ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“IMPLEMENTACIÓN DE TÉCNICA DE MAPEO Y LOCALIZACIÓN SIMULTÁNEO (SLAM) EN VEHÍCULO AUTÓNOMO”

INFORME DE PROYECTO INTEGRADOR

Previo la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN

Presentado por:

DAVID ISAÍAS SOQUE

MARÍA GABRIELA GUERRA PINTADO

GUAYAQUIL - ECUADOR

Año: 2020

DEDICATORIA

Dedico el presente trabajo de investigación académica a mis queridos padres por el gran esfuerzo que día a día han realizado por brindarme los recursos necesarios para recibir una educación superior de excelencia y por haber confiado en mi capacidad y esfuerzo, a pesar de las dificultades que hemos enfrentado, pero con la ayuda de Dios hemos salido adelante. A mis profesores por impartir sus conocimientos y consejos durante todos estos años que fui estudiante de esta prestigiosa institución académica llamada ESPOL y a todos mis compañeros por brindarme su amistad, cariño, su conocimiento y sobre todo el respeto mutuo que mantuvimos.

David Isaías Soque León

DEDICATORIA

Este proyecto académico va dedicado a mi familia que son mi pilar y mi refugio predilecto, que por medio de su apoyo y ejemplo me han permitido iniciar y culminar mis estudios universitarios. También dedico este trabajo a mi fiel e incondicional consejera, mi madre del cielo, María.

María Gabriela Guerra Pintado

AGRADECIMIENTOS

En primer lugar, doy gracias a Dios por bendecirme con la oportunidad de estudiar en esta maravillosa institución y cuidarme durante mi etapa universitaria donde atravesé momentos alegres y difíciles, sin embargo, no me rendí y me esforcé mucho para llegar a la meta. También agradezco a mis padres y a todas las personas que fueron parte de este proceso que me enseñó que sin sacrificio no hay éxito. Finalmente agradezco a el compañero Steven Silva estudiante de la carrera de Mecatrónica en ESPOL, por su apoyo incondicional en el desarrollo de este interesante proyecto.

David Isaías Soque León

AGRADECIMIENTOS

Agradezco a Dios por permitirme vivir esta hermosa etapa de mi vida académica. A mis padres y hermanos que con su apoyo y consejos me han impulsado a dar lo mejor de mí. A mis amigos y ahora colegas que gracias a su compañía hicieron de la universidad un lugar ameno y llevadero. A mis maestros que con su ejemplo y consejos han inspirado en mis grandes metas. A mi querida Alma mater ESPOL por brindarme sus instalaciones para mi preparación como politécnica y sin duda alguna a nuestro compañero de investigación Steven Silva por su valioso tiempo, dedicación y compromiso que ha tenido con este proyecto.

María Gabriela Guerra Pintado

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *David Isaías Soque* y *María Gabriela Guerra Pintado* damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

David Isaías Soque

María Gabriela Guerra

EVALUADORES

M.Sc. Dennys Dick Cortez Alvarez

PROFESOR DE LA MATERIA

PhD. Douglas Antonio Plaza Guingla

PROFESOR TUTOR

RESUMEN

En el presente trabajo tiene como objetivo la implementación de una técnica SLAM, donde la importancia de implementar una técnica de navegación en un robot autónomo se basa en establecer de forma referencial una ruta libre de obstáculos para el traslado del robot a través de un entorno físico, desde un punto específico hasta su lugar de destino. El método de investigación utilizado para el desarrollo del proyecto, desde un enfoque cuantitativo es la investigación experimental, donde se trabajó con algunas variables que intervinieron en el desarrollo de la navegación autónoma. La plataforma Arlo robot con todos sus componentes, el sensor RPLIDAR A1, los encoders de cuadratura, el controlador ESP32, el driver DBH-10, la mini PC y los softwares ROS, ONSHAPE, GAZEBO, RVIZ son los recursos de hardware y software utilizados en la simulación y experimentación de la navegación autónoma. El diseño de la arquitectura del sistema, los paquetes de ROS, las especificaciones técnicas de los equipos, costos y descripción del procedimiento se encuentran en diagramas, tablas y figuras. Los resultados reflejados a través de figuras y análisis mostraron que la técnica SLAM Cartographer en conjunto con los paquetes de ROS, ejecutaron la navegación autónoma de manera aceptable, a pesar de ruidos y perturbaciones externas en sensores y equipos. El algoritmo SLAM Cartographer genera mapas de alta calidad, pero necesita integrar paquetes de ROS para desarrollar, la localización, planificación y seguimiento de trayectorias con evasión de obstáculos; aspectos que hacen posible la navegación autónoma.

Palabras Clave: SLAM, navegación autónoma, robot móvil diferencial, mapeo, planificación, localización, sensor LIDAR.

.

ABSTRACT

This project aims to implement a SLAM technique, where the importance of implementing a navigation technique in an autonomous robot is focused on establishing a reference path free of obstacles for the movement of the robot through a physical environment. , from a specific point to it destination. The research method used for the development of the project, since a quantitative approach, is experimental research, where some variables that intervened in the development of autonomous navigation were worked on. The Arlo robot platform with all its components, the RPLIDAR A1 sensor, the quadrature encoders, the ESP32 controller, the DBH-10 driver, the mini PC and the ROS, ONSHAPE, GAZEBO, RVIZ software's integrate the hardware and software resources used in the simulation and experimentation of autonomous navigation. The design of the system architecture, ROS packages, technical specifications of the equipment, costs and description of the procedure are found in diagrams, tables, and figures. The results reflected through figures and analysis verify that the SLAM Cartographer technique in conjunction with the ROS packages, performed the autonomous navigation in an acceptable way, despite noise and external disturbances in sensors and equipment. The SLAM Cartographer algorithm generates high quality maps, but it needs to integrate ROS packages to develop, locate, plan and track trajectories with obstacle avoidance; aspects that make autonomous navigation possible.

Keywords: SLAM, autonomous navigation, differential mobile robot, mapping, planning, location, LIDAR sensor

ÍNDICE GENERAL

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
SLAM	Simultaneous Localization and Mapping
VAG	Vehículos Autónomos Guiados
ROS	Robot Operating
RMA	Robot Móvil Autónomo
IFR	International Federation of Robotics
LIDAR	Light Detection and Ranging
EFK	Filtro Extendido de Kalman
EIF	Filtro Extendido de Información
LSD-SLAM	SLAM Monocular Directo a Gran Escala
DSO-SLAM	Odometría Dispersa Directa
DPPTAM	Seguimiento y Mapeo Paralelo Denso Por Partes
RTAB SLAM	Mapeo Basado en Apariencia en Tiempo Real SLAM
ORB-SLAM	Orientado Rápido y Girado Breve SLAM
S-PTAM	Seguimiento y Mapeo Paralelo

SIMBOLOGÍA

Δ	letra griega Delta, mayúscula Indica cambio o variación
ϕr	representa la orientación del robot
m^2	metros cuadrados
mm	milímetros
g	gramos
V	voltios
ma	miliamperios
bps	bits por segundo
°C	grados centígrados
m	metros
Hz	hertzios
cm	centímetros
VDC	voltaje de corriente continúa
MB	megabyte
PWM	Modulación por ancho de pulso

