estructurales y fijadas con tornillos de rosca fina M3, M4 y M5 con cabeza hexagonal, junto con tuercas de fuerza que aseguran una sujeción robusta como se evidenció en la Figura 3.14a y Figura 3.14b respectivamente.

Cabe destacar que la estructura del robot fue diseñada con un enfoque modular, utilizando piezas con configuraciones similares que se encajan entre sí de forma precisa. Esta modularidad permite una fácil sustitución de componentes, adaptación a nuevos diseños y escalabilidad del sistema. Dicho enfoque puede observarse claramente en la Figura 3.20 correspondiente al ensamble completo del robot dentro del entorno de *SolidWorks*.



(a) Vista frontal (b) Vista lateral

(c) Vista superior (d) Vista inferior

Figura 3.20: Vistas de ensamble del robot.

3.2.3. Procesamiento de la información

Utilizando el lenguaje de programación *Python*, se desarrollaron métodos para el procesamiento de imágenes RGB, de profundidad (*Depth*) y parámetros adicionales asociados al sistema de visión para la posterior reconstrucción con *Visual SLAM*.

Procesamiento de los parámetros

Archivos como los evidenciados en la Figura 3.29 son enviados y recibidos por el cliente y el servidor, permitiendo la comunicación entre los ejecutables. Para reconocer cuál parámetro o información se requiere leer previamente se ha guardado un archivo de texto en la ruta *MEDIA/dependencies/others* sobre una línea específica y con una llave única para identificarlo, también para lograr unir múltiples valores en una misma línea se han delimitado con dos puntos para el identificador y con comas para los valores *identificador:val1,val2,val3*. A continuación se muestra un ejemplo de archivo de parámetros presentado en la Figura 3.21:

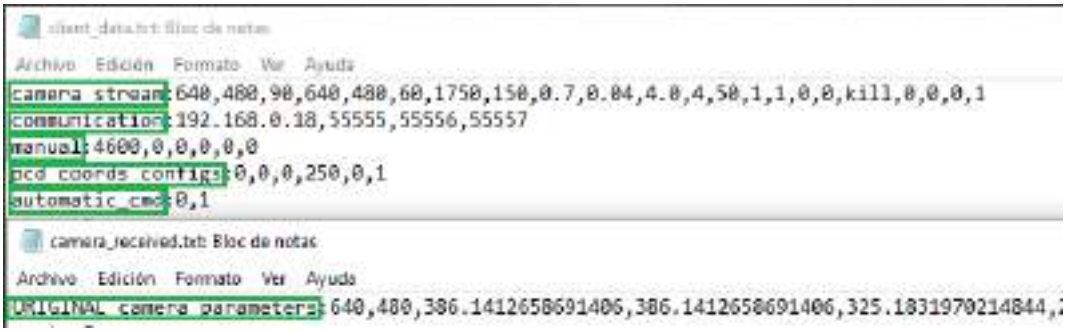


Figura 3.21: Archivo de texto con parámetros e identificador único

Procesamiento de imágenes RGB-D

Durante el manejo en modo manual del robot se capturan imágenes desde el servidor con extensión *JPG* que contienen información sobre la capa *RGB* y con extensión *PNG* para la capa de profundidad *D* y se envian por sockets codificadas a el cliente el cual las recibe y decodifica en el mismo orden.

3.2.4. Creación de las nubes puntos

Por cada captura de color y profundidad es necesario hacer el emparejamiento de las mismas para generar una captura en 3D en cada instante de tiempo. Para lograr lo anterior se requiere la matriz de valores intrínsecos, la cual es una representación matricial de los parámetros intrínsecos de la cámara, que describen su geometría interna y características ópticas. Estos parámetros incluyen la distancia focal, el punto principal y los factores de

escala en las direcciones horizontal y vertical.

En el contexto de la cámara RGBD, que combina información de color (RGB) y profundidad (D), esta matriz también tiene en cuenta las diferencias en la geometría de la profundidad. Estos parámetros son necesarios para convertir las coordenadas de píxeles de una imagen en coordenadas tridimensionales en el espacio del mundo real. La matriz intrínseca de una cámara RGBD típicamente es representada por cuatro valores, donde:

Fx y **Fy**: son las distancias focales expresadas en píxeles en las direcciones horizontal (x) y vertical (y) respectivamente. Estas están relacionadas con la distancia focal real de la lente y el tamaño de los píxeles del sensor.

Cx y **Cy**: representan las coordenadas del punto principal también denominado centro óptico de imagen, es decir, el punto donde el eje óptico de la cámara intersecta el plano de la imagen. Normalmente, este valor está cerca del centro de la imagen, aunque puede variar ligeramente.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Se ha utilizado la librería propia de la cámara en Python *pyrealsense2* que proporciona dicha información.

$$K = \begin{bmatrix} 386.141 & 0 & 325.183 \\ 0 & 386.141 & 235.666 \\ 0 & 0 & 1.000 \end{bmatrix} \tag{3.4}$$

Los valores de la matriz anterior se leen desde cámara en el servidor donde se guardan en un archivo de texto y posteriormente se envían por *sockets* hacia el cliente el cual los recibe junto al par de imágenes *RGB-D* y los transfiere a una función de la librería *Open3D* que construye la nube de puntos y las guarda en cada instante de tiempo.

3.2.5. *Visual SLAM* (*PoseGraphs* con librería *Open3D)*

Para la creación del mapa donde el robot opera, desde la pestaña de la interfaz mostrada en la Figura 3.32 se selecciona la cantidad de imágenes a tomar y la velocidad de muestreo, las mismas se almacenan sobre la carpeta *Client..MEDIA..saved..pointcloudsframes* hasta alcanzar un tope máximo de *250 Imágenes 3D* para ser importadas a continuación por los algoritmos de *Visual SLAM* de manera offline debido a la exigencia computacional del algoritmo y la cantidad información por captura que posee una nube de puntos. El funcionamiento de *Visual SLAM* se describe de manera detallada con los siguientes pasos:

- **Creación de la clase *KeyFrame*:** La captura 3D en un instante de tiempo *t(x)* es denominada *KeyFrame* . El mapeo 3D de un entorno puede contener una lista de cantidad desde *2 hasta (n) KeyFrames*. Para para realizar SLAM, cada *KeyFrame*

es un objeto *Python* como se muestra en el Código 5.1 que contiene las siguientes características: Un *cloud* es una captura de profundidad 3D, *fpfh (Fast Point Feature Histogram)* codifica cómo está dispuesta la captura 3D (en términos de orientación, distancia y forma), lo cual ayuda a comparar regiones con nubes diferentes, *odometry* para estimar la posición y orientación (pose) del robot con la cámara a traves del producto punto entre la sucesión de capturas a lo largo del tiempo y un *Node* que contiene la Odometria en un instante de tiempo *Od(t)* con la pose (posición + orientación) del robot.

- **Creación de la clase *SLAM***: SLAM es un objeto *Python* como se evidencia en el Código 5.2 que contiene un *PoseGraph* o grafico de poses con múltiples *Nodes* y *Edges* que son las transformaciones relativas entre poses (por ejemplo, cuánto se movió de un nodo a otro) permitiendo hacer el emparejamiento entre capturas. También incluye una lista de *KeyFrames* que guarda las capturas de nubes de puntos de acuerdo a lo explicado anteriormente. A continuación se ilustra el código que crea la clase *SLAM* en *Python* utilizando la librería *Open3D*.

- **Creación de la función de optimización**: El *backend* utilizado y las funciones necesarias para realizar *SLAM* esta basado en gráfico de poses con *Nodes*,*Edges* y *Global Optimization* con la la minimización del error de mínimos cuadrados no lineales con *Levenberg & Marquardt* como se muestra en el Código 5.3. La función de optimización de *Levenberg–Marquardt* se utiliza en sistemas de *Visual SLAM* para refinar de manera iterativa la estimación de la trayectoria de la cámara y la reconstrucción correcta del entorno. Para ello se ajustan simultáneamente las posiciones de la cámara (*poses*) y las ubicaciones de los puntos 3D del mapa minimizando el *error de reproyección* acumulado, es decir, la diferencia entre las posiciones proyectadas de los puntos 3D en las imágenes y las posiciones donde realmente fueron observados. El método de *Levenberg–Marquardt* implícito en *Open3D* combina las ventajas del *descenso del gradiente* y del método de *Gauss–Newton*.

- **Cálculo de las transformaciones**: El cálculo de las transformaciones se realiza con el algoritmo implementado por el sistema de *Visual SLAM* tal como se evidencia en el Código 5.4. Este algoritmo tiene como objetivo emparejar (reconstruir) de manera iterativa el mapa 3D utilizando pares nubes de puntos. Se detallan los pasos clave del proceso, que incluyen la inicialización, el registro de nuevas nubes de puntos, la actualización de *keyframes*, el cierre de bucles para corregir errores acumulados, y la optimización del grafo de poses para mejorar la precisión global del sistema.

  1. **Inicialización y procesamiento de la nube de puntos:**
     a) La primera operación es el *voxel downsampling*, que reduce la resolución de la nube de puntos *cloud* mediante un tamaño de *voxel* especificado (*voxel size*). Este paso ayuda a reducir la cantidad de datos y mejora el rendimiento sin perder demasiada precisión.

b) A continuación, se calculan las características *FPFH* (*Fast Point Feature Histograms*) de la nube de puntos. Estas características son útiles para la correspondencia de nubes de puntos durante el proceso de registro, ya que capturan la geometría local de la nube de puntos.

2. **Primer keyframe:**

   a) Si aún no existen keyframes, el algoritmo asume que es la primera vez que se ejecuta y se establece la odometría inicial como la matriz de identidad (*identity matrix*). Este paso asigna la posición y orientación inicial del robot.

   b) Un nuevo *keyframe* es creado y almacenado en la lista *keyframes*. Este keyframe contiene la nube de puntos procesada (*cloud*), las características *FPFH* (*fpfh*) y la odometría inicial (*odom*).

   c) Se crea una visualización de la cámara en el espacio 3D, utilizando la función *LineSet.create_camera_visualization* de la librería Open3D, que permite ver la posición y orientación de la cámara.

3. **Actualización de keyframes:**

   a) Si ya existen keyframes, el algoritmo intenta registrar la nube de puntos nueva con la última nube de puntos almacenada en el keyframe anterior. Esto se realiza mediante el método *register_point_cloud_fpfh*, que alinea las nubes de puntos utilizando las características *FPFH*.

   b) Si el registro es exitoso, se calcula la nueva odometría como la multiplicación de la odometría anterior con la transformación obtenida (*np.dot*).

   c) Se agrega un nuevo nodo al grafo de poses, que es representado por el objeto *PoseGraphEdge*. Este grafo mantiene la relación entre las odometrías de los keyframes consecutivos.

4. **Cierre de bucles:**

   a) Si hay más de 5 keyframes, el algoritmo realiza un chequeo para verificar si las posiciones de los keyframes son muy cercanas entre sí (menos de 1 metro de traslación y menos de 10 grados de rotación). Si se cumple esta condición, se intenta realizar un registro de la nube de puntos con el keyframe más antiguo.

   b) Si el registro es exitoso, se agrega un nuevo *PoseGraphEdge* con la transformación correspondiente y se marca como *uncertain*, indicando que el registro tiene un grado de incertidumbre que se debe tener en cuenta.

5. **Optimización del grafo de poses:**

   a) Después de actualizar los keyframes, el algoritmo realiza una optimización del grafo de poses (*optimize_posegraph*) para ajustar las odometrías y las relaciones entre keyframes. Esta optimización minimiza los errores globales de la estimación de la trayectoria y mejora la precisión en la reconstrucción del mapa 3D.

- **Aplicación de las transformaciones**: La aplicación de las transformaciones se evidencian a través del Código 5.5, el cual aplica las transformaciones previamente calculadas por el algoritmo de *Visual SLAM* a una secuencia de nubes de puntos para construir el mapa 3D final. El procedimiento se detalla en los siguientes pasos:

  1. **Carga de nubes de puntos:** Se recorren los índices de los archivos PLY almacenados y se cargan las nubes de puntos utilizando la función *o3d.io.read_point_cloud*. Estas se almacenan en la lista *frames_to_recons*.
  2. **Inicialización del sistema SLAM:** Se crea una instancia de la clase *SLAM*, la cual contiene los métodos de procesamiento de cada nube de puntos, registro y construcción del grafo de poses. El parámetro *color_mode* se establece en *True* para usar información de color.
  3. **Preprocesamiento de las nubes de puntos:** Para cada nube:
     - Se estiman las normales con *estimate_normals* usando una búsqueda de vecinos más cercanos basada en *KDTree*.
     - Se reduce la nube con *farthest_point_down_sample*, lo que conserva puntos distantes para mantener la estructura.
     - Se elimina el ruido con *remove_statistical_outlier*, filtrando puntos que con una distancia mayor a 85 *cm* y que no se encuentra cerca de por lo menos 80 puntos vecinos.
  4. **Actualización del SLAM:** Cada nube preprocesada es pasada al método *update* del objeto *slam*, que actualiza el grafo de poses y agrega nuevos *keyframes* si es necesario. Se mide el tiempo de procesamiento para cada iteración y se calcula la frecuencia de actualización en *FPS*.
  5. **Reconstrucción del mapa 3D:**
     - Se inicializa el visualizador 3D con *Visualizer*, configurando opciones como el color de fondo y visibilidad del marco de coordenadas.
     - Se recorre cada *keyframe* del sistema *SLAM*, y se aplica la transformación calculada (*pose*) sobre su nube de puntos mediante el método *transform*.
     - Cada nube transformada se suma a una nube acumulada llamada *pcd_combined*, y se actualiza la visualización en tiempo real.
  6. **Postprocesamiento del mapa combinado:**
     - A la nube total *pcd_combined* se le estiman nuevamente las normales.
     - Se realiza una reducción adicional de puntos con *farthest_point_down_sample*.
     - Finalmente, se guarda el mapa completo en formato *PLY* mediante *o3d.io.write*.