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

La navegación autónoma para un robot móvil requiere la aplicación de tres tareas de suma importancia que se encuentran relacionadas entre sí; estas son el mapeo, la localización, y la planificación de la trayectoria. La tarea de localización ha sido un problema de estudio persistente dentro de la robótica y a partir de esto surgió la técnica de Mapeo y Localización Simultánea (SLAM) que junta dos de estas tareas en una sola. La navegación autónoma usando SLAM permite a los robots ser capaces de alcanzar un punto objetivo de forma segura; es decir escogiendo la trayectoria que lo dirija a su meta a través de un camino óptimo libre de obstáculos cuando el entorno por el que navega es dinámico y desconocido. Se denota la importancia de poseer siempre un mapa del lugar para lograr referenciar al robot dentro del mismo y volver a actualizar en el mapa datos sobre nuevas posibles características añadidas al entorno que permitan trazar una correcta planificación. En los últimos años se ha evidenciado nuevas aplicaciones donde se involucran a robots en tareas colaborativas, especialmente en procesos de manufactura, pero también ha surgido como una necesidad su aplicación en otras áreas útiles por su versatilidad en ejecutar diferentes tareas. “Según la Federación Internacional de Robótica (IFR), el mercado de AMR en la automatización está en auge, con un aumento esperado de 110.000 en 2019 a más de 700.000 en 2022. Esto representa una tasa compuesta anual del 45% en la industria manufacturera y del 60% en entornos no relacionados con la fabricación, como el comercio electrónico y los hospitales” (Mir, 2019). En respuesta a estas nuevas tecnologías que apuntan a mejorar la eficiencia dentro de las industrias, se busca dar pasos hacia la solución de la problemática sobre localización y mapeo, implementando una técnica SLAM en un robot móvil con manejo diferencial dentro de espacios interiores y de entorno dinámico. Para llevar a cabo este objetivo se requiere hacer un estudio del estado del arte de las diferentes técnicas SLAM existentes, seleccionar la más conveniente según los requerimientos disponibles. Se deberá también definir una posible estructura de software para correr el programa con la técnica SLAM y de hardware para realizar una implementación física de la técnica. El documento del presente proyecto se

encuentra estructurado en 4 capítulos. En el capítulo 1 se describe el marco teórico o estudio del arte, en el capítulo 2 se redacta la metodología a seguir para obtener los objetivos planteados, en el capítulo 3 se detallan los resultados tanto de la simulación como de la implementación, finalmente en el capítulo 4 se presentan las conclusiones y recomendaciones.

1.1. Descripción del problema

Existen dos tipos de robots móviles; los Vehículos Autónomos Guiados (VAG), por ejemplo, los seguidores de línea, y los Robots Móviles Autónomos (RMA), estos últimos se caracterizan por trazar su propia ruta de navegación e incluso modificarla de acuerdo con el entorno en que se mueven. Cuando se efectúa la tarea de navegación no siempre se conoce el medio por donde se desplazará el robot por lo que es necesario adquirir datos del lugar a través de sensores para estimar un mapa del entorno y lograr la localización del robot. Existen algunos tipos de técnicas de localización y mapeo simultáneas, pero no todas se ajustan a las necesidades y limitaciones del cliente.

La carencia de autonomía de un robot móvil debido a un problema de posicionamiento limita las posibles aplicaciones que se podrían implementar cuando se tiene un entorno dinámico; por ejemplo, la desinfección en espacios cerrados de difícil acceso o también en ambientes donde exista la posibilidad de contagio de alguna enfermedad peligrosa, entre otras aplicaciones que en la actualidad son muy demandadas.

1.2. Objetivos

1.2.1. Objetivo General

Implementar técnica de Mapeo y Localización (SLAM) aplicadas a un robot móvil con manejo diferencial en espacios interiores y de entornos dinámicos.

1.2.2. Objetivos Específicos

- Estudiar el estado de arte de las técnicas de SLAM implementadas para robots móviles.
- Definir una posible estructura de software para correr el programa con la técnica SLAM y de hardware para hacer posible una implementación física de la técnica.
- Implementar una técnica de navegación usando SLAM en un robot móvil.

1.3. Marco teórico

1.3.1. Navegación autónoma

La navegación en robótica es el proceso mediante el cual se obtiene un recorrido óptimo que será ejecutado por el robot. La navegación autónoma tiene entre sus

prioridades la evasión de obstáculos, al mismo tiempo que se permite al robot desplazarse en entornos dinámicos y desconocidos de forma segura. Este último aspecto es posible mejorar al trabajar más en el sistema de percepción del robot, las restricciones en su movilidad debido al control de sus grados de libertad y la capacidad computacional del robot. (Savkin, 2015). Los componentes de la navegación autónoma son:

- Localización
- Mapeo
- planificador de trayectorias

Localización de robot móvil

La localización de un robot móvil durante un lapso dentro de un entorno dinámico ha sido motivo de estudio y desarrollo de las técnicas SLAM para la navegación autónoma, para lo cual es importante conocer la posición del robot móvil de manera relativa y absoluta, es decir, respecto al comienzo del movimiento y al sistema de coordenadas. (Martínez Q, 2018, pág. 15). De forma general la localización es una función integrada y que se adapta con diferentes algoritmos SLAM. (Chan S, 2018, pág. 1263)

Localización global

La localización global o también llamada localización absoluta es aquella que presenta todas las trayectorias de desplazamientos generadas incluyendo el objetivo sobre un mapa, el cuál es presentado de manera más amplia. (Fernández A, 2019, pág. 61) Es muy útil para ubicar al robot a un punto de referencia fijo, a pesar de que no sea muy utilizado para la localización en ambientes interiores debido a que la señal de GPS es débil y otras ocasiones no es detectada por el receptor en esos ambientes. (Martínez Q, 2018, pág. 22)

Localización local

La localización local o también llamada localización relativa es aquella que ilustra sobre el mapa las trayectorias de desplazamientos que se van generando en ese momento; el cual, permite estar atento y reaccionar con anticipación ante cualquier cambio sobre el mapa. Además, se define la localización local con respecto al inicio del movimiento, es decir, se define la posición en base a

posiciones anteriores del robot. (Fernández A, 2019, pág. 61). Los errores de precisión de la ubicación del robot son menores porque usan el método de estimación de la posición (odometría); además esta localización permite posicionar al robot móvil de forma relativa con el ambiente de navegación. (Martínez Q, 2018, pág. 22)

Mapeo

Proceso que permite construir una representación del escenario por el cual el robot navega. En algunos casos el robot tiene la capacidad para construir el mapa, otras veces es necesario cargarlo a la programación del robot. Se emplean mapas de tipo métricos en lo cuales se ubican elementos en el mapa respecto a un sistema de referencia (Reinoso García, 2016).

Existen mapas métricos de ocupación donde se localizan zonas libres u ocupadas.

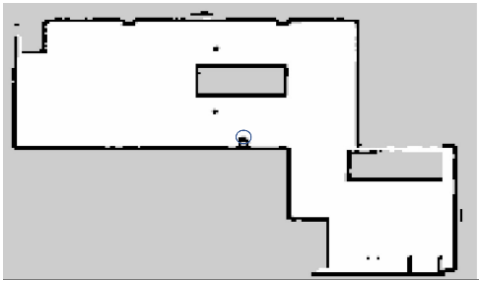


Figura 1.1 Mapas métricos de ocupación

Otro tipo de mapas usados para la navegación son los mapas basados en marcas que almacenan la posición de un conjunto de elementos (Reinoso García, 2016).

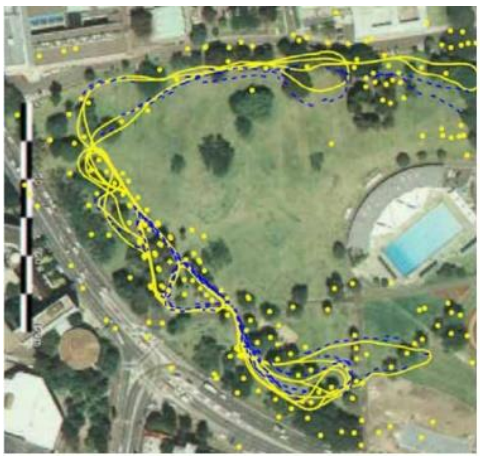


Figura 1.2 Mapas basados en marcas

[Tomado de (Reinoso García, 2016)]

Planificación

Crea una secuencia estructurada de puntos objetivos que deberán ser alcanzadas por el robot. Esta secuencia se calcula con la ayuda de un mapa del entorno, descripción de la tarea que se realizará y datos sobre que estrategia se usará (Anónimo).