Finalmente, la Figura 3.22 presenta el diagrama de flujo que organiza de manera secuencial las principales etapas del proceso de *Visual SLAM*, permitiendo visualizar la interacción entre las fases de algoritmo de manera mas general e intuitiva.

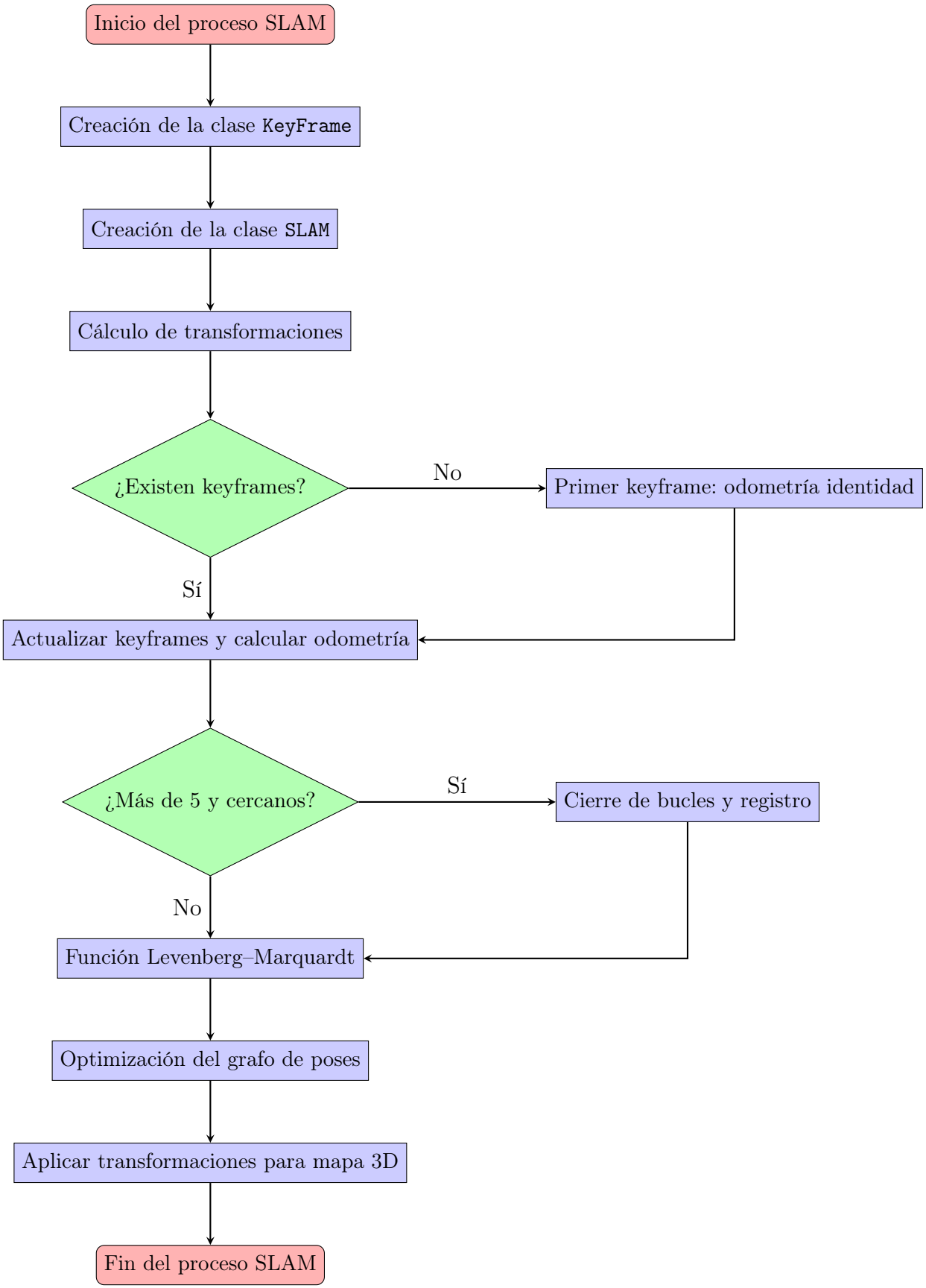


Figura 3.22: Diagrama de flujo del proceso de SLAM

3.2.6. Selección de trayectorias

La selección de trayectorias para el robot omnidireccional de tres ruedas se realiza de manera manual por parte del usuario, utilizando el mapa 3D generado a partir de la nube de puntos obtenida mediante *Visual SLAM*. Una vez construido este mapa, es el usuario quien define los puntos por los que debe transitar el robot. Es importante destacar que la validación de que dichos puntos estén realmente sobre el suelo también recae en el usuario, ya que el robot no cuenta con percepción suficiente para distinguir si un punto se encuentra en el suelo o en otra superficie; por lo tanto, se requiere del conocimiento previo del entorno por parte del operador. Esta intervención manual permite mayor flexibilidad en entornos complejos o cambiantes, donde una supervisión humana garantiza trayectorias más seguras y precisas. Además, dado que el sistema no interpreta semánticamente el entorno, es el usuario quien debe evitar trayectorias que incluyan obstáculos, desniveles o superficies elevadas. Por ello, el proceso de planificación depende directamente del criterio y experiencia del usuario que interactúa con el mapa 3D.

3.2.7. Generación de trayectorias

Para generar la trayectoria del robot a partir de los puntos seleccionados manualmente por el usuario, se utiliza la Ecuación 3.1, que corresponde al modelo cinemático inverso del robot. Esta ecuación permite calcular las velocidades angulares necesarias en cada una de las ruedas para que el robot se desplace de un punto a otro. Sin embargo, como dicho modelo requiere velocidades lineales y angulares en coordenadas globales, y solo se dispone de posiciones globales x, y, φ extraídas del mapa 3D reconstruido mediante *Visual SLAM*, es necesario estimar las velocidades que permitan conectar esos puntos. Para ello, se recurre al uso de polinomios de quinto orden, que permiten interpolar suavemente entre un punto de partida y uno de llegada, considerando condiciones iniciales y finales tanto de posición como se evidencia en la Ecuación 3.5, velocidad en la Ecuación 3.6 y aceleración en la Ecuación 3.7.

$$s(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \tag{3.5}$$

$$\dot{s}(t) = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E \tag{3.6}$$

$$\ddot{s}(t) = 20At^3 + 12Bt^2 + 6Ct + 2D \tag{3.7}$$

Los seis coeficientes del polinomio se determinan resolviendo un sistema de ecuaciones lineales de la forma $AX = B$, donde X es el vector de coeficientes, A es una matriz generada a partir de las condiciones en los extremos del intervalo de tiempo (tiempo de evaluación, posición inicial, posición final deseada, tiempo inicial igual a cero y tiempo final correspondiente al tiempo total de simulación), y B es el vector que contiene dichos valores. Aunque la función utilizada para generar estos polinomios también devuelve aceleraciones, solo se consideran las posiciones y velocidades resultantes, ya que el modelo cinemático no requiere información sobre la aceleración.

La matriz A se construye utilizando las condiciones iniciales y finales de posición y velocidad para definir un sistema de ecuaciones lineales con seis incógnitas. El vector B contiene los valores correspondientes a esas condiciones, y al resolver el sistema $AX = B$, se obtienen los coeficientes *A, B, C, D, E, F* del polinomio. En caso de que el sistema sea singular, se utiliza un enfoque de mínimos cuadrados con la función *np.linalg.lstsq*. Posteriormente, se evalúa el polinomio usando *np.polyval* para obtener la trayectoria (posición), su derivada (velocidad) y su segunda derivada (aceleración). Estas tres salidas son devueltas por la función en el orden: *s[0]*, *sd*, *sdd*, aunque solo se utilizan *s[0]* y *sd*, ya que el modelo cinemático no requiere aceleración.

La función mostrada en el Código 5.6 implementa el cálculo de los coeficientes del polinomio de quinto orden a partir de los parámetros de entrada *t (tiempo actual)*, *i_v (valor inicial)*, *f_v (valor final)*, *i_t (tiempo inicial)* y *f_t (tiempo final)*.

3.2.8. Escalización de las velocidades del robot

La matriz de cinemática inversa proporciona los valores necesarios de velocidad angular para cada motor. Sin embargo, los motores son controlados mediante señales comprendidas entre 0 y 12000 por el controlador *RoboClaw*2x15, utilizando modulación por ancho de pulso (PWM, *Pulse Width Modulation*). Las velocidades angulares calculadas por la cinemática inversa están limitadas por las especificaciones físicas del motor Pololu, que tiene un máximo de *34.5 rad/s* (equivalente a 330 RPM). Por lo tanto, es necesario escalar estos valores para adaptarlos al rango de control del *RoboClaw*.

Para lograr escalizar los valores calculados por la matriz inversa, el Código 5.7 implementa una función llamada *scale*, que permite convertir (escalar) las velocidades angulares calculadas por la cinemática inversa, expresadas en radianes por segundo (*rad/s*), a valores de *PWM* comprendidos entre 0 y 12000, los cuales son entendidos por el controlador *RoboClaw*2x15. Esta función realiza un mapeo lineal entre dos rangos: el de entrada, que va desde 0 hasta la velocidad máxima física del motor Pololu (34.5 *rad/s*), y el de salida, que va de 0 a 12000. Por ejemplo, si se solicita una velocidad de 17.25 *rad/s* (la mitad de la máxima), el resultado será un valor *PWM* de 6000. Esta conversión asegura que las señales enviadas al controlador sean proporcionales a la velocidad real deseada, respetando los límites físicos del motor. No obstante, como el código actual contempla velocidades negativas (necesarias para rotación en sentido contrario), se extiende automáticamente para manejar movimiento bidireccional, escalando valores negativos de *-max_motor_vel* a *max_motor_vel* hacia un rango *PWM* de *-12000* a *12000*.

Para describir de manera más detallada la función `scale` se presenta en el Código 5.7 que implementa una escalización lineal con los parámetros mostrados en la Ecuación 3.8 en donde:

- ***inp***: Valor de entrada a escalar. En este caso, una velocidad angular real, en radianes

por segundo, que puede ser positiva o negativa según la dirección del giro.

- ***in_ min***: Valor mínimo del rango de entrada. Para este caso, corresponde a 0 rad/s o -34.5 rad/s.

- ***in_ max***: Valor máximo del rango de entrada, igual a 34.5 rad/s.

- ***out_ min***: Valor mínimo del rango de salida del controlador, que representa la velocidad negativa máxima (0 o -12000).

- ***out_ max***: Valor máximo del rango de salida del controlador, asociado a la velocidad positiva máxima (12000).

La fórmula aplicada dentro de la función es:

$$\texttt{out} = \left(\frac{\texttt{inp} - \texttt{in\_min}}{\texttt{in\_max} - \texttt{in\_min}} \right) \cdot (\texttt{out\_max} - \texttt{out\_min}) + \texttt{out\_min} \tag{3.8}$$

Este cálculo preserva la proporcionalidad entre la entrada y la salida, escalando cualquier velocidad angular dentro del rango de entrada al rango del sistema digital del controlador para finalmente convertir el resultado en un entero usando la función `int()`, ya que el controlador espera una señal digital discreta, típicamente representada como un número entero.

3.2.9. Seguimiento de trayectorias

El Código 5.8 muestra cómo, a partir de las trayectorias generadas y utilizando el modelo cinemático inverso descrito en la Ecuación 3.1, se calculan las velocidades angulares requeridas para cada motor, las cuales son luego escaladas al rango aceptado por el controlador *RoboClaw*. Este proceso permite que el robot siga de forma continua las trayectorias deseadas, moviéndose entre los puntos de control definidos en el espacio, y asegurando que la ejecución respete tanto la dinámica del sistema como las limitaciones físicas de los actuadores. También, la Figura 3.23 representa el proceso de seguimiento de trayectorias a través de un diagrama de flujo.