Odometría

La estimación de la posición de un robot móvil a lo largo de una trayectoria se basa en el método de estimación llamado odometría, el cual usa la información de la velocidad angular de las ruedas medidas por encoders, integrados en el robot móvil, permitiendo la estimación en el cambio de posición del robot durante un lapso y realizando la integración del camino con el uso de agrupación de desplazamientos. (Martínez Q, 2018, pág. 13)

La expresión matemática que representa la odometría para un robot móvil de desplazamiento diferencial se encuentra expresada en la ecuación 1.1

$$\begin{pmatrix} x(t+\Delta t) \\ y(t+\Delta t) \\ z(t+\Delta t) \end{pmatrix} = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} \qquad (1.1)$$

Para hacer posible la localización es necesario tener fuentes de datos como por ejemplo la posición en base a la odometría, el registro de los sensores, un mapa del entorno.

Robot móvil

En general los robots móviles se definen como “Un vehículo con propulsión autónoma y movimiento (re)programado a través del control automático para realizar una tarea específica” (Lazea Gh, 2001, pág. 1).

Una característica esencial para un robot móvil es que tenga integrado el sistema de locomoción permitiendo navegar en un ambiente determinado. Además, los robots móviles son clasificados en base al control de sus grados de libertad; se los denomina holonómico y no holonómico. Del sistema no holonómico se derivan en gran parte los modelos cinemáticos. (Caverzasi A, 2014, pág. 786)

1.3.2. Receptores de robot móvil

En los sistemas de mapeo y localización simultánea para robots móviles el uso de sensores para la adquisición de datos es muy importante, ya que estos datos permiten conocer la localización del robot móvil en tiempo real y a su vez obtener un mapeo de la trayectoria que realizó el robot, puede ser visualizado a través de un software, en este caso ROS (Robot Operating System).

Sensor Láser

Uno de los sensores más destacado en la navegación autónoma de un robot móvil en un ambiente interior es el sensor LIDAR, debido a la gran capacidad de detección de objetos en tiempo real y su alta precisión en la toma de datos, lo que conlleva a que sea motivo exhaustivo de estudio en los últimos años (Filipenko & Afanasyev, 2018, pág. 400). La detección del sensor LIDAR se basa en la emisión de haces de rayos de luz laser infrarroja, la cual es reflejada en el objeto detectado y receptado por el lente infrarrojo del sensor.

IMU

Los robots móviles en gran parte contienen una Unidad de Medición Inercial conocida por la sigla IMU, con la finalidad de conocer y obtener la localización del robot móvil usando el Sistema de Navegación Inercial (INS). La Unidad de Medición Inercial incluye un conjunto de dispositivos que miden la velocidad angular, orientación y aceleración, los cuales son: giroscopio, magnetómetro y acelerómetro (Filipenko & Afanasyev, 2018, pág. 400).

Sensor de profundidad

Dispositivo que tiene la capacidad de obtener la información de la profundidad de un entorno, en el cual se genera un mapa estimado del ambiente. Este sensor hace referencia a cámaras monoculares, estéreos o RGB-D que permiten conocer la información de la profundidad aplicando algoritmos de SLAM visual (Kuang H, 2018, pág. 350). Captan la profundidad a partir de dos elementos que vienen integrados en la mayoría de las cámaras Kinects. La primera es el proyector de luz infrarroja que refleja una matriz que contiene rayos de luces infrarrojas sobre el entorno y el segundo es el sensor de luz infrarroja que se encarga de receptar este paquete de luces infrarrojas que rebotaron en los objetos (Barchesi S, 2012).

Encoders

Es un codificador rotatorio encargado de arrojar magnitudes físicas con respecto a la velocidad, posición, orientación. Este tipo de sensor que se encuentra ubicado en algún dispositivo giratorio se encarga de convertir el movimiento del dispositivo en señales eléctricas que son captadas por una tarjeta de control (Wang J, 2018, pág. 501).

1.3.3. Cinemática del Robot móvil diferencial

La cinemática del robot móvil diferencial analiza los movimientos del robot sin considerar la fuente que provoca el movimiento, es decir, el análisis de fuerzas que intervienen es omitida.

La plataforma Arlo robot para la navegación en entornos interiores consta de dos ruedas de tracción, donde se acoplan los motores DC que permitirán el desplazamiento diferencial, produciendo movimiento traslacional y rotacional; además posee otras dos ruedas de estabilización que sostienen la base del robot y mantienen su equilibrio. (Valencia J, 2009, pág. 191)

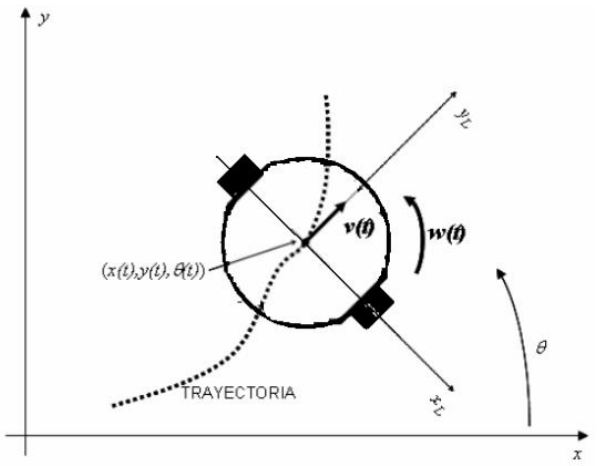


Figura 1.3 Localización del robot móvil en un plano xy (Valencia J, 2009)

Respecto a un sistema de referencia global, dado por los ejes x,y que se observa en la figura 1.3, el punto central (x(t), y(t)), θ(t)) donde gira la plataforma robótica está referida al sistema de referencia mencionado. Las siguientes ecuaciones cinemáticas expresan el movimiento del robot en un plano 2D con respecto al tiempo:

$$\begin{aligned} \dot{x} &= v(t)cos\big(\theta(t)\big) \\ \dot{y} &= v(t)sen\big(\theta(t)\big) \\ \dot{\theta} &= w(t) \end{aligned} \tag{1.2}$$

Integrando las ecuaciones cinemáticas 1.2 dentro de un periodo determinado Δt se obtienen la posición y orientación de la plataforma del robot móvil. (Valencia J, 2009, pág. 192) Las expresiones matemáticas (1.3) son:

$$\begin{aligned} x(t) &= x(t_0) + \int_{\Delta t} v(t)cos\left(\theta(t)\right)dt \\ y(t) &= y(t_0) + \int_{\Delta t} v(t)sen\big(\theta(t)\big)dt \\ \theta(t) &= \theta(t_0) + \int_{\Delta t} w(t)dt \end{aligned} \tag{1.3}$$

Si se discretiza las ecuaciones haciendo que el Δt, se aproxime a 0, permitiendo que Δx, Δy, $\Delta \theta$ que son los desplazamientos diferenciales reemplacen a las expresiones anteriores, donde la frecuencia de muestreo se mantiene. Las siguientes ecuaciones (1.4) en diferencia permitirán estimar el cálculo de la posición y orientación de la plataforma del robot móvil. (Valencia J, 2009, pág. 192)

$$\begin{aligned} x_k &= x_{k-1} + \Delta x_k \\ y_k &= y_{k-1} + \Delta y_k \\ \theta_k &= \theta_{k-1} + \Delta \theta_k \end{aligned} \tag{1.4}$$

1.3.4. Métodos probabilísticos

En navegación los métodos probabilísticos cumplen el papel de presentar una distribución de probabilidad de la posición del robot además del mapa del entorno de navegación. (Andrade & Llofriu)

Filtro Extendido de Kalman

De acuerdo con la explicación descrita en el libro: Simultaneous Localization and Mapping Exactly Sparce Information Filter. (Zhan, 2009, págs. 5-6) para la obtención de la distribución de probabilidad conjunta del robot.

El estado del robot es denotado por $X_r = [x_r, y_r, \phi r]^T$, donde (x_r, y_r) equivale a la posición y ϕ_r representa la orientación del robot. Las características se denotan por $X_m = [x_1, y_1, x_2, y_2, \dots]^T$, donde (x_j, y_j) representa la posición de una característica j por lo tanto el vector de estado es $X = [X_r^T, x_m^T]^T$

El modelo cinético del robot que relaciona la postura de este con los pasos k y $k+1, X_r(k)$ y $X_r(k+1)$ está dada por

$$X_r(K+I) = f\big(X_r(k), u(k)\big) + w_n(k) \tag{1.5}$$

Donde $u(k)$ hace referencia a los datos de medición de control en el paso del tiempo $k, w(k)$ es el ruido del proceso asumido que es blanco gaussiano con media cero y una covarianza Q_r, y la función f depende del modelo del robot.

La ecuación de estado de transición para la población de características es

$$x_m(k+1) = X_m(k) \tag{1.6}$$

las características se asumen estacionarias.

Por lo tanto, el modelo del estado de transición para todo el sistema puede ser escrito como.

$$X(k+1) = F\big(X, (k), u(k)\big) + w(k) \tag{1.7}$$

Es decir

$$\begin{bmatrix} X_r & (k+1) \\ X_m & (k+1) \end{bmatrix} = \begin{bmatrix} f\big(X_r(k), u(k)\big) \\ x_m(k) \end{bmatrix} + \begin{bmatrix} w_r(k) \\ 0 \end{bmatrix} \tag{1.8}$$

Donde la matriz de covarianza de $w(k)$ es denotada como Q.

Con la ayuda de las expresiones expuestas, es posible obtener la distribución de probabilidad de la ubicación tanto del robot, como de las características alcanzada en la observación con los elementos receptores del robot.

El Filtro Extendido de Kalman supone al ruido w(k) y v(k) como gaussiano blanco y sirve para resolver el problema de probabilidad conjunta como una estimación no lineal, donde “la media del vector de estado y la matriz de covarianza asociada son suficientes para describir la distribución de probabilidad gaussiana conjunta” (Zhan, 2009, pág. 6).

$$P = \begin{bmatrix} P_{rr} & P_{rm} \\ P_{rm}^{T} & P_{mm} \end{bmatrix} \tag{1.9}$$

P es la matriz de covarianza, P_{rr} es la submatriz correspondiente a la estimación de pose del robot, P_{mm} es la submatriz de la estimación del mapa y P_{rm} es la matriz de correlación. (Zhan, 2009, pág. 7).

El filtro trabaja en dos etapas; la de predicción y la actualización (F., 2017), Ocupando mayor capacidad computacional en la etapa de actualización, donde generalmente se analizan gran cantidad de puntos de referencias y limitaciones. El número de referencias debe ser menor a 1000 (Thrun, 2003).

Filtro Extendido de Información (EIF)

Al EIF se lo conoce como el dual matemático del EKF porque se plantean los mismos supuestos, sobre ruido gaussiano blanco, pero ahora la matriz de covarianza P es invertida y se la llama matriz de información que junto con el llamado vector de información precisan la distribución de probabilidad gaussiana de la ubicación del robot y las características (Zhan, 2009, pág. 7).

$$I = P^{-1} \tag{1.10}$$
$$i = I\hat{x}$$

I representa la matriz de información, i hace referencia al vector de información.

A diferencia del EKF en este filtro la etapa de predicción es conlleva un mayor esfuerzo computacional.

Filtro de partículas

En este tipo de filtro al igual que el EKF se estable los pasos de predicción y actualización para ellos “se realiza un muestreo recursivo de Monte Carlo para

estimar las densidades de probabilidad, ahora representadas por partículas" (Zhan, 2009, pág. 8). En cada partícula está representado un estado determinado del sistema compuesto por el estado del robot y un mapa. Para manifestar un movimiento del robot, en cada paso se modifica la función de densidad y en la actualización se renueva los datos recibidos por la función de densidad con los últimos percibidos por el sensor. (Pineda-Torres, 2019).

1.3.5. SLAM

La técnica de navegación SLAM permite generar un mapeo del medio por el cual navega el robot al mismo tiempo que se estima su localización dentro del mismo, sin haber recibido previamente datos de las características del lugar. EL principal problema del SLAM consiste en la adquisición de datos de la localización relativa entre el robot y las características próximas del entorno en que este se mueve.

1.3.6. Métodos de SLAM con cámaras monoculares y binoculares

SLAM visual

Existen dos tipos de SLAM Visual, el basado en características y el directo, ambos enfocados en la construcción del mapa mediante las particularidades observadas en el entorno.

El SLAM Visual basado en características utiliza las coordenadas de la imagen de un conjunto de correspondencias puntuales sobresalientes. Mientras que el directo utiliza la intensidad de valores de los pixeles sin procesar para estimar el mapa del entorno y el movimiento de la cámara. (Concha & Civera , 2015)

Detección de bucle cerrado

Proceso mediante el cual el robot reconoce si la localización en que se encuentra ya ha sido visitada antes, caso contrario la almacena en una base de datos. Es ejecutado mediante comparación de datos.

Al adquirir datos de localización es importante considerar que el tiempo para el cálculo de comparación, no supere el tiempo de adquisición, lo que suele suceder en largas trayectorias y afecta el mapeo en tiempo real.

SLAM Monocular Directo a Gran Escala (LSD-SLAM)

SLAM monocular directo a gran escala, realiza un filtrado probabilístico para la estimación de profundidad del mapa. Contiene la optimización de gráfico de pose y cierre de lazo para grandes entornos de navegación. (Concha & Civera , 2015)

Odometría Dispersa Directa SLAM (DSO-SLAM)

Trabaja con un modelo directo de probabilidad; (minimizando el error fotométrico) y con una coherente optimización conjunta de parámetros como la profundidad inversa desde un marco de referencias, que es posible lograr a través del muestro equitativo de pixeles en las imágenes. (J. Engel, 2018)

Para su utilización necesita de un contenedor para integrarlo con ROS (Robot Operating System). Entre las ventajas que posee están que arroja un denso mapa y es un sistema robusto en términos de seguimiento de poses. Por otro lado, la falta de un bucle para cerrar la creación del mapa hace que se generen datos redundantes que provocan ruidos en la gráfica. (Filipenko & Afanasyev, 2018)

Seguimiento y Mapeo Paralelo Denso Por Partes (DPPTAM)

Proporciona una estimación directa de pose, asume que todas las regiones que poseen el mismo color están dentro del mismo plano. (Afanasyev, 2018).

Es un algoritmo para la estimación de áreas planas, basado en la información de una segmentación superpíxel y el mapa semidenso de áreas texturizadas. (Concha & Civera , 2015)

Mapeo Basado en Apariencia en Tiempo Real SLAM (RTAB SLAM)

Técnica SLAM basado en características que detecta bucles cerrados a partir de las imágenes de la cámara. Posee detección de lazo cerrado, tiene una buena integración con ROS. (Filipenko & Afanasyev, 2018).

El objetivo de esta técnica es realizar SLAM en tiempo real sin importar las largas distancias, ni el tiempo que conlleva realizar la tarea. Dentro del proceso se trabaja en un tratamiento de memoria para limitar el número de localizaciones que conforman una detección de lazo cerrado. (López Torres, 2016)

Orientado Rápido y Girado Breve SLAM (ORB-SLAM)

Sistema basado en características usado tantos en ambientes interiores como en exteriores. Posee un sistema de reacción robusta a movimientos desordenados, además admite amplio cierre de bucle de referencia y de inicialización automática. (Mur-Artal, Montiel, & Tardós, 2015)

Seguimiento y Mapeo Paralelo (S-PTAM)

Sistema basado en características con cierre de bucle. En su desarrollo, divide el proceso en dos partes; el seguimiento de la cámara y la optimización del mapeo. Cada tarea se ejecuta independientemente, pero de forma paralela y comparten solo el mapa. El hilo de seguimiento actualiza los datos de los sensores, crea nuevos puntos y propone la pose de la cámara, mientras el hilo de mapeo refina los puntos de referencias cercanos que forman el mapa. (Pire, Fischer, Civera , De Cristóforis , & Jacobo Berlles, 2015)

1.3.7. Métodos SLAM con sensor láser

En los últimos años se han desarrollado varios métodos SLAM basados en el sensor LIDAR, donde ha sido llamativa la aplicación de estos métodos en robots móviles, los cuales usan OGM (mapeo de cuadrícula de ocupación), es decir, este método OGM hace referencia a los mapas que pueden ser creados a través de la detección de ambientes interiores por medio del sensor de luz infrarroja (Syahrul F, 2019). Los métodos usualmente usados son:

Gmapping

Es una de las técnicas SLAM que al mapear sus resultados son precisos respecto a la creación de mapas. Su método se fundamenta en la generación de mapas basados en gráficos y cuadrículas de ocupación RBPF. Para la representación del SLAM, es fundamental el uso de una cantidad relativamente pequeña de partículas RBPF en su algoritmo; así permitiendo una reducción de recursos computacionales para el muestreo y generación de mapas. Se adapta a la necesidad y objetivo de la aplicación que se requiera debido a la libertad de acceso a la configuración de los parámetros de mapeo, como son: el límite inferior de muestreo, cantidad de partículas RBPF y pasos de desplazamiento (Abdelrasoul Y, 2016, pág. 1).