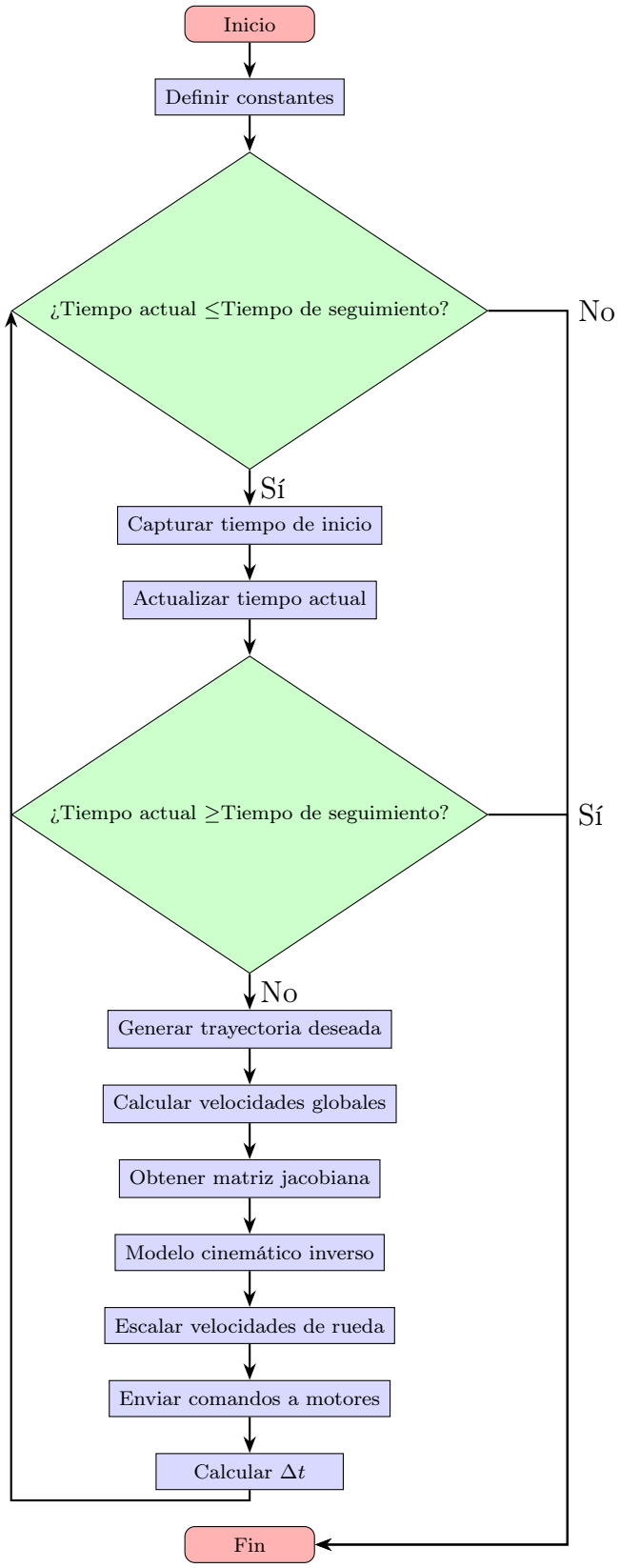


Figura 3.23: Diagrama de flujo del seguimiento de trayectorias del robot.

3.3. Descripción funcional del sistema

Es esencial contar con una descripción funcional del sistema, ya que esta proporciona una visión clara de cómo operan sus distintos componentes y cómo interactúan entre sí. Esta descripción permite identificar los objetivos principales del sistema, su arquitectura general y las funciones clave que desempeña. Al establecer este contexto, se facilita la comprensión del diseño, la toma de decisiones técnicas y la justificación de las metodologías empleadas

3.3.1. Descripción general

En la Figura 3.24 se muestra una representación visual del funcionamiento general del sistema.

1. **Robot (Servidor)**
   El robot actúa como servidor, es decir, es quien ofrece servicios tales como la transmisión de datos de profundidad, color y parámetros. Está equipado con una tarjeta de red inalámbrica que le permite conectarse al punto de acceso. A nivel de software, ejecuta un servidor basado en TCP/IP y sockets que queda a la espera de peticiones provenientes del cliente (la PC).

2. ***Access Point (Router)***
   El router funciona como punto de acceso inalámbrico, creando una red local (WLAN, *Wireless Local Area Network*) a la cual se conectan tanto el robot como la PC. Su principal función es enrutar los paquetes de datos entre los dispositivos de la red. Gestiona las direcciones IP y facilita la comunicación bidireccional entre el cliente y el servidor, funcionando como puente entre ambos.

3. **PC (Cliente)**
   La computadora funciona como cliente, estableciendo la conexión activa con el robot para enviar comandos y recibir datos. A través de la red proporcionada por el punto de acceso, la PC puede realizar solicitudes al robot. Las respuestas pueden incluir imágenes, estados del sistema u otra información relevante permitiendo el control remoto del robot.

3.3.2. Descripción detallada

En la Figura 3.25, se presenta un diagrama detallado que describe la arquitectura interna de la comunicación entre los componentes del sistema, en donde se destacan dos bloques principales: el recuadro de *Ejecutables (.exe)* y el recuadro de *Almacenamiento*. Los archivos ejecutables representan los programas compilados que controlan tanto el servidor (robot) como el cliente (PC), y que se encargan de gestionar la lógica de comunicación y operación del sistema. Estos ejecutables establecen una conexión directa con el recuadro de *MEDIA*, el cual actúa como repositorio central donde se almacena la información generada

Figura 3.24: Funcionamiento del sistema

o utilizada por ambos extremos. De esta forma, se establece una estructura organizada en la que los datos pueden ser compartidos, registrados o accedidos por las aplicaciones en ejecución. Esta interacción es común tanto en el lado del servidor como en el del cliente, garantizando un flujo de información constante.

A continuación se explica el propósito de cada ejecutable que se encuentra en el diagrama detallado del sistema. Estos archivos compilados representan módulos funcionales específicos que trabajan de forma conjunta para cumplir con las tareas asignadas, tanto en el servidor como en el cliente. Cada ejecutable se encarga de una función clave.

1. **Servidor**: El funcionamiento interno de los archivos ejecutables utilizados en el servidor del sistema son el ejecutable *Online Tracking* encargado de gestionar tanto el modo automático como el modo manual de los controladores del robot, específicamente mediante el controlador *Roboclaw 2x15A*. Por su parte, *Camera Server.exe* se encarga de establecer la comunicación y gestión de la cámara *Intel RealSense*, permitiendo la adquisición de imágenes y su posterior transmisión. El archivo *Microcontroller.exe* permite la conexión con el microcontrolador *Arduino UNO*, mediante el cual se gestionan los comandos para la pinza. Finalmente, *Robot Server.exe* es el ejecutable principal que arranca y mantiene el servicio del servidor, coordinando la comunicación entre todos los módulos mencionados y garantizando el funcionamiento continuo del sistema en el robot.

2. **Cliente**: Los archivos ejecutables que conforman el cliente del sistema son el ejecutable *Static visualizer.exe* que corresponde a una interfaz que permite visualizar las trayectorias o puntos que el robot seguirá durante su navegación. Complementariamente, *Dinamic visualizer.exe* es una interfaz dinámica desde la cual el usuario puede interactuar directamente para seleccionar puntos o coordenadas que el robot debe seguir, entre los cuales se encuentran el punto de partida, puntos intermedios y el punto de llegada. El archivo *SLAM subprocess.exe* es responsable de ejecutar el algoritmo de *SLAM* a partir de las imágenes almacenadas en la carpeta *MEDIA*, funcionando como un subproceso para evitar bloquear el programa principal debido a la alta carga computacional. El ejecutable *Frames Client.exe* se encarga de recibir

las imágenes de color y profundidad desde el robot, a través de los puertos 55555 y 55556 respectivamente. Por su parte, *my Client.exe* gestiona la recepción de parámetros como coordenadas y estados a través del puerto 55557. Finalmente, *Robot Client (GUI).exe* actúa como el programa principal del cliente, siendo responsable de lanzar y coordinar la ejecución de los demás procesos, además de desplegar una interfaz gráfica desde la cual se puede manejar remotamente el robot desde el PC.

- ***Configuración del punto de acceso*** Para esta práctica se ha utilizado el enrutador *TP-Link Archer C1200* presentado en la Figura 3.26. Antes de realizar cualquier fase es necesario tener un punto de acceso no necesariamente con conexión a internet, solo es necesaria la red para la comunicación y tener buen alcance con la misma, por ello se recomienda el uso de un enrutador. La necesidad anterior se debe al protocolo que ha sido seleccionado para la comunicación el cual es TCP/IP, dicho protocolo permite hacer el *stream* en paralelo mediante *sockets* de red en tres diferentes puertos por defecto. Para la comunicación de los datos obtenidos con la cámara *Intel RealSense* se utiliza el puerto 55555 para el *stream* de profundidad, 55556 para el *stream* de Color y finalmente el puerto 55557 para los parámetros.

- ***Activación del Servidor*** La fase inicial de la prueba consiste en activar el servicio de control y *streaming* mediante un ejecutable en la computadora con sistema operativo *Windows* puesta sobre el robot como se muestra en la Figura 3.27. Dicho programa no requiere de librerías y se ha compilado previamente para el sistema operativo de *Windows*. Una vez activado el servidor quedará a la escucha para la conexión de un solo cliente que tendrá el control del robot.

- ***Activación del Cliente*** La activación del cliente es realizada de igual forma sobre un ejecutable, pero desde otra computadora perteneciente al usuario, como se muestra en la Figura 3.27. Permite controlar mediante una interfaz gráfica como se muestra en la Figura 3.28 todos los parámetros del robot de manera remota, dicha interfaz consta de dos métodos de control, manual y automático. También posee cuatro pestañas de configuraciones, comunicación, cámara y control de *stream*, manual y automático. .

- ***Almacenamiento de datos*** En las Figuras 3.27 y 3.28, se evidencia la carpeta *MEDIA* la cual contiene la información para el respectivo servicio. El servidor almacena datos en archivos de texto con los parámetros de control requeridos para hacer la comunicación interna entre archivos y funcionar de manera adecuada. El cliente también almacena información de configuración dada por la interfaz de usuario, datos de envío hacia el servidor en formato de texto y datos de nubes de puntos en formato *.PLY* de capturas y reconstrucciones 3D generadas por el algoritmo de *Visual SLAM*. A continuación, en la Figura 3.29 se muestra como ejemplo las rutas de *MEDIA* para el cliente, que posee archivos de funcionamiento, parámetros y nubes de puntos.

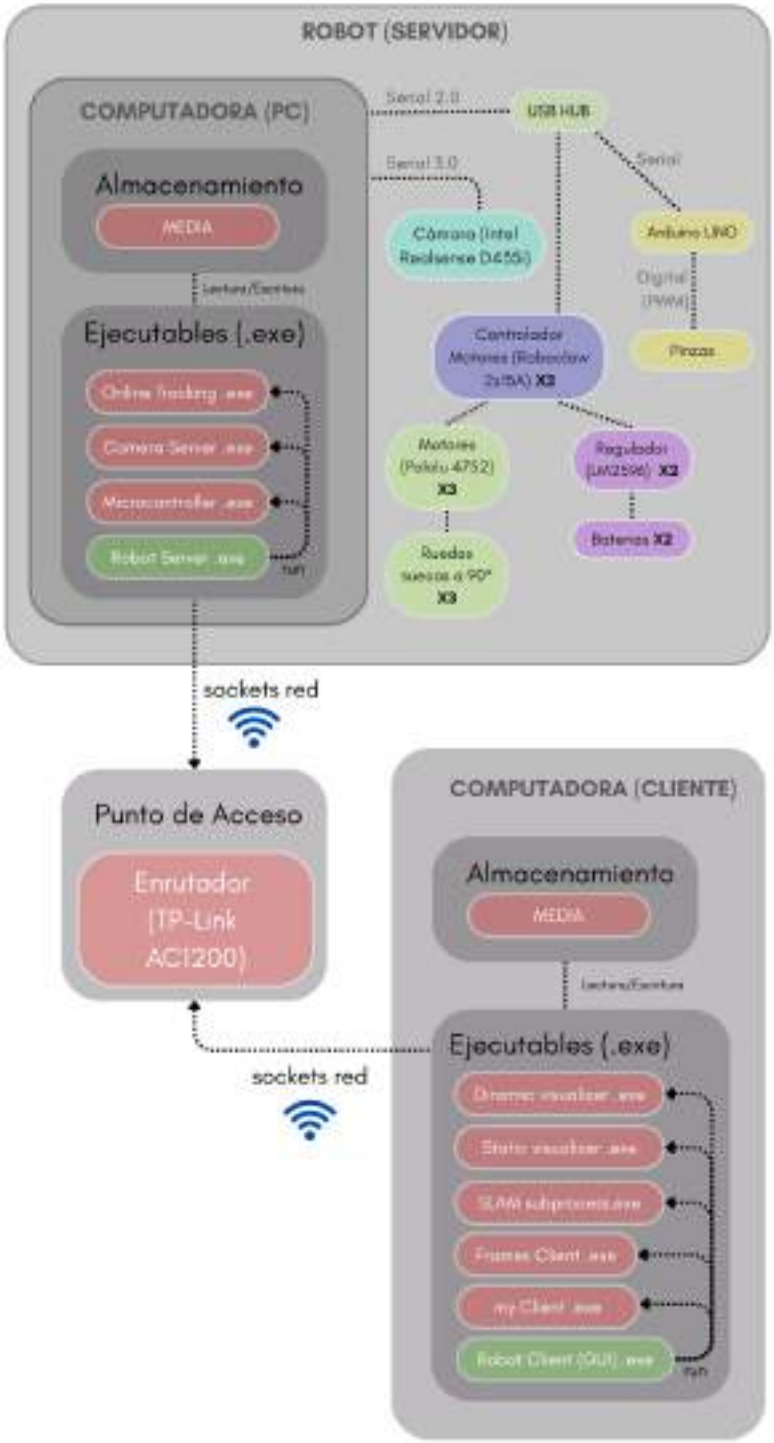


Figura 3.25: Arquitectura del sistema. El bloque Servidor y el bloque cliente conectados al punto de acceso.