Hector SLAM

Método SLAM que sirve para mapear un entorno, el cual se acopla con el sensor de luz infrarroja LIDAR. La estimación del actual estado del sistema se basa en resultados de escaneos previos; sin embargo, en su etapa inicial padece de graves desviaciones, sin alterar la ejecución del mapeo durante el proceso, pero interfiriendo en la estimación de la posición futura del robot móvil. Hector SLAM compara sus trayectorias creadas con trayectorias de marcos de referencias anteriores, con la finalidad de obtener la estimación de rotaciones y traslaciones que son consecuencia del desplazamiento del robot móvil desde su punto inicial. (Wei W, 2019, pág. 125)

Cartographer SLAM

A través del almacenamiento en cuadrículas de probabilidad, o submapas, de los datos resultantes del escaneo y de las restricciones con enfoque de rama y límite para el cierre de bucles, Google Cartographer se establece como una óptima biblioteca para aportar en su aplicación con SLAM. (Nuchter, Bleier, Schauter, & Janotta, 2017)

Para comprender la forma de trabajo de Cartographer es necesario mencionar los subsistemas que los componen que son el SLAM global y el local. En el SLAM local se construyen submapas de regiones y en el SLAM global se juntan los submapas para hacer una presentación completa del entorno.

CAPÍTULO 2

2. METODOLOGÍA

2.1. Selección del método SLAM

Para desarrollar la parte de implementación del proyecto se dispone de una base móvil de manejo diferencial compuesta de un sistema de odometría y un sensor laser, con estos datos de delimitó la búsqueda de la técnica sólo entre las basadas en SLAM 2D con sensor laser; que son Gmapping, Hector SLAM y Cartographer. En un estudio donde se compara estos tres tipos de SLAM, resultó que GMapping construyó un mapa inexacto, mientras que Héctor SLAM y Catographer proporcionaron mapas bastante similares con los mejores resultados en la calidad del mapeo. Sin embargo, dado que Cartographer utiliza el ciclo de optimización de mapas global y actualizaciones de mapas probabilísticos locales, hace que este sistema sea más robusto a los cambios del entorno. (Filipenko & Afanasyev, 2018). A partir de este estudio se seleccionó a Cartographer SLAM como el método que se utilizaría para el desarrollo del proyecto. Es importante señalar que esta técnica ya es utilizada por empresas dedicadas a la fabricación de robots móviles autónomos colaborativos, como es el caso de Mobile Industrial Robots (MIR) empresa internacional que emplea este tipo de técnica dentro de la navegación de sus robots.

2.2. Requerimiento de hardware

RPLIDAR A1

Es un sensor láser que se fundamenta en un principio que detalla el rango de triangulación laser, para lo cual usa recursos en hardware de procesamiento y adquisición de visión de mucha velocidad, donde los datos de distancia que mide son más de 8000 veces por segundo. Su rango de detección laser omnidireccional es de 360 grados a favor de las manecillas del reloj en un entorno circundante. Además, la precisión y rapidez del mapeo por parte del robot es dependiente de la frecuencia de muestreo del sensor LIDAR. (SLAMTEC, s.f.)

Figura 2.4 Plataforma Arlo Robot

Tomado de [http://www.SLAMtec.com/en/LIDAR/a1]

Este sensor Laser está conformado por un sistema de rango de escaneo y sistema de motor que luego de encender cada uno de sus subsistemas el RPLIDAR inicia su giro y empieza a escanear conforme al sentido de las manecillas del reloj. La comunicación a través del puerto serial/ USB permite obtener información del rango de escaneo. La figura 2.2 describe los sistemas del RPLIDAR A1

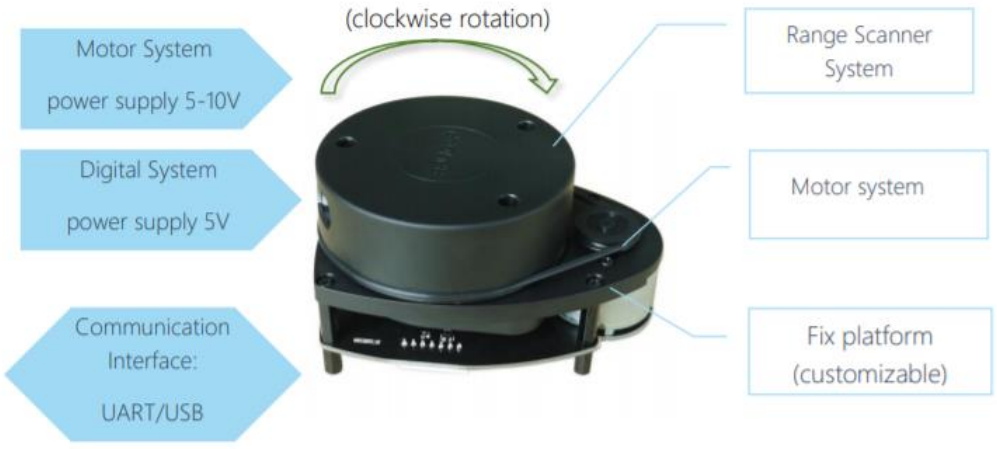


Figura 2.5 los sistemas del RPLIDAR A1

[Tomado de
http://bucket.download.slamtec.com/7fe7e3656e811ab1a645753af40809f05fa7ddcd/LD108_SLAMTEC_rplidar_datasheet_A1M8_v2.4_en.pdf]

La velocidad del sistema de rango de escaneo es alta, el cual está ubicado encima de un rotor giratorio que cuenta con un sistema incorporado de codificación angular, donde dentro de un entorno interior es capaz de realizar un escaneo de 360 grados como se muestra en la figura 2.3

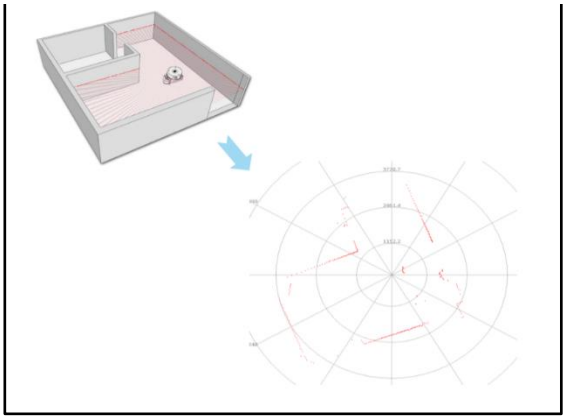


Figura 2.6 Escaneo de 360 grados

El puerto serial (UART) de 3.3V TTL es usado por el sensor RPLIDAR A1 como interfaz de comunicación con el mini PC. Las especificaciones de cada pin se muestran en la figura 2.4