Figura 3.26: Enrutador TP-Link Archer C1200

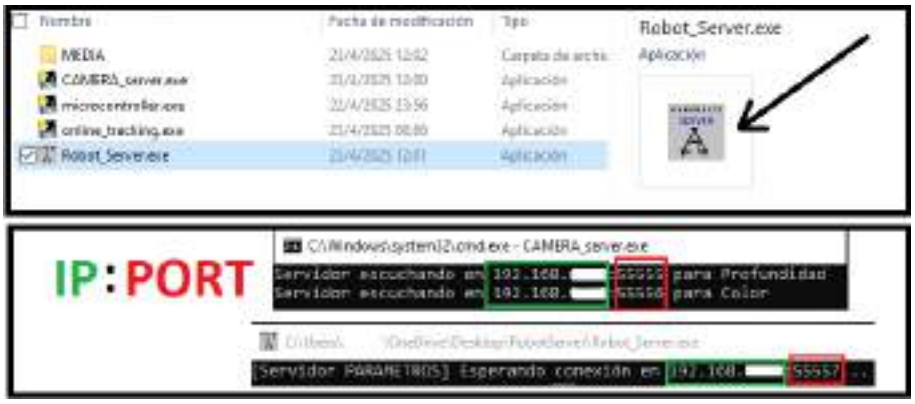


Figura 3.27: Ejecutable del Servidor

Figura 3.28: Ejecutable del Cliente

- ***Procesamiento de la información*** Los servicios procesan la información de manera independiente. Aunque la comunicación se hace de manera conjunta, es decir,

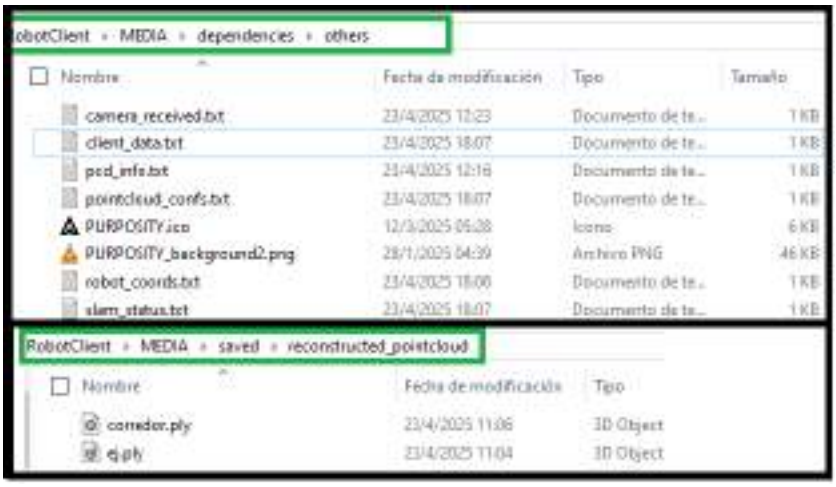


Figura 3.29: Rutas de almacenamiento de datos

el servidor recibe y envía información y el cliente también, cada uno la procesa por separado. El servidor procesa información de comunicación entre el microcontrolador Arduino, los controladores *RoboClaw*s y la cámara a través de los archivos de texto, los puertos seriales de la computadora y los sockets de la red. Por otro lado, el cliente procesa la información de manera similar pero a diferencia del servidor solo recibe y guarda información de imágenes de profundidad, parámetros de la cámara y posteriormente los procesa con scripts en paralelo para la reconstrucción 3D y la visualización de las mismas con librerías previamente compiladas.

- ***Selección de coordenadas para seguimiento*** Desde el cliente se envían puntos de control al servidor que contienen *x, y, z, orientación (phi)* usando la interfaz de la Figura 3.30 para selección de puntos, guardando la información en el archivo *robots coords.txt* evidenciado en la Figura 3.29 sobre la ruta *RobotClient..Media..dependencies...others* y posteriormente enviándolo por el puerto de comunicación de parámetros 55557.

En la Figura 3.30 se muestra un mapa de ejemplo con las dos interfaces gráficas. La interfaz *POINT SELECTOR* permite la selección de los puntos para el seguimiento y *COORDINATE CONFIGURATOR* la visualización:

3.4. Arquitectura del sistema

Se ha proporcionado el contexto de los componentes principales del sistema, ahora se procede a describir la arquitectura de su funcionamiento de manera detallada.

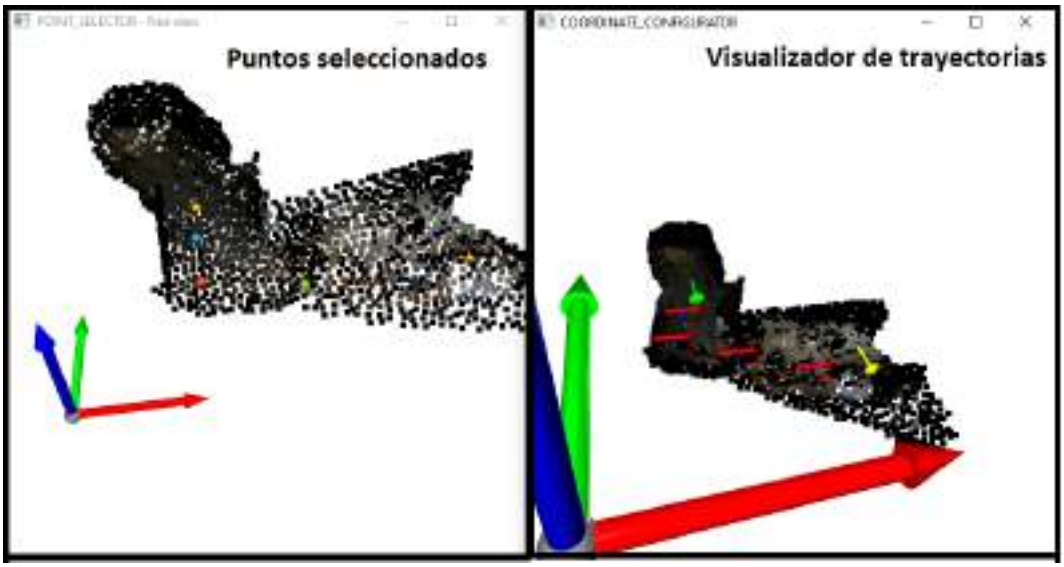


Figura 3.30: Interfaz para la selección de puntos de seguimiento del robot

3.4.1. Bloque módulo servidor

Este módulo es el robot y funciona como un servidor que ofrece características de *streaming* con la cámara, control manual y control automático con seguimiento de trayectorias seleccionadas por el usuario desde el cliente. También contiene los dispositivos físicos mencionados en el apartado anterior.

- **Comunicación:** El servidor se comunica inalámbricamente con el cliente a través de la computadora que tiene antena para ser conectada al enrutador. Los parámetros e imágenes enviados desde el robot y recibidos por el cliente se realizan a través de *sockets* programados en Python con el protocolo TCP/IP. Los puertos para ello son: 55555 para profundidad, 55556 para el color y 55557 para los parámetros, presentados en la Figura 3.27. Adicionalmente, los valores son almacenados en la carpeta MEDIA cuando son leídos y enviados, dichos archivos con extensión de texto permiten compartir información entre los ejecutables. Finalmente, la comunicación entre la computadora y dispositivos físicos se realiza con protocolo serial.

- **Lectura de los dispositivos:** Para controlar los dispositivos Arduino UNO y *RoboClaw*s se ha utilizado un concentrador para comunicación serial 2.0 debido a la cantidad de conexiones requeridas. La cámara Intel realsense D435i se ha conectado directamente al puerto 3.0 de la computadora debido al requerimiento para obtener el máximo rendimiento dado en su manual de usuario y la pantalla táctil se ha acoplado directamente sobre la placa de la computadora con el puerto HDMI.

3.4.2. Bloque módulo cliente

Este módulo es una computadora cliente que contiene un ejecutable, el mismo despliega una interfaz con características por pestaña: una para comunicación que permite conectarse al servidor, otra para la configuración de la cámara, parámetros del *streaming* y de la reconstrucción 3D con Dense SLAM para mapear el entorno donde opera el robot, una más para controlar el robot de manera manual teniendo indicadores gráficos de la dirección y finalmente el modo automático para la carga de un espacio reconstruido con SLAM, la selección de coordenadas de manera intuitiva a través del ratón y el mapa con ayuda de una interfaz de visualización como también la configuración de los parámetros de control para el punto inicial y final del recorrido del robot.

- **Comunicación:** El cliente se comunica inalámbricamente con el servidor de igual forma como el servidor lo hace con el cliente, a través de *sockets* programados en Python con el protocolo TCP/IP. Los puertos para ello son: 55555 para profundidad, 55556 para el color y 55557 para los parámetros. En la Figura 3.31, se muestra la pestaña Comunicación con la forma correcta de configuración de IP y puertos.

Figura 3.31: Pestaña de comunicación

- **Pestaña de cámara y control de *stream*:** Sobre la pestaña presentada en la Figura 3.32, se puede configurar la potencia de la cámara, la resolución de operación en color, profundidad y los fotogramas por segundo, la percepción de profundidad para filtrar en la nube de puntos el ruido generado por el reflejo y el alcance óptimo. También permite desplegar la visualización 2D de color/profundidad en el servidor y el cliente, una visualización 3D en el servidor y una más de manera local en el cliente. Por otro lado permite la reducción del tamaño en la nube de puntos al minimizar su resolución, tamaño de los puntos y su espaciado, permitiendo la ejecución del *stream* 3D y de la reconstrucción con SLAM dependiendo de la necesidad del cliente teniendo en cuenta calidad/velocidad de procesamiento de acuerdo a la potencia de la computadora que posea.
- **Pestaña de modo manual:** En la pestaña de modo manual presentada en la Figura 3.33, se configura la dirección de operación del robot por medio de combinación de teclas y la velocidad por medio de un botón deslizador, también cabe recalcar que en este modo se reinician las configuraciones previas dadas en el modo automático.

Figura 3.32: Pestaña de cámara y control de stream

Figura 3.33: Pestaña de modo manual

- **Pestaña de modo automático** Sobre la última pestaña, presentada en la Figura 3.34 se encuentra el modo automático que permite configurar el punto inicial y final del robot las posiciones de acuerdo al modelo cinemático y la posición del efector sobre dichos puntos, también dispone de botones para lograr que el robot retorne al punto de partida y otro para iniciar el seguimiento. Asimismo, se encuentra embebido el ejecutable para la visualización y selección de los puntos de manera intuitiva, permitiendo que el usuario elija los puntos de control de acuerdo a la ubicación inicial conocida dentro del espacio y la ubicación final requerida para entregar un objeto.

Figura 3.34: Pestaña de modo automático

3.5. Operación del sistema

En esta sección se aborda la operación del sistema propuesto y se organiza en dos elementos esenciales. En la primera parte, se explica la preparación de todos los componentes del sistema. En la segunda parte la ejecución del software para adquirir la información, reconstruir el mapa y la selección de los puntos dada la reconstrucción con SLAM para la entrega del paquete desde el origen hasta un punto de llegada.

3.5.1. Preparación del sistema

Inicialmente se debe realizar una preparación de los componentes principales, este proceso se ejecuta en diversos pasos con el fin de asegurar una operación correcta del robot.

- **Preparación de los componentes:** Primero se inicia la computadora del cliente con un portátil Legion 5 Pro que posee un procesador Intel Core i9 13900HX y 16GB RAM DDR5 para el manejo y se conecta a la red por medio de una antena inalámbrica junto al servidor (robot) presentado en la Figura 3.5 a través de su computadora hacia el enrutador. Estos componentes se muestran en la Figura 3.35.

(a) Cliente (Legion 5 pro) (b) Robot (Servidor) (c) Enrutador

Figura 3.35: Preparación de los dispositivos

- **Conexión de componentes:** Completado el paso anterior, la cámara RGB-D debe estar conectada al servidor (Robot) con todos sus componentes encendidos, incluidos los controladores *RoboClaw* y Arduino a través del concentrador a la computadora del robot. Los componentes descritos se encuentran en la sección 3.1.3. En la Figura 3.36 se ofrece una representación detallada del concentrador con sus conexiones.

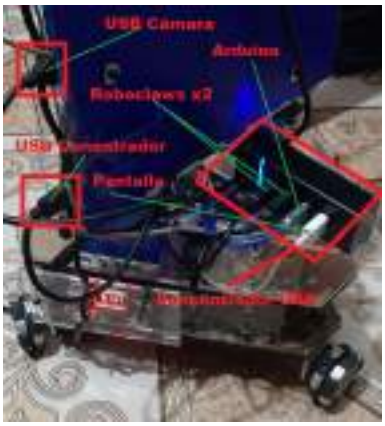


Figura 3.36: Conexiones al concentrador USB

3.5.2. Ejecución del software

La aplicación se ejecuta tal como se ha mencionado en la Figura 3.27 para el servidor y en la Figura 3.28 para el cliente, cada uno en su respectiva computadora. Primero se configura el servidor para ser puesto en modo escucha y después se conecta desde el cliente.

- **Activación del servidor (Robot):** Se activa el servidor desde el ejecutable mostrando sobre ventanas el estado de los componentes como se muestra en la Figura 3.37.

Figura 3.37: Activación del servidor

- **Activación del cliente:** Posteriormente, se lanza el ejecutable del cliente que se encuentra en el Escritorio, como se muestra en la Figura 3.38.

Figura 3.38: Activación del Cliente

- **Configuración del *Stream*:** Al dirigirse a la pestaña de *Camera and Stream Ctrl* como se muestra en la Figura 3.39, se configuran los diferentes parámetros antes de la conexión al robot y hacer la reconstrucción del espacio donde operará. Entre las opciones está la configuración de SLAM que permite elegir cuantas imágenes 3D se van a guardar *N - FRAMES*, la densidad de la nube de puntos *Voxel Size*, tamaño de los puntos *Point Size*, espaciado entre puntos *Uniform Down* y cada cuanto se tomara una imagen en milisegundos *Sampling (ms)*. Para la configuración de la cámara se tiene la resolución, los fotogramas por segundo y inicializar *RUN*.
  Se debe tener en cuenta que al desactivar *Clear FRAMES* se empiezan a guardar las capturas hasta alcanzar el tope de *N - FRAMES* y al activarlo entonces se borrarán todas las capturas almacenadas.