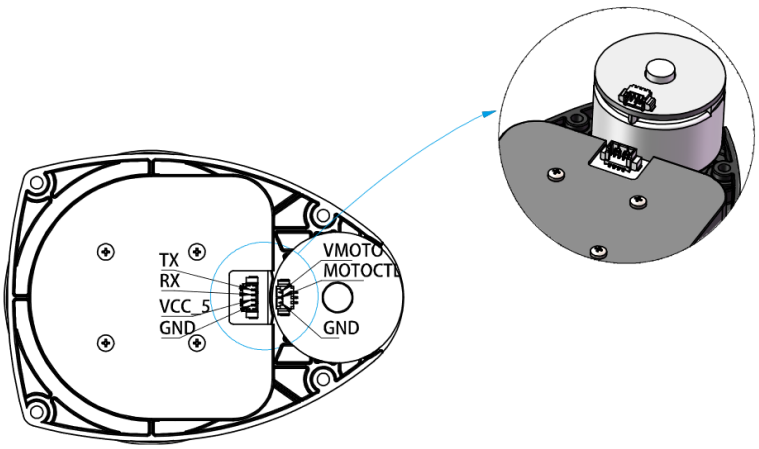


Figura 2.7 Puerto serial RPLIDAR A1

Encoder

El encoder usado en la plataforma del robot ARLO es el encoder de cuadratura Parallax 36-position, este es un tipo de transductor electromecánico que convierte el movimiento mecánico en códigos o pulsos digitales. Además, las 36 posiciones que vienen integrada en el mismo ofrecen retroalimentación rotacional óptimas para las ruedas del robot ARLO, y poseen dentro de un conjunto de sensor dos salidas con un respectivo desfase.

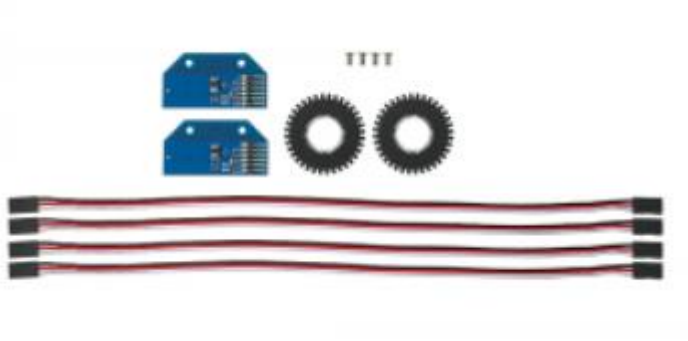


Figura 2.8 Parallax 36-position Quadrature Encoder

[Tomado de https://www.parallax.com/product/29321]

Es el encargado de convertir el movimiento de las llantas en datos digitales de velocidad y dirección que son procesados en la programación en ROS para el mapeo, navegación y localización usando la técnica SLAM Cartographer. La figura 2.6 muestra la conexión del conjunto de cables a la tarjeta del encoder en base a los colores establecidos.

Figura 2.2.9 Encoder

[Tomado de https://www.parallax.com/sites/default/files/downloads/29321-36-Pos-Encoder-Set-v2.0.1.pdf]

Arlo Robot

Arlo Complete Robot System es una plataforma terrestre y móvil que es capaz de realizar tareas guiadas y autónomas sin importar su complejidad. Este tipo de robot está compuesto por partes electrónicas y mecánicas; tales como: motores DC 12V, controlador DHB-10, placa de distribución de energía, sensores ultrasónicos, encoders, placas superiores e inferiores de HDPE, soportes y ruedas de aluminio.

Figura 2.10 Plataforma Arlo Robot

[Tomado de https://learn.parallax.com/tutorials/robot/arlo/arlo-robot-assembly-guide]

ESP32

Sistema embebido integrado por varios módulos, puertos, elementos electrónicos, antenas. Posee un microprocesador Tensilica Xtensa LX6 de 1 o 2 núcleo. Este sistema es programable con el fin de desarrollar aplicaciones. Tiene la característica de funcionar como sistema independiente o esclavo de uno o varios microprocesadores, así como es posible la interacción con otros sistemas para la asignación de funcionalidades.

Figura 2.11 ESP32

[Tomado de https://www.espressif.com/en/products/socs/esp32/overview?1]

Mini PC

Ordenador que aloja en ella los softwares, el sistema operativo y drivers que son recursos importantes para el desarrollo del proyecto. El sistema operativo usado es Ubuntu 18.04.04 y el desarrollador del software robótico es ROS Melodic. Para el manejo de los recursos de software de manera eficiente es necesario que la PC cumpla con los siguientes requerimientos de hardware:

- Memoria RAM al menos 8 GB
- Procesador Intel Core i7-8550U
- Disco duro de 512 GB SSD

Figura 2.12 Mini PC

[Tomado de https://www.amazon.com/HISTTON-Generation-Processors-Fanless-1000Mbps/dp/B07K27JGYW]

DHB-10 Driver

Controlador de doble puente H para motores de 10A es de código abierto, posee un microcontrolador a bordo para mejorar la precisión de bucle cerrado o abierto, las entradas permiten la identificación de PWM o datos en serie.

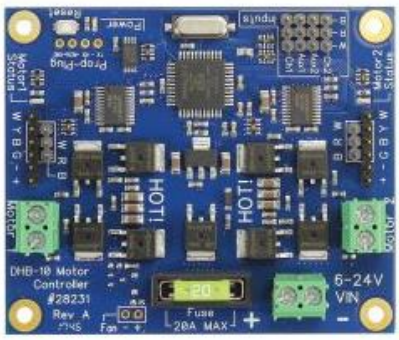


Figura 2.13 DHB-10 Dual H-Bridge 10 Amp Motor Controller

[Tomado de https://www.parallax.com/product/28231]

En base a la revisión de los dispositivos que podrían intervenir en el desarrollo de la navegación autónoma; se seleccionó los equipos mencionados y descritos en la sección recursos de hardware porque las características físicas y electrónicas, costos, y la compatibilidad con el software, se apegan a los dispositivos necesarios para la implementación de la técnica de navegación autónoma de forma experimental. Las especificaciones técnicas de los sensores, driver, controlador y la plataforma Arlo robot se encuentran detalladas en las tablas que se presentan a continuación.

Tabla de especificaciones técnicas de los sensores

Tabla 2.1 Especificaciones técnicas de los sensores

Sensor	Ítem	Especificaciones
RPLIDAR A1	Dimensiones	60×98.5 mm
	Peso	170 g
	Voltaje de operación	5 V
	Corriente de operación	100 mA
	Band rate	115200 bps
	Comunicación	Puerto serial UART 3.3 V-TTL o USB
	Rango de temperatura	0-40 °C
	Rango de medición	0.15-12 m
	Rango angular	0-360 grados
	Frecuencia de medición	≥ 8000
	Pines de entrada	4 (Tx, Rx, Vcc, Gnd)
	Resolución de distancia y angular	<0.5 mm y <1 grados
	Rango de escaneo	5.5 Hz
Encoder	Dimensiones	5.1×2.7×0.6 cm
	Voltaje de alimentación	2.5 VDC - 5.5 VDC
	Corriente de alimentación	3.5 mA – 11.6 mA
	Comunicación	Salida de pulso high/low de dos canales
	Rango de temperatura	0-70 °C
	Disco del encoder	36 posiciones
	Pines de entrada	6 (Gnd, Vin, A, B, Vin, Gnd)

Nota. Fuente: RPLIDAR A1 Introduction and Datasheet rev 2.4 del fabricante SLAMTEC. 36-position Quadrature Encoder Set Datasheet v2.0 del fabricante PARALLAX INC.

Tabla de especificaciones técnicas del controlador

Tabla 2.2 Especificaciones del controlador ESP32-WROOM

Categoría	Ítems	Especificaciones
Hardware	Interfaz de módulo	SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	Sensor On-chip	Sensor Hall
	Cristal integrado	Cristal de 40 MHz
	SPI flash integrado	4 MB
	Voltaje de operación	3.0 V ~ 3.6 V
	Corriente de operación	Promedio: 80 mA
	Corriente mínima de la alimentación	500 mA
	Temperatura de operación	-40°C ~ +85°C
	Tamaño del paquete	(18.00±0.10) mm × (25.50±0.10) mm × (3.10±0.10) mm
	Nivel de sensibilidad de la humedad (MSL)	Nivel 3

Nota. Fuente: ESP3-WROOM-32 Datasheet v2.9 del fabricante ESPRESSIF SYSTEMS.