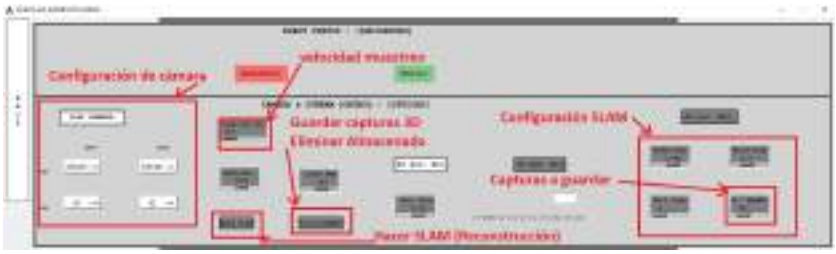


Figura 3.39: Configuración del stream

- **Configuración de la comunicación:** Seguido, se selecciona la pestaña de *COM Crtl* para configurar la IP y puertos del robot que han sido generados al activar los servicios, como se muestra en la Figura 3.37. En la Figura 3.40, se evidencia como se han ingresado los campos requeridos.

Figura 3.40: Configuración de la comunicación

- **Adquisición de las imágenes:** Al conectar el cliente con el servidor se empiezan a obtener las imágenes que puede ser visualizadas en línea como se evidencia en la Figura 3.41 y se guardan en orden hasta alcanzar el tope, como se muestra en la Figura 3.42.

Figura 3.41: Adquisición de las imágenes en línea (Stream)

Figura 3.42: Nubes de puntos guardadas

- **Movimiento del robot en modo manual:** Mientras se capturan las imágenes se debe manejar el robot lentamente con el modo manual en la pestaña *Manual*. Los movimientos deben ser suaves para permitir obtener una cantidad adecuada de información del espacio. El robot se puede mover a una velocidad entre 0 a 12000 y varía por medio del botón deslizador *Wheels Velocity*. Para moverlo en diferentes direcciones se usan combinaciones del teclado como se aprecia en la Figura 3.43, las *Flechas* lo mueven en la dirección seleccionada, al combinar más de una flecha como por ejemplo *Arriba + Izquierda* el robot se moverá en la diagonal izquierda avanzando, como se muestra en la Figura 3.44, es decir hacia el frente e izquierda y lo mismo para todas las combinaciones restantes. Al presionar *Control + Derecha* el robot gira sobre su propio eje a la derecha y con *Control + Izquierda* hacia la izquierda, como se exhibe en la Figura 3.45. De manera lúdica se puede manipular las pinzas con *Control + Shift + Derecha* para cerrar como se muestra en la Figura 3.46 y *Control + Shift + Izquierda* para abrir, como se evidencia en la Figura 3.45.

  La siguiente lista especifica las todas combinaciones de comandos para el robot:

  1. *Control + Shift + Derecha* : Abrir pinza
  2. *Control + Shift + Izquierda* : Cerrar pinza
  3. *Arriba* : Avanzar
  4. *Abajo* : Retroceder
  5. *Derecha* : Movimiento recto hacia la derecha
  6. *Izquierda* : Movimiento recto hacia la izquierda
  7. *Arriba + Derecha* : Avanzar diagonalmente hacia la derecha
  8. *Arriba + Izquierda* : Avanzar diagonalmente hacia la izquierda
  9. *Abajo + Derecha* : Retroceder diagonalmente hacia la derecha
  10. *Abajo + Izquierda* : Retroceder diagonalmente hacia la izquierda
  11. *Control + Derecha* : Giro sobre el propio eje hacia la derecha
  12. *Control + Izquierda* : Giro sobre el propio eje hacia la izquierda

Figura 3.43: Teclas asignadas para combinación de comandos

Figura 3.44: Movimiento diagonal hacia la izquierda y avanzando

Figura 3.45: Giro sobre el propio eje hacia la izquierda

Figura 3.46: Cerrar pinzas

- **Reconstrucción del entorno:** Al tener el barrido del entorno con las capturas 3D se acciona sobre el botón *Make SLAM* mostrado en la Figura 3.39 y se abre el código de SLAM, surge un recuadro de comandos que solicita el nombre de guardado del mapa 3D que será reconstruido y empieza su ejecución, como se aprecia en la Figura 3.47. Una vez finalizado, se exhibe en la Figura 3.48 el mapa 3D con las poses que ha tenido el robot y se guarda en la ruta *MEDIA..saved..reconstructedpointcloud.*

Figura 3.47: Script de SLAM en ejecución

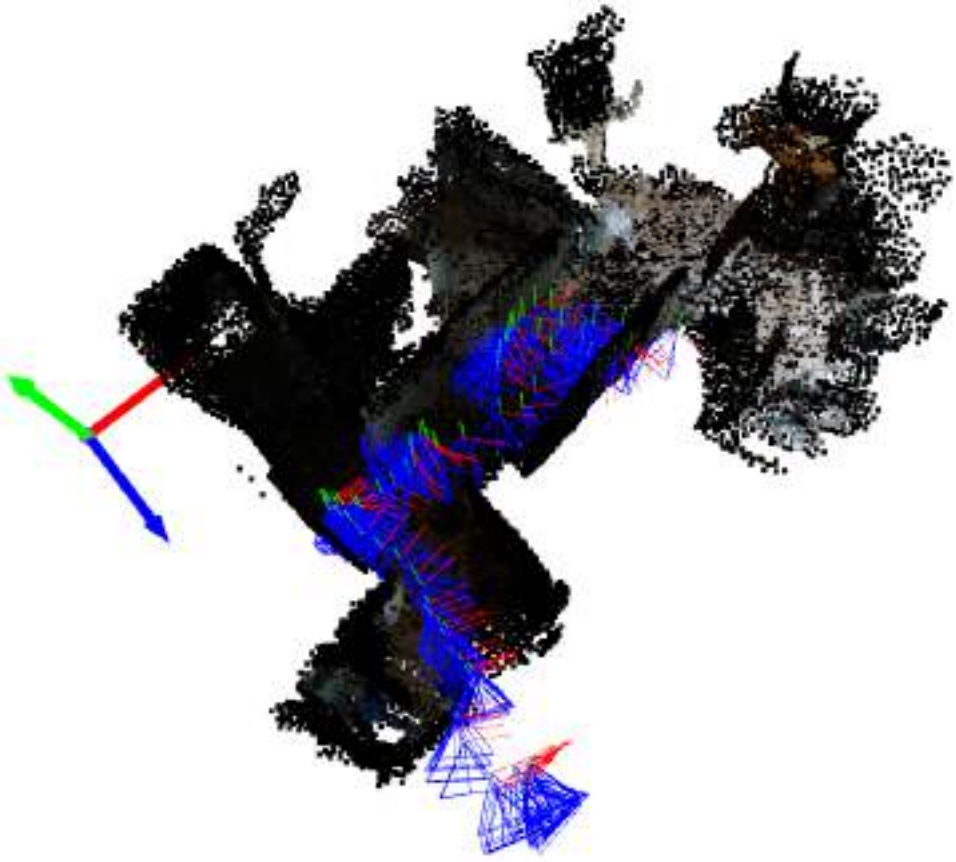


Figura 3.48: Mapa 3D reconstruido

- **Selección de las coordenadas:** Por último, es presionado el botón de *Automatic* y el robot entra en modo automático para el seguimiento de trayectorias. Las trayectorias son generadas por el usuario a través de las interfaces gráficas. En la interfaz denominada *POINT SELECTOR* se seleccionan los puntos con la combinación del teclado y ratón *Shift + Click izquierdo* para agregar un punto y para eliminar el

último punto *Shift + Click derecho*. Sobre la interfaz *COORDINATE CONFIGURATOR* se visualizan las trayectorias que tendrá el robot en el espacio.

Las configuraciones restantes como orientación, carga del mapa desde una ruta y estados de las pinzas para el punto inicial y final se realizan a través de los campos y botones de la pestaña, cabe resaltar que todas las configuraciones no serán establecidas hasta accionar en el teclado *ENTER*.

En la Figura 3.49 se explica el uso de la pestaña *Automatic* junto a las interfaces para configurar las coordenadas:

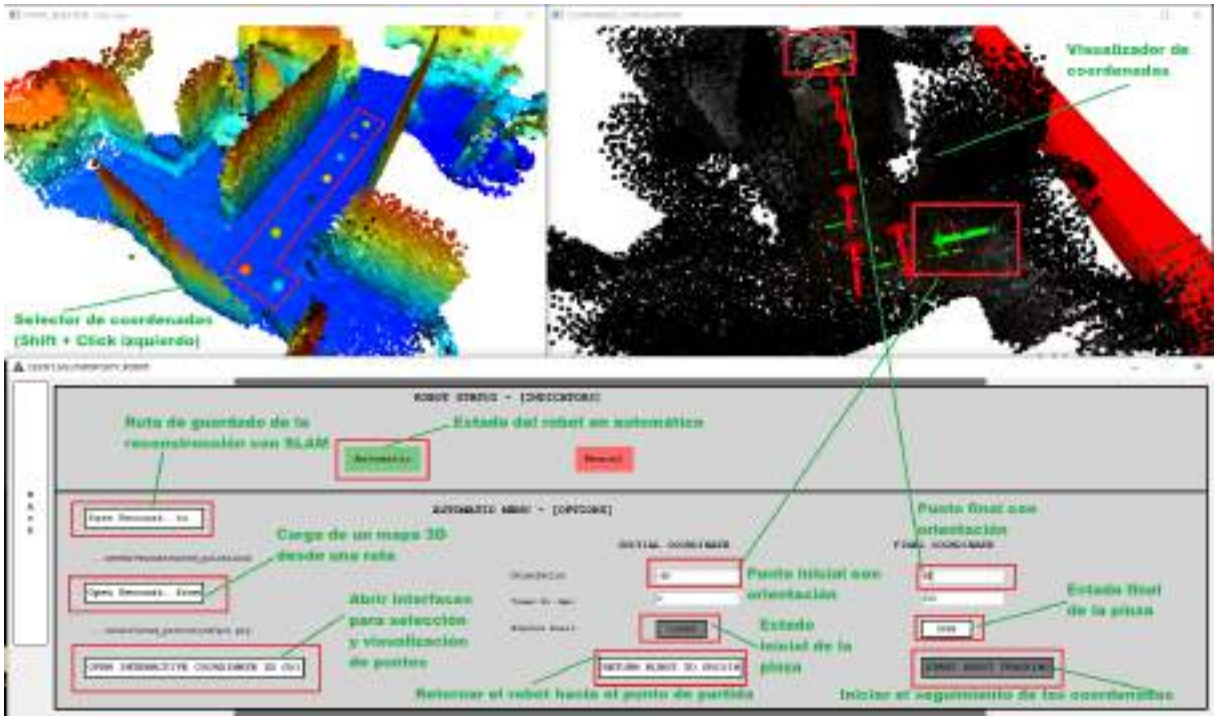


Figura 3.49: Configuraciones de pestaña modo automático y selección de coordenadas para seguimiento

Capítulo 4

Resultados

4.1. Prototipo final

El prototipo final del robot omnidireccional, conformado por tres ruedas suecas dispuestas a noventa grados, fue diseñado utilizando la herramienta CAD *SolidWorks* en el CTPI, Centro de Teleinformática y Producción Industrial de Popayán, con el apoyo y las recomendaciones de aprendices con experiencia. El diseño se basó en las proporciones reportadas en el estado del arte, con el objetivo de garantizar un comportamiento dinámico adecuado. Para la estructura se emplearon dos tipos de materiales: una primera capa en MDF de 5 mm, que proporciona rigidez, y una segunda capa en acrílico transparente de 5 mm, que mejora la estética del robot.

Para la fijación de los motores se utilizaron ángulos de aluminio a 90°, los cuales fueron mecanizados en un torno industrial para tallar orificios que coincidieran con los agujeros roscados de los motores Pololu. De esta manera, cada motor se acopla a una de las caras del ángulo, mientras que la otra se fija al chasis.

Las piezas modulares del robot fueron cortadas con maquinaria láser y ensambladas mediante elementos de refuerzo (nervios) unidos con tornillos M3, M4 y M5 de cabeza hexagonal, acompañados de tuercas de seguridad. Los tornillos M3, M4 y M5 pertenecen al sistema métrico ISO, donde la letra M indica que poseen una rosca métrica y el número representa el diámetro exterior de la rosca en milímetros: 3 mm, 4 mm y 5 mm respectivamente. Estos tornillos son ampliamente utilizados en aplicaciones mecánicas y robóticas debido a su tamaño compacto y facilidad de integración.

El robot cuenta con una estructura interna en forma de triángulo, donde se alojan todas las conexiones y componentes eléctricos. Esta disposición no solo permite un diseño compacto, sino que también proporciona protección adicional frente a la intemperie y a la manipulación externa. Para favorecer la estabilidad y el equilibrio del sistema, se fijó un ángulo de aluminio en el centro del chasis, sobre el cual se sujetó la computadora, ubicada encima del conjunto de baterías. En la parte frontal, el chasis presenta un corte cuadrado destinado a la instalación de una pinza y/o cualquier otro efector final que se

desee incorporar en el futuro.

La Figura 4.1 muestra la implementación del robot móvil con tres ruedas suecas dispuestas a 90°. Este diseño fue concebido con base en los fundamentos teóricos previos, considerando aspectos como la ubicación del centro de masa y el equilibrio de fuerzas entre los componentes. Se optó por una estructura robusta, conformada por piezas cortadas mediante maquinaria de precisión, lo cual garantiza una separación exacta de 120° entre las ruedas. De esta manera, se logra una mayor correspondencia entre el modelo cinemático teórico y la construcción física del robot.

Finalmente, en la Figura 4.1 y en la Figura 4.2 se evidencia el robot en diferentes poses, lo cual permite apreciar su configuración física final, así como su capacidad para adoptar distintas orientaciones en el entorno de operación dada su configuración omnidireccional con tres ruedas. Estas imágenes proporcionan una visión general del resultado del proceso de diseño e implementación propuestas.

Figura 4.1: Robot móvil implementado

(a) Vista lateral (b) Vista posterior

Figura 4.2: Versión final del robot en diferentes poses

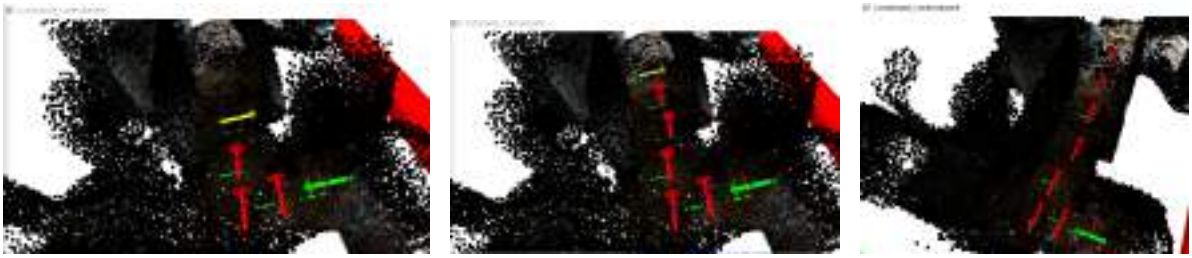
4.2. Validación

Durante esta fase, se evaluó el desempeño del robot midiendo manualmente la distancia, en milímetros, entre el punto de partida y el punto de retorno, con el fin de determinar su exactitud. Para ello, se realizó la reconstrucción tridimensional del espacio mostrado en la Figura 3.48, correspondiente a una de las oficinas del CTPI, y se seleccionaron los puntos de referencia según lo ilustrado en la Figura 3.49. Posteriormente, se instaló una pequeña caja —simulando una encomienda— sobre la pinza del robot.

Para saber la exactitud se adhirió con cinta un marcador en el suelo para el sitio de partida del robot como se muestra en la Figura 4.3, de tal forma que permita medir el desplazamiento cuanto retorne, ya que para saber la exactitud en el punto de entrega de la caja puede ser erróneo dada la incertidumbre del punto seleccionado, puede haber una discrepancia entre la colocación de un marcador en el mundo real para el punto final a la selección dada por software en el mapa 3D, pero sí hay certeza del punto donde arranca el robot, por tanto, se puede medir con un marcador si regresa al mismo punto, no importa si no es el mismo que el seleccionado por software.

Figura 4.3: Robot posicionado sobre el marcador

Se realizaron diez pruebas con distancias cada vez más largas que permitan evaluar el error de desplazamiento a medida que aumenta a distancia. En la Figura 4.4 y Figura 4.5, se evidencian las selecciones de puntos para las pruebas, cada vez con trayectorias más amplias.



(a) Prueba 1 (b) Prueba 2 (c) Prueba 3

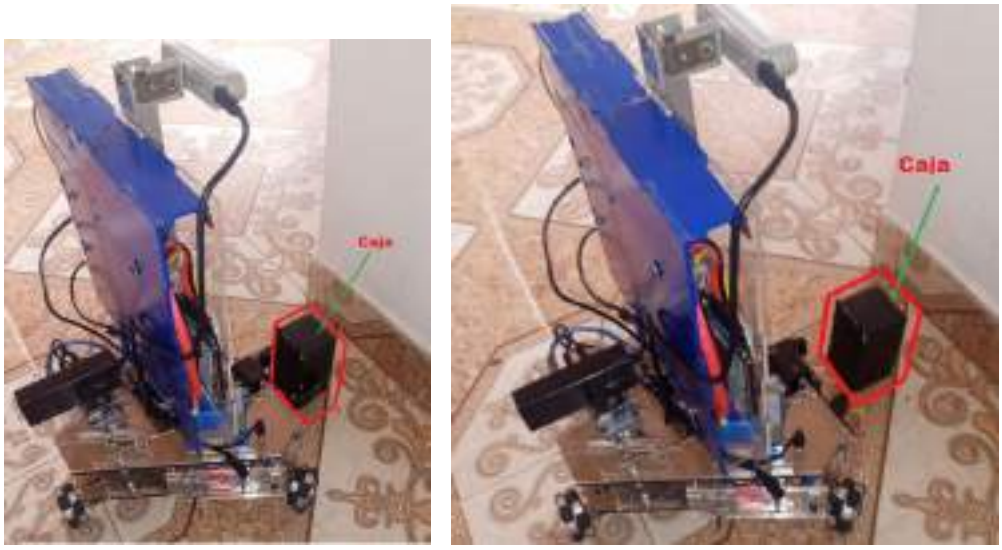
Figura 4.4: Selección de seguimiento de trayectorias

Cuando se siguen las trayectorias, el robot suelta la caja en el punto final, como se muestra en la Figura 4.6. Seguido, el robot retorna hacia el punto de origen donde se encuentra el marcador como se muestra en la Figura 4.7.

Se realizaron diez pruebas siguiendo los pasos descritos previamente, cada una con trayectorias progresivamente más largas. Al finalizar cada recorrido, se midió el error en milímetros una vez el robot retornaba al punto de origen, utilizando como referencia el marcador previamente definido. En la Figura 4.8, se presentan ejemplos correspondientes

(a) Prueba 4 (b) Prueba 5

Figura 4.5: Selección de seguimiento de trayectorias 2



(a) Caja sin soltar (b) Caja entregada en el punto final

Figura 4.6: Entrega de la caja en el punto final

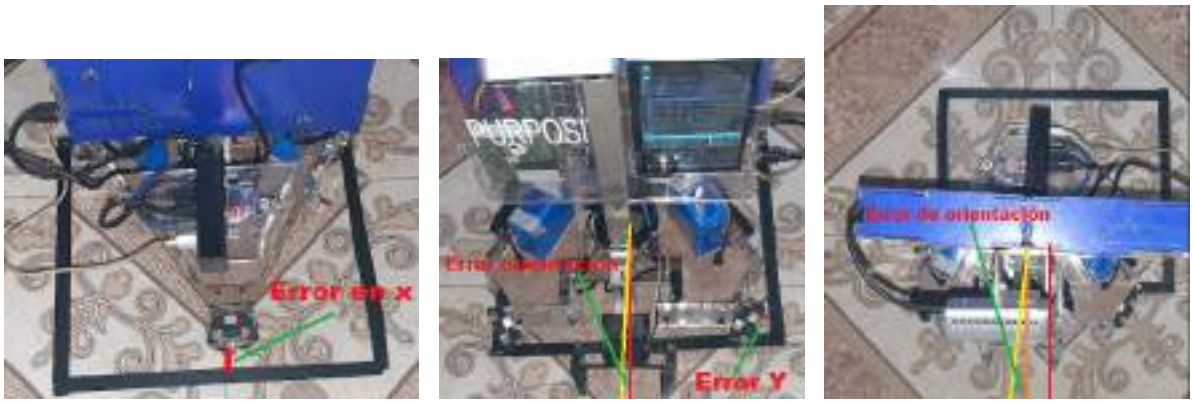
a las pruebas dos y tres, junto con la medición de algunos errores registrados. El error en las coordenadas X y Y se determinó utilizando una regla; la orientación angular se midió con un transportador, y la distancia total recorrida se obtuvo mediante un flexómetro. Esta misma lógica se aplicó para la evaluación de los demás puntos de prueba.

Resumen de las pruebas

Realizadas la diez pruebas, se obtuvo la medición de los errores utilizando la metodología mencionada con anterioridad, en la Tabla 4.1 se detalla el error en X, error en Y, error de orientación (ϕ) y la distancia total recorrida del robot para cada prueba:

(a) Retorno punto final (b) Retorno medio camino (c) Retorno cerca al marcador

Figura 4.7: Retorno del robot hacia el punto de origen



(a) Medida error X prueba 2 (b) Medida errores Y, orientación prueba 2 (c) Medida error orientación prueba 3

Figura 4.8: Medida de errores en el origen

Análisis de resultados

A partir de los datos mostrados en la Tabla 4.1, se puede realizar un análisis de la exactitud del posicionamiento del robot omnidireccional de tres ruedas. Se observa tendencia del error en función de la distancia como un incremento progresivo en los errores de posición y orientación a medida que aumenta la distancia total recorrida. Por ejemplo, en la prueba 1 (1.84 m), el error en X fue de 9 mm, mientras que en la prueba 5 (4.79 m) fue de 35 mm. Esto sugiere una relación directamente proporcional entre la distancia recorrida y el error acumulado, como es típico en sistemas que no poseen realimentación continua.

Por otro lado, respecto al error en orientación ϕ también aumenta con la distancia, desde 4° en la prueba 1 hasta 31° en la prueba 10. Este comportamiento indica que, además del error de traslación, el robot presenta desviaciones en su dirección final, posiblemente por deslizamientos o imprecisión en el control de velocidad diferencial de sus ruedas.

Tabla 4.1: Medida de errores del robot

Prueba	Error X (mm)	Error Y (mm)	Error ϕ (grados)	Distancia total (m)
1	9	6	4	1.84
2	13	10	12	2.82
3	21	16	14	3.58
4	29	21	17	4.12
5	35	27	22	4.79
6	42	32	24	5.67
7	50	38	26	6.55
8	58	44	28	7.43
9	66	50	29	8.45
10	69	55	31	9.48

Aunque los errores son bajos en distancias cortas, se acumulan significativamente en trayectos largos. Esto muestra que el sistema tiene una exactitud aceptable en trayectos cortos, pero podría requerir corrección mediante sensores de retroalimentación para trayectorias más largas o precisas. Los errores en X y Y siguen una proporción similar, lo cual indica un comportamiento relativamente equilibrado en ambos ejes. Sin embargo, el error en X tiende a ser ligeramente mayor, lo que podría indicar una leve asimetría en el modelo presentado.

Finalmente, la consistencia observada en el patrón de errores sugiere que el sistema de locomoción presenta un comportamiento predecible, lo cual es una ventaja desde el punto de vista del control. Esta característica permitiría aplicar técnicas de corrección basadas en modelos, como filtros de Kalman o controladores adaptativos, que compensen sistemáticamente los errores conocidos. De este modo, aunque el robot no sea perfectamente exacto por sí solo, podría alcanzar altos niveles de precisión mediante algoritmos adicionales y control por retroalimentación.

Capítulo 5

Discusión y conclusión

En este trabajo de grado se construyó un robot móvil repartidor omnidireccional de tres ruedas suecas orientadas a 90°, junto con una herramienta software para la reconstrucción de entornos, generación de trayectorias y su seguimiento en espacios tridimensionales estáticos. Esta iniciativa surge de la necesidad detectada en el Centro de Teleinformática y Producción Industrial del SENA Popayán de disponer de un robot móvil de pequeña escala acompañado de un software intuitivo que permitiera generar y seguir trayectorias, operando en sistemas Windows sin requerir el uso de ROS ni habilidades avanzadas de programación. De esta manera, se buscó facilitar el aprendizaje y la manipulación del sistema por parte de los usuarios, sin necesidad de conocimientos previos en codificación, *Visual SLAM* o control cinemático.

El sistema desarrollado integra tres componentes principales: (i) una plataforma robótica móvil equipada con dos controladores Roboclaw2X15A, un Arduino UNO, dos baterias de 12 V a 12000 mAh, una pinza servocontrolada, tres reguladores LM2596, una computadora con core i7-4710MQ y 8GB de RAM DDR3 con pantalla táctil UCTRONICS UC-595 y tres motores Pololu 4752 con sus respectivas ruedas omnidireccionales a noventa grado, (ii) un sensor de profundidad RGB-D Intel Realsense D435i para la captura del entorno, y (iii) una interfaz gráfica de usuario diseñada para visualizar la reconstrucción tridimensional, la generación de trayectorias y su ejecución. La captura y reconstrucción de los espacios se realizó utilizando el algoritmo de visión por computadora SLAM aplicados sobre los datos de profundidad.

La planificación de la trayectoria fue seleccionada por el usuario dentro del entorno mapeado y el seguimiento de trayectorias está basada en las soluciones de las ecuaciones cinemáticas del robot . La validación del sistema se efectuó en las instalaciones del Centro de Teleinformática y Producción Industrial del SENA Popayán, donde se llevaron a cabo pruebas de entrega de paquetes entre ubicaciones específicas dentro de un espacio tridimensional previamente reconstruido.

Los resultados de la validación del robot para evaluar el error de desplazamiento a medida que aumenta a distancia se encuentran en la Tabla 4.1. Del cual se puede decir

que a medida que la distancia aumenta el error es más grande. El error es mayor para X y el más bajo para la orientación.

La limitación del robot evaluado es la ausencia de retroalimentación de posición cuando se sigue la trayectorias propuesta ya que al operar exclusivamente con control abierto, el robot ejecuta los comandos de movimiento sin evaluar si realmente la posición y orientación deseadas son las correspondientes al espacio operado. Esto significa que cualquier desviación generada por errores de control, variaciones en el terreno o perturbaciones externas no puede ser corregida automáticamente, lo cual repercute directamente en la exactitud del sistema, especialmente en trayectorias largas.

Otro aspecto relevante es la posible influencia del peso y la distribución de los componentes del robot. Si el centro de masa no está bien equilibrado o si existe una sobrecarga en una sección del chasis, se pueden generar variaciones en el comportamiento dinámico del robot durante la marcha. Estas variaciones afectan la tracción y el desempeño de las ruedas, alterando la respuesta esperada frente a las señales de control. Esto se agrava si el sistema no ha sido calibrado considerando el peso real en operación.