Tabla de especificaciones técnicas del Driver

Tabla 2.3 Especificaciones técnicas del DHB-10 Driver

Driver	Ítems	Especificaciones
DHB-10 Dual H-Bridge 10 A Motor Controller	Voltaje de alimentación	6-24 VDC
	Corriente máxima para el motor	10 A por canal
	Corriente máxima del motor de sobretensión	12 A por canal
	Comunicación	19200-115200 baud Serial o PWM
	Dimensiones	77.5 x 65 mm
	Rango de temperatura de operación	0-70 °C

Nota. Fuente: DHB-10 Dual H-Bridge 10 Amp Motor Controller v1.0 del fabricante PARALLAX INC.

Tabla de la descripción del sistema completo del Arlo robot

Tabla 2.4 Descripción del Arlo Complete Robot System

Cantidad	Descripción
1	DHB-10 Dual H-Bridge Motor Controller
1	Arlo Base Kit
2	Caster Wheel Kit
1	Motor Mount & Wheel Kit –Aluminum
1	Arlo Top Deck
1	Arlo Power Distribution System
1	Propeller Plug programming tool
1	Propeller Activity Board WX
1	1 x 6 Straight Header
1	Arlo Smart Charger (2.1 mm plug)
2	Battery, 12 V, 7.2 amp-hour, sealed lead acid

Nota. Fuente: Arlo Complete Robot System v1.2 del fabricante PARALLAX INC.

Diseño de hardware

En la figura 2.11 se presenta el diseño de la arquitectura para la implementación de la técnica de navegación autónoma utilizando SLAM Cartographer en la plataforma del Arlo robot.

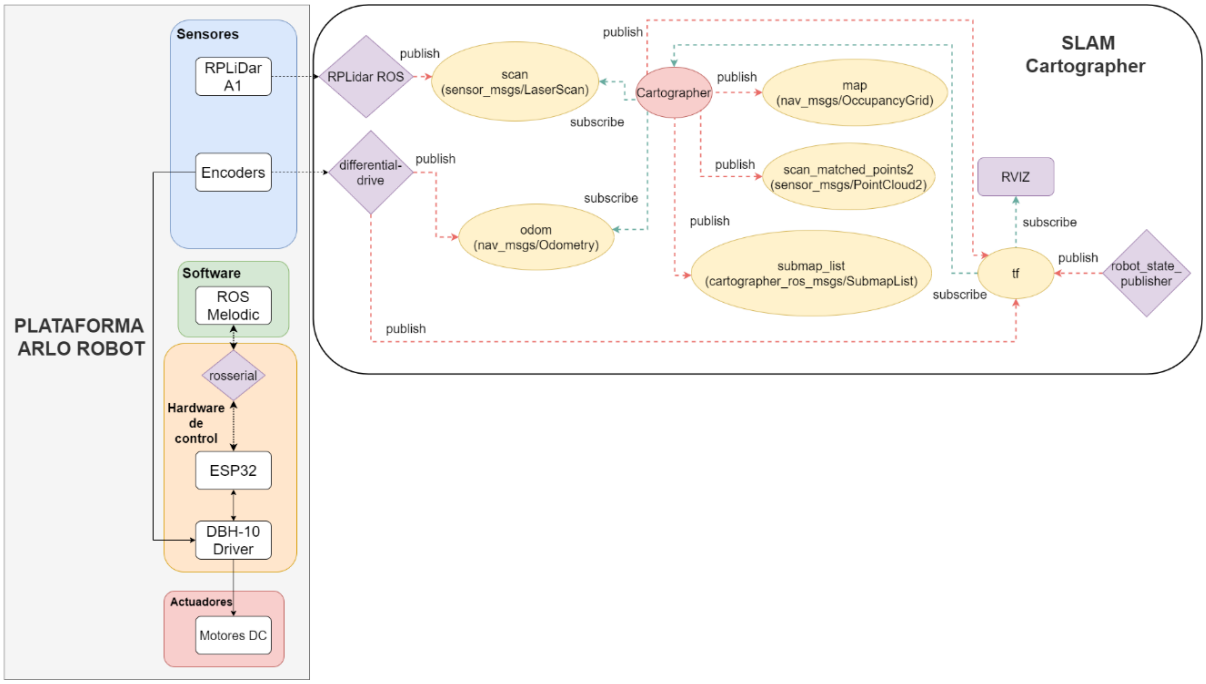


Figura 2.14 Arquitectura para la implementación

La arquitectura está conformada por la plataforma del Arlo robot, el controlador ESP32-WROOD32, el DBH-10 Driver, los dos motores DC, los dos encoders, y el sensor RPLIDAR A1 eso con respecto al Hardware de la implementación. Para ejecutar la técnica SLAM de Cartographer para el mapeo y localización, de manera general se utiliza el software ROS Melodic, RViz, los paquetes de ROS los cuales son: rosserial, RPLIDAR ROS, differential_drive, robot_state_publisher, y el nodo de Cartographer. En la arquitectura hace referencia a Tópicos que son publicados y suscritos en el nodo de Cartographer a través de mensajes (.msgs).

Descripción general de la arquitectura del proyecto

Inicialmente el controlador ESP32-WROOM-32 establece comunicación serial con el driver DBH-10 controla los dos motores de 12 VDC que por medio de los encoders toman los datos del movimiento con ROS Melodic a través del paquete rosserial. El sensor RPLIDAR A1 se conecta a un puerto USB de la mini PC al

igual que los hilos que provienen del controlador. Una vez establecida la conexión de los sensores se procede a correr el programa en los terminales, donde se utiliza los paquetes RPLIDAR ROS y differential_drive para el manejo del sensor laser RPLIDAR A1 y los encoders de cuadratura de 36 pasos respectivamente. Además, dentro de estos paquetes se encuentran los nodos que se publican en los tópicos scan y odom; a su vez el nodo de Cartographer se suscribe en los tópicos mencionados a través de los mensajes sensor_msgs/LaserScan y nav_msgs/Odometry. En el tópico tf se publica el estado del robot a través del paquete robot_state_publisher. El software Rviz y el nodo de Cartographer se suscriben en el tópico tf permitiendo al usuario realizar un seguimiento de múltiples marcos de coordenadas a través del tiempo. Finalmente, el nodo de Cartographer publica en los tópicos scan_matched_point2, submap_list y map; este último Tópico arroja el mapa en Rviz realizado por Cartographer.

En la sección de diseño de software para la implementación se hará una descripción de todos los nodos y Tópicos que intervienen en la programación del sistema para el mapeo, navegación y localización a través de una arquitectura gráfica.

Conexión del driver DBH-10

Dentro de la plataforma del Arlo robot se encuentra el driver DBH-10 que controla los dos motores de 12 VDC y permitirán mover el kit de ruedas de la plataforma. Los encoders se encuentran conectadas a los motores 12 VDC y están comunicados con el DBH-10 driver. En la figura 2.12. se observa los componentes del driver DBH-10.

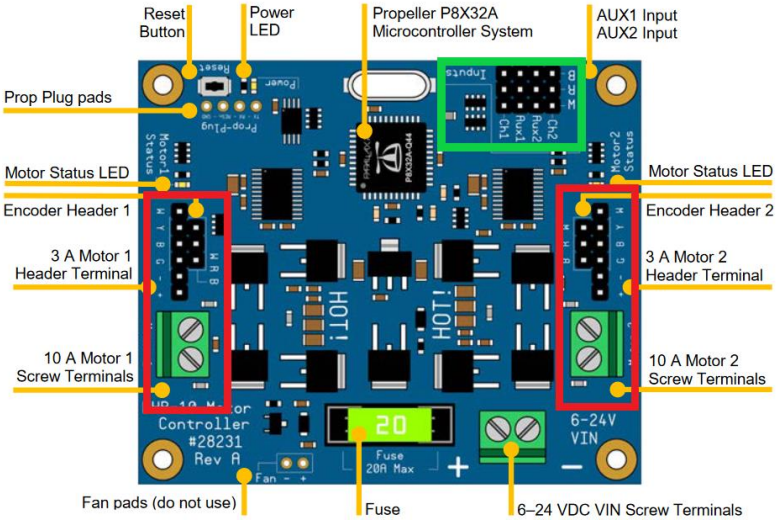


Figura 2.15 Conexión del driver DBH-10

Los recuadros rojos que se observan en la figura 2.12 encierran los terminales Encoder Header compatibles para la conexión de los encoders de cuadratura de 36 posiciones Parallax, donde los 3 pines para la conexión están designados por colores negro, rojo y blanco BRW, de acuerdo con los colores de los cables de los encoders y los motores de 12 VDC van conectados en los Screw Terminals. El recuadro verde indica los pines de entrada CH1 y CH2 para la comunicación serial entre el driver DBH-10 y el controlador ESP32-WROOM-32.