Además, el modelo de control implementado se basa únicamente en la cinemática del robot, ignorando completamente los efectos dinámicos como la inercia, las fuerzas de fricción o el deslizamiento de las ruedas. Aunque este enfoque cinemático es útil para tareas de planificación básica, resulta limitado para aplicaciones de exactitud, ya que no contempla cómo el sistema reacciona realmente ante aceleraciones, pendientes o cambios bruscos de dirección.

En este tipo de robots, también es importante tener en cuenta las pérdidas por fricción y las imperfecciones mecánicas, como el roce desigual entre ruedas, deformaciones en la superficie de contacto o juegos mecánicos en los motores. Estos elementos introducen errores que no siempre son constantes ni fáciles de predecir, y que con el tiempo pueden aumentar por el desgaste de los componentes. Estas pérdidas se traducen en pequeños desvíos que, acumulados en el tiempo, reducen la fidelidad de la trayectoria ejecutada. Por tanto, podemos concluir que la exactitud del posicionamiento y la orientación de un robot omnidireccional de tres ruedas puede verse afectada por una combinación de factores: la falta de sensores de realimentación, la simplificación del modelo de control, la influencia de la dinámica del sistema y las pérdidas no modeladas. Por tanto, si se requiere mayor precisión en el desplazamiento, será necesario incorporar estrategias de control más avanzadas y mecanismos de retroalimentación, como sensores inerciales, sistemas de visión o técnicas de localización basadas en mapas. Solo así se podrá garantizar un rendimiento más robusto y confiable en escenarios reales.

Por otro lado, dadas las limitaciones de tiempo y conocimientos, cabe aclarar que el robot tiene aspectos claves que mejorar respecto a soluciones actuales y algoritmos adicionales que deben tenerse en cuenta para temas de autonomía en espacios de mayor complejidad.

Otra limitación importante es que solo cuenta con una cámara 3D con alcance de tres metros, si se mapean entornos muy amplios la misma no podrá abstraer información relevante más que el suelo y por la naturaleza del SLAM que solo ha utilizado las capturas 3D el emparejamiento del mapa será incongruente y no sabrá si el robot está avanzando, pues lo único que detectará es suelo, que no posee diferenciación.

También, al utilizar una cámara RGB-D, presenta limitación en cuanto a rango efectivo y precisión en condiciones de iluminación variable, reflectancia sobre vidrio y luz generando ruido o falta de información en estos casos. Una alternativa para superar estas restricciones sería la incorporación de sensores LiDAR 3D de mayor alcance, que permitirían capturar mapas tridimensionales más extensos y detallados, reduciendo los errores en la reconstrucción del entorno.

Aunque el prototipo propuesto en este trabajo de grado presenta ventajas significativas en términos de facilidad de uso y costos de fabricación. Sin embargo, para optimizar su funcionamiento, se podrían implementar algunas mejoras de hardware y software que permitirían optimizar su desempeño y aumentar su fiabilidad en aplicaciones futuras.

Además, el sistema de generación de trayectorias y seguimiento podría beneficiarse de una mejor retroalimentación posicional si se integraran al algoritmo SLAM el cálculo de la posición global con los *encoders* de cuadratura en las ruedas, lo cual ofrecería una estimación más precisa del desplazamiento del robot a través de la odometría y, por tanto, mejores reconstrucciones. Esta información, combinada con los datos de visión por computadora, permitiría implementar un sistema de SLAM más robusto y fiable, donde la fusión de sensores mejoraría tanto la localización como la construcción de mapas en tiempo real. Tales mejoras no solo incrementarían la autonomía del robot, sino que también extenderían su aplicabilidad a escenarios más complejos o dinámicos, manteniendo la premisa de ser una herramienta accesible y fácil de operar para usuarios sin experiencia en programación avanzada.

La plataforma robótica propuesta representa una alternativa prometedora para fines educativos y de desarrollo básico, especialmente por su accesibilidad y simplicidad en entornos *Windows*. Sin embargo, presenta limitaciones importantes frente a las soluciones modernas basadas en ROS (Robot Operating System) que actualmente constituye el estándar en robótica.

De lo anterior se puede reconocer que ROS ofrece una gran variedad de paquetes y algoritmos ya desarrollados, incluyendo soluciones avanzadas de SLAM que permiten un desarrollo más rápido y robusto sin necesidad de implementar funcionalidades desde cero. Además, la complejidad y escalabilidad que exigen las aplicaciones robóticas actuales suelen superar lo que una plataforma simplificada puede manejar, por lo que su utilidad se ve aplicada como aprendizaje y restringida a contextos más exigentes.

Una de las principales áreas de mejora de la plataforma actual para trabajos futuros es perfeccionar SLAM con la integración ROS. A diferencia de sistemas desarrollados con comunicación basada únicamente en *sockets*, ROS proporciona un entorno modular y escalable que permite una gestión más eficiente de sensores, motores, y procesos de localización. Al implementar ROS, se facilitaría la interoperabilidad con múltiples herramientas de robótica, se optimizaría el flujo de datos en tiempo real, y se habilitaría el uso de nodos ya desarrollados para tareas como navegación autónoma junto a paquetes para percepción y control sin tener que desarrollar toda la plataforma desde cero usando librerías de terceros.

En la Tabla 5.1 se muestran comparaciones entre ROS y Python al momento de crear plataformas como las aplicadas en este trabajo de grado. Características como la arquitectura de conexión, la escalabilidad, la comunicación entre sus componentes, la visualización de los datos, soporte para SLAM y el mantenimiento son importantes. Por ejemplo, los *sockets* utilizados para la comunicación del robot son reemplazables por los nodos de ROS y los programas externos a ROS como el procesamiento de la cámara y la interfaz pueden manejarse en Python para lograr mayor modularidad de componentes y tecnologías del proyecto.

Tabla 5.1: Comparación entre la plataforma y ROS

Característica	**Python**	**ROS 2**
Arquitectura	Punto a punto (manual)	Modular y distribuida
Escalabilidad	Limitada	Alta
Comunicación entre módulos	Programada manualmente (TCP/UDP)	Mediante temas, servicios y acciones
Nodos reutilizables	No disponible	Sí
Interoperabilidad	Limitada	Alta (estandarizada)
Visualización integrada	No incluida (debe desarrollarse)	Herramientas como `rviz`, `rqt`, `rosbag`
Compatibilidad con Python	Totalmente compatible	Compatible mediante `rclpy`
Soporte para SLAM	Requiere implementación desde cero	Existen módulos SLAM
Mantenimiento	Complejo y manual	Diagnóstico y monitoreo incluidos

A partir de la comparación presentada en la Tabla 5.1 y las plataformas SLAM mencionadas en la Tabla 2.2, se evidencian diferencias sustanciales entre las plataformas de código abierto de última generación como ORB-SLAM2, Kimera o VINS-Fusion, y la implementación propia desarrollada únicamente en Python utilizando la librería *Open3D*. Mientras que las plataformas reconocidas cuentan con documentación consolidada, contenedores *Docker* y soporte completo para ROS que permiten ejecutar procesos de SLAM en tiempo real sobre entornos Linux, su instalación y uso suelen requerir conocimientos avanzados y resultan poco intuitivos para usuarios no especializados. En contraste, la plataforma desarrollada en este trabajo es ejecutable directamente sobre *Windows*, posee una

interfaz gráfica personalizada y permite visualizar de forma inmediata las poses estimadas y las trayectorias reconstruidas en 3D a través de *Open3D*, facilitando la comprensión del funcionamiento del SLAM. No obstante, esta propuesta carece de documentación pública, no cuenta con un sistema de despliegue en contenedores, ni está optimizada para funcionar en entornos ROS o escenarios de alto rendimiento. Así, aunque la solución propuesta no alcanza la precisión, escalabilidad ni interoperabilidad de los sistemas más consolidados, destaca por su sencillez, portabilidad y utilidad como plataforma de experimentación, visualización y enseñanza.

Otra línea importante de mejora para la plataforma desarrollada, dejando de lado la integración con sistemas como ROS, se relaciona con su capacidad de adaptarse de manera autónoma a entornos cambiantes. Actualmente, el robot no cuenta con un sistema que le permita recalcular su ruta automáticamente cuando encuentra un obstáculo, lo cual limita su funcionalidad en escenarios reales o no controlados. Incluir algoritmos de planificación de rutas como búsqueda en grafos, expansión por zonas o caminos alternativos permitiría generar trayectorias nuevas en tiempo real, sin necesidad de reiniciar todo el proceso de navegación.

Asimismo, el sistema no realiza segmentación automática del terreno o del entorno, es decir, no puede distinguir de forma inteligente qué zonas son transitables, peligrosas o restringidas. La implementación de estas funciones de replanificación de rutas y segmentación automática mejoraría considerablemente el nivel de autonomía del robot y permitiría una navegación más robusta y segura, acercándose a las características deseables en sistemas de robótica móvil para enseñanza como la propuesta por [23].

Finalmente se concluye que, aunque la plataforma desarrollada en este trabajo representa un avance importante en términos de visualización y control para un robot desde Python, existen múltiples oportunidades de mejora que permitirían aumentar significativamente su funcionalidad y grado de autonomía. En primer lugar, la integración con un sistema de comunicación modular como ROS permitiría escalar el proyecto y en segundo lugar la capacidad del robot para adaptarse a entornos dinámicos.

5.1. Anexos

```
class Keyframe():
    def __init__(self, id, cloud, fpfh, odom):
        self.id = id
        self.cloud = copy.deepcopy(cloud)
        self.fpfh = copy.deepcopy(fpfh)
        self.odom = copy.deepcopy(odom)
        self.node = o3d.pipelines.registration.PoseGraphNode(odom)
```

Listing 5.1: Clase KeyFrame

```
class SLAM():
```

```
    def __init__(self, voxel_size, color_mode = False):
        self.voxel_size = voxel_size
        self.graph = o3d.pipelines.registration.PoseGraph()
        self.keyframes = []
        self.camera_marker = []
        self.traj = []
        self.raw_traj = []
        self.color_mode = color_mode
```

Listing 5.2: Clase SLAM

```
def optimize_posegraph(self):
    option = o3d.pipelines.registration.GlobalOptimizationOption(
        max_correspondence_distance=self.voxel_size * 1.2,
        edge_prune_threshold=0.25,
        reference_node=0
    )
    o3d.pipelines.registration.global_optimization(
        self.graph,
        o3d.pipelines.registration.GlobalOptimization
        LevenbergMarquardt(),
        o3d.pipelines.registration.GlobalOptimization
        ConvergenCeCriteria(),
        option
    )
```

Listing 5.3: Función de Optimización del gráfico de poses con Levenberg & Marquardt

```
def update(self, cloud):
        cloud = cloud.voxel_down_sample(self.voxel_size)
        cloud, fpfh = compute_features(cloud, self.voxel_size)

        if not len(self.keyframes):
            #odom = np.array([[1, 0, 0, 1.5], [0, 1, 0, 1.5], [0, 0, 1,
               -0.3], [0, 0, 0, 1]])
            odom = np.identity(4)
            self.keyframes.append(Keyframe(0, cloud, fpfh, odom))
            self.graph.nodes.append(self.keyframes[-1].node)

            odom_inv = np.linalg.inv(odom)
            self.frustum = o3d.geometry.LineSet.
            create_camera_visualization(intrinsic,odom_inv, 1)
            return

        if self.update_keyframe(cloud, fpfh):
            self.optimize_posegraph()


    def update_keyframe(self, cloud, fpfh):
        success, transformation, information =
           register_point_cloud_fpfh(
            cloud, self.keyframes[-1].cloud,
            fpfh, self.keyframes[-1].fpfh,
```

```
            self.voxel_size, True, color_mode= self.color_mode
        )

        if not success:
            return False

        odom = np.dot(self.graph.nodes[-1].pose, transformation)
        self.keyframes.append(Keyframe(len(self.keyframes), cloud,
           fpfh, odom))

        odom_inv = np.linalg.inv(odom)
        self.frustum = o3d.geometry.LineSet.
        create_camera_visualization(intrinsic, odom_inv, 0.3)
        self.camera_marker.append(self.frustum)

        self.graph.nodes.append(self.keyframes[-1].node)
        edge = o3d.pipelines.registration.PoseGraphEdge(
            self.keyframes[-1].id, self.keyframes[-2].id,
            transformation, information, uncertain=False
        )
        self.graph.edges.append(edge)

        for i in range(0, len(self.keyframes) - 5, 5):
            transformation =
               np.dot(np.linalg.inv(self.keyframes[i].odom),
               self.keyframes[-1].odom)
            trans = compute_distance(transformation)
            rot = compute_angle(transformation)

            if trans < 1 and rot < (np.pi / 18):
                success, transformation, information =
                   register_point_cloud_fpfh(
                    self.keyframes[i].cloud, cloud,
                    self.keyframes[i].fpfh, fpfh,
                    self.voxel_size, False, color_mode= self.color_mode
                )
                if success:
                    edge = o3d.pipelines.registration.PoseGraphEdge(
                        self.keyframes[i].id, self.keyframes[-1].id,
                        transformation, information, uncertain=True
                    )
                    self.graph.edges.append(edge)
        return True
```

Listing 5.4: Código para el cálculo de las transformaciones

```

for i in range(0,camera_local_par[12]):
    point_cloud =
       o3d.io.read_point_cloud(f"MEDIA\\saved\\pointclouds_frames
    \\pcd_{i:03}.ply")
    print(f"UN-packing: {len(frames_to_recons)}")
    frames_to_recons.append(point_cloud)
```

```
7
8  slam = SL.SLAM(camera_local_par[9], color_mode = True)
9  run_times = []
10 for i in range(0, len(frames_to_recons)):
11
12     source = copy.deepcopy(frames_to_recons[i])
13     source.estimate_normals(search_param=o3d.geometry.
14     KDTreeSearchParamHybrid(radius=0.25, max_nn=25))
15     source  = o3d.geometry.PointCloud.farthest_point_down_sample(source,
16     int(0.3*np.asanyarray(source.points).shape[0]))
17     cl, ind = source.remove_statistical_outlier(nb_neighbors=80,
           std_ratio=0.85)
18     source = source.select_by_index(ind)
19
20     start = time.time()
21     slam. update(source)
22     end = time.time()
23
24     run_times.append(end - start)
25     try:
26         print(f'FRAME {i} of {len(frames_to_recons)} FRAMES, FPS: {1 /
               (end - start):6f}')
27     except ZeroDivisionError:
28         pass
29
30 slam.save_trajectory()
31
32 vis = o3d.visualization.Visualizer()
33 vis.create_window(window_name = "Reconstr 3D",width = 500,
       height=500,visible= True )
34 vis.get_render_option().background_color= np.asarray([1,1,1])
35 print('Generating scene:')
36 pcd_combined = o3d.geometry.PointCloud()
37 vis.add_geometry(pcd_combined)
38
39 for i in tqdm(range(len(slam.keyframes))):
40     pcd_combined +=
           slam.keyframes[i].cloud.transform(slam.graph.nodes[i].pose)
41     vis.update_geometry(pcd_combined)
42     try:
43         vis.add_geometry(slam.camera_marker[i])
44         pass
45     except IndexError:
46         pass
47     vis.get_render_option().show_coordinate_frame = True
48     vis.get_render_option().light_on = False
49     vis.update_geometry(slam.keyframes[i].cloud.transform(
50     slam.graph.nodes[i].pose))
51     vis.reset_view_point(reset_bounding_box=True)
52     vis.poll_events()
53     vis.update_renderer()
54     #time.sleep(0.05)
55     print("make slam: ",pcd_combined)
```

```
56
57 pcd_combined.estimate_normals(search_param=o3d.geometry.
58 KDTreeSearchParamHybrid(radius=0.25, max_nn=25))
59 pcd_combined  =
       o3d.geometry.PointCloud.farthest_point_down_sample(pcd_combined,
60 int(0.7*np.asanyarray(pcd_combined.points).shape[0]))
61 o3d.io.write_point_cloud(f"MEDIA\\saved\\reconstructed_pointcloud\\
62 {file_name}.ply", pcd_combined,format="ply",write_ascii=False)
63 vis.run()
```

Listing 5.5: Código para la aplicación de las transformaciones y guardado del mapa

```
1  def getCoef(t,i_v,f_v,i_t,f_t):
2      Ax = np.array([[i_t**5,i_t**4,i_t**3,i_t**2,i_t,1],
3                     [f_t**5,f_t**4,f_t**3,f_t**2,f_t,1],
4                     [5*i_t**4,4*i_t**3,3*i_t**2,2*i_t,1,0],
5                     [5*f_t**4,4*f_t**3,3*f_t**2,2*f_t,1,0],
6                     [20*i_t**3,12*i_t**2,6*i_t,2,0,0],
7                     [20*f_t**3,12*f_t**2,6*f_t,2,0,0]])
8      b = np.array([[i_v],[  f_v],[0],[0],[0],[0]]) #s_(0), s(f),
           s_d(0),s_d(f),s_dd(0),s_dd(f)
9      x = None
10     try:
11         x = np.linalg.solve(Ax , b ) #A,B,C,D,E,F Coeficients
12     except np.linalg.LinAlgError:
13         #print("Error: Matriz singular, Resolviendo por minimos
               cuadrados.")
14         x, residuals, rank, s = np.linalg.lstsq(Ax, b, rcond=None)
               #A,B,C,D,E,F Coeficients
15     s = np.polyval(x,t) #SIGNAL
16     sd = np.polyval(np.diag(np.diag([5,4,3,2,1])*x[0:5]),t)#1 ORDER
           SIGNAL DERIVATIVE
17     sdd = np.polyval(np.diag(np.diag([20,12,6,2])*x[0:4]),t)#2 ORDER
           SIGNAL DERIVATIVE
18     return s[0],sd,sdd
```

Listing 5.6: Código de generación de trayectorias

```
1 def scale( inp,  in_min,  in_max,  out_min,  out_max):
2     return int(((inp - in_min) * ((out_max - out_min) / (in_max -
          in_min))) + out_min)
```

Listing 5.7: Código de escalización

```
1 def trajectory_tracking(x_i,x_d,y_i,y_d,phi_i,phi_d,rc,
2 rc2,automatic_mode=True):
3     global EXEC_THREAD
4     #%%%%%%%%%% C O N S T A N T S %%%%%%
5     R  = 0.031;# % 3 cm
6     L = 0.161;# % 11 cm
7     max_motor_vel = 34.5 #34.5 rad/s or 330 RPM ( read datasheet)
8     max_PWM = 12000 #rad/s or 330 RPM ( read datasheet)
9     address = 0x80
```

```
    #%%%%%%%%% S U B R U T I N E %%%%%%%%
    while curr_time<=calculated_time and EXEC_THREAD:
        ########################################
        #START TIME
        ########################################
        start_time = time.time()
        ########################################
        #CODE EXECUTION CALCULATION  (DELTA TIME)
        ########################################
        curr_time+=delta_t
        #print("current time: ", curr_time)

        if curr_time>=calculated_time:
            #time.sleep(0.15)
            return True
        #time.sleep(0.01)

        '''
        INPUT (TRAJECTORY GENERATION) POS, VEL , ACCEL
        '''
        x_req,x_dot_req,x_dot_dot_req =
            getCoef(t=curr_time,i_v=x_i,f_v=x_d,i_t=0,f_t=calculated_time)
        y_req,y_dot_req,y_dot_dot_req =
            getCoef(t=curr_time,i_v=y_i,f_v=y_d,i_t=0,f_t=calculated_time)
        phi_req,phi_dot_req,phi_dot_dot_req =
            getCoef(t=curr_time,i_v=phi_i,f_v=phi_d,i_t=0,
        f_t=calculated_time)
        '''
        REQUIRED GLOBAL VELS
        '''
        globals_vels_required =
            np.array([[x_dot_req],[y_dot_req],[phi_dot_req]])
        '''
        DIRECT JACOBIAN MATRIX
        '''
        J = np.array([
                [-np.sin(phi_req), np.cos(phi_req), L],
                [-np.cos((np.pi/6) + phi_req), -np.sin((np.pi/6) +
                    phi_req), L],
                [np.cos(phi_req - (np.pi/6)), np.sin(phi_req -
                    (np.pi/6)), L]
            ])
        '''
        INVERSE CINEMATIC MODEL / CALCULATE REQUIRED WHEELS SPEED
        '''
        MCI = ((1/R)*J)@globals_vels_required

        '''
        INVERSE CINCEMATIC MODEL WHEEL REQUIERED ANGULAR VELS
        '''
        q1_req = float(MCI[0][0])
        q2_req = float(MCI[1][0])
        q3_req = float(MCI[2][0])
```

```
57
58          '''
59          OUT READ
60          '''
61          try:
62              '''
63              SCALE CONTROL VARIABLE (CONTROL EFFORT)
64              '''
65              SC1 = scale(q1_req,0,max_motor_vel,0,max_PWM)
66              SC2 = scale(q2_req, 0,max_motor_vel,0,max_PWM)
67              SC3 = scale(q3_req, 0,max_motor_vel,0,max_PWM)
68
69              '''
70              INJECT CONTROL AUTOMATIC CONTROL, ERROR CALCULATED FOR
                    ROBOCLAWS (INTERNAL PID CONTROL)
71              '''
72
73              rc.SpeedM2(address,SC1) #q1
74              rc2.SpeedM2(address,SC2) #q2
75              rc2.SpeedM1(address,SC3) #q3
76          except AttributeError:
77              #print("no roboclaw!")
78              pass
79
80          ######################################
81          #END TIME CALCULATION
82          ######################################
83          end_time = time.time()
84          delta_t = end_time - start_time
85  #%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing 5.8: Código de seguimiento de trayectorias del robot

Bibliografía

[1] A. J. Dávila Aarón, "Modelo cinemático y cinetico de un robot omnidireccional," 2007. [Online]. Available: http://hdl.handle.net/1992/9609

[2] H. Taheri and Z. C. Xia, "Slam; definition and evolution," *Engineering Applications of Artificial Intelligence*, vol. 97, p. 104032, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197620303092

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[4] R. Mur-Artal, J. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[5] G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

[6] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[7] G. Younes, D. Asmar, E. Shammas, and J. Zelek, "Keyframe-based monocular slam: design, survey, and future directions," *Robotics and Autonomous Systems*, vol. 98, pp. 67–88, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889017300647

[8] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," in *Robust Reconstruction of Indoor Scenes*, 06 2015.

[9] J. Park, Q.-Y. Zhou, and V. Koltun, "Colored point cloud registration revisited," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 143–152.

[10] D. Sharafutdinov, M. Griguletskii, P. Kopanev, M. Kurenkov, G. Ferrer, A. Burkov, A. Gonnochenko, and D. Tsetserukou, "Comparison of modern open-source visual SLAM approaches," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 107, no. 3, 2023. [Online]. Available: www.scopus.com

[11] H. Kress-Gazit, K. Eder, G. Hoffman, H. Admoni, B. Argall, R. Ehlers, C. Heckman, N. Jansen, R. Knepper, J. Křetínský, S. Levy-Tzedek, J. Li, T. Murphey, L. Riek, and D. Sadigh, "Formalizing and guaranteeing human-robot interaction," *Commun. ACM*, vol. 64, no. 9, p. 78–84, aug 2021. [Online]. Available: https://doi.org/10.1145/3433637

[12] D. Xu, F. Li, and H. Wei, "3D point cloud plane segmentation method based on RANSAC and support vector machine," in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2019, pp. 943–948.

[13] S. Anderson and T. D. Barfoot, "RANSAC for motion-distorted 3D visual sensors," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 2093–2099.

[14] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3212–3217.

[15] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 2001, pp. 145–152.

[16] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[17] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2010.

[18] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 2005.

[19] H. Zhang, Y. Liu, and Z. Wang, "Design and control of a three-wheeled omnidirectional mobile robot," *IEEE Access*, vol. 9, pp. 125 789–125 798, 2021.

[20] A. S. Martins and R. R. de Azevedo, "Kinematic modeling and control of a three-wheel omnidirectional robot for indoor navigation," *Robotics and Autonomous Systems*, vol. 133, p. 103631, 2020.

[21] J. Kim, D. Lee, and M. Park, "Omnidirectional mobile robot with three omni-wheels for human-robot interaction," *Sensors*, vol. 22, no. 15, p. 5689, 2022.

[22] G. Bermudez, "Robots móviles. teoría, aplicaciones y experiencias," *Tecnura*, vol. 5, no. 10, pp. 6–17, 2002.

[23] R. Raudmäe, S. Schumann, V. Vunder, M. Oidekivi, M. K. Nigol, R. Valner, H. Masnavi, A. K. Singh, A. Aabloo, and K. Kruusamäe, "Robotont – open-source and ros-supported omnidirectional mobile robot for education and research," *HardwareX*, vol. 14, p. e00436, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2468067223000433

[24] J. Li, Y. Wang, and X. Chen, "Kinematic analysis and control of a three-wheel omnidirectional mobile robot for autonomous indoor navigation," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 7, pp. 5501–5510, 2019.

[25] M. Savaee, H. D. Taghirad, and A. Yousefi-Koma, "Effective kinematic parameter estimation for holonomic mobile robots," *Journal of Intelligent & Robotic Systems*, vol. 100, pp. 81–96, 2020.

[26] J. Palacín, E. Rubies, and E. Clotet, "Nonparametric calibration of the inverse kinematic matrix of a three-wheeled omnidirectional mobile robot using genetic algorithms," *Applied Sciences*, vol. 13, no. 2, p. 1053, 2023.

[27] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, p. 381 – 395, 1981. [Online]. Available: http://doi.org/10.1145/358669.358692

[28] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008. [Online]. Available: https://doi.org/10.1016/j.cviu.2007.09.014

[29] P. Azad, T. Asfour, and R. Dillmann, "Combining Harris interest points and the SIFT descriptor for fast scale-invariant object recognition," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 4275–4280.

[30] M. He, Y. Dai, J. Zhang, and L. Bai, "Rotation invariant feature descriptor integrating HAVA and RIFT," in *APSIPA ASC 2010 - Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2010, pp. 935–938. [Online]. Available: www.scopus.com

[31] H. Mingliang and X. Shubin, "Study of improving the stability of SUSAN corner detection algorithm," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 11, 2010, pp. V11–627–V11–630.

[32] G. Arbeiter, S. Fuchs, R. Bormann, J. Fischer, and A. Verl, "Evaluation of 3D feature descriptors for classification of surface geometries in point clouds," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1644–1650.

[33] S. Salti, F. Tombari, and L. Di Stefano, "Shot: Unique signatures of histograms for surface and texture description," *Computer Vision and Image Understanding*, vol. 125, 08 2014.

[34] Z. Zhou, D. Huang, and Z. Liu, "Point cloud registration algorithm based on 3D shape context features," *Journal of Applied Optics*, vol. 44, no. 2, pp. 330–336, 2023. [Online]. Available: www.scopus.com

[35] F. Steinbrücker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense rgb-d images," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011, pp. 719–722.

[36] X. Zhang, H. Yu, and Y. Zhuang, "A robust RGB-D visual odometry with moving object detection in dynamic indoor scenes," *IET Cyber-systems and Robotics*, vol. 5, no. 1, 2023. [Online]. Available: www.scopus.com