Conexión del controlador ESP32-WROOM-32

Es importante que el controlador ESP32-WROOM-32 establezca comunicación con el driver DHB-10 y con ROS Melodic, para aquello utiliza la comunicación serial a través de sus pines GPIO: Rx y Tx. En la figura 2.13 se observa la descripción de los pines del controlador.

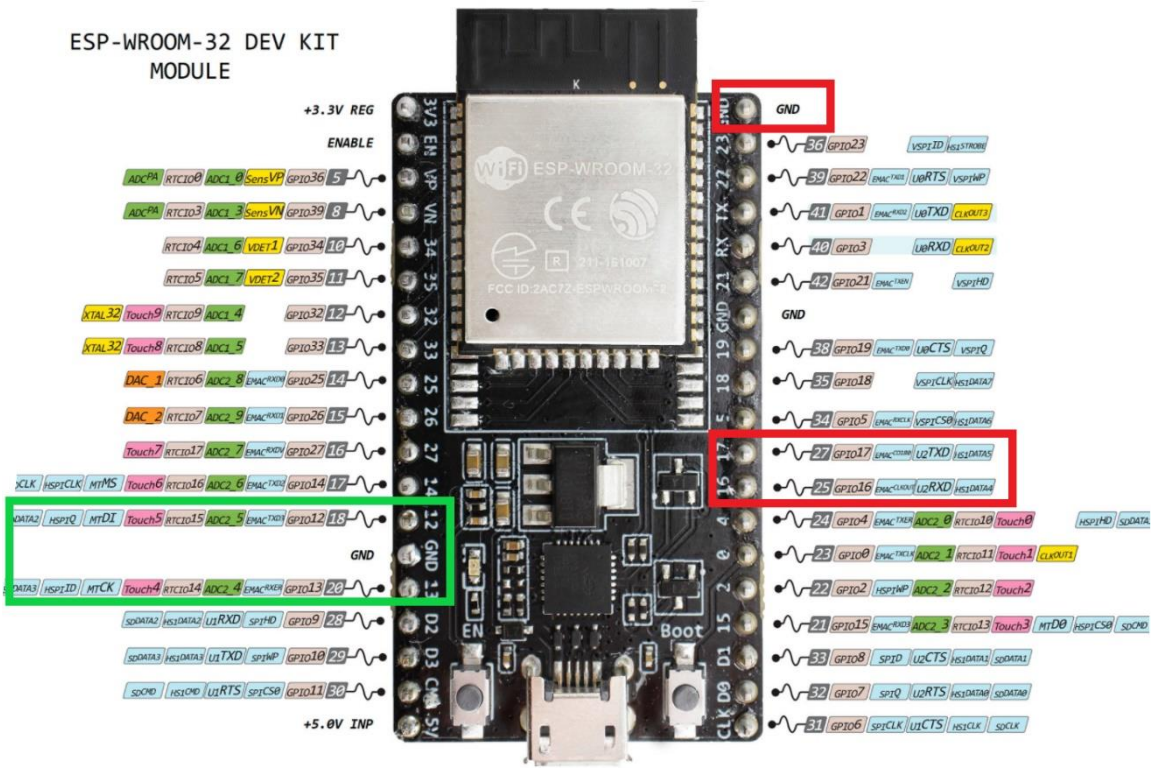


Figura 2.16 Conexión del controlador ESP32-WROOM-32

Para la comunicación serial con el driver DHB-10 se utiliza los pines GPIO 17, 16 que actúan como Tx y Rx, y el pin de GND. Para la comunicación serial con el sistema operativo del robot ROS Melodic se utiliza los pines GPIO 12, 13, a pesar de no actuar como TX y Rx, se puede establecer la comunicación mediante software en ROS Melodic. Para cargar al controlador ESP32 el main.py se asigna en la programación los pines GPIO1 y GPIO3.

Descripción del funcionamiento de los actuadores

Para el manejo del movimiento de la base robótica se utiliza dos motores de aluminio de 12 VDC instaladas en cada rueda respectivamente, formando un sistema de accionamiento mecánico diseñado para plataformas robóticas móviles. Estos motores son accionados por medio de engranajes helicoidales, donde la medida del eje de transmisión principal es de 0.5 pulgadas de diámetro. Posee una rodadura suave debido al alojamiento del eje en una agrupación de cojinetes de bola. En la figura 2.14 se observa el ensamble del motor.

Figura 2.17 Funcionamiento de los actuadores

Con respecto a los encoder de cuadratura de 36 posiciones que están conectados a los motores calculan 144 pasos discretos de encoder por cada revolución de la llanta, esto implica una alta precisión en la navegación por el entorno interior. En la figura 2.15 se observa el sistema de accionamiento mecánico para la navegación.

Figura 2.18 Encoder de cuadratura de 36 posiciones

2.3. Requerimiento de software

Ubuntu 18.04

software abierto, proviene de Linux y está basado en Debian. Es uno de los más preferidos por programadores debido a su amplio soporte.

ROS

“ROS es un middleware robótico, es decir una colección de frameworks para el desarrollo de software de robots” (openwebinars, s.f.). Suministra servicios de un sistema operativo, a pesar de no serlo; por ejemplo, abstracción del hardware, control de dispositivo, intercambio de mensajes y el mantenimiento de paquetes.

Concepto y arquitectura de ROS

De acuerdo con el libro: Effective Robotics Programming with **ROS** (Mahtani, Snachéz , Fernández , & Martinez , 2016, págs. 73 -96)

ROS se encuentra estructurado en tres niveles:

- El nivel de sistema de archivos
- El nivel gráfico de cálculo
- El nivel comunitario

En el nivel de sistemas de archivo se establecen la cantidad de archivos necesarios para ejecutar los programas en ROS. En el segundo nivel el gráfico de cálculo se produce la comunicación entre los procesos y sistemas. En el tercer nivel se comparten conocimientos entre una comunidad ya que ROS es un software de código abierto.

Características del nivel de sistemas de archivo

Se proporciona flexibilidad para descentralizar dependencias

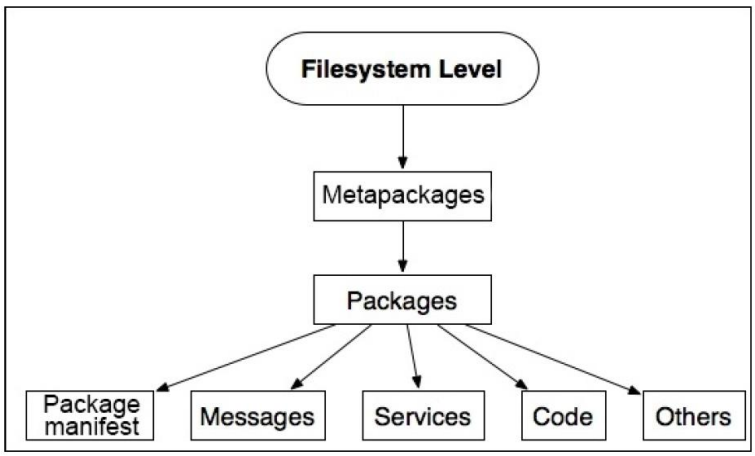


Figura 2.19 Nivel de sistema de archivos

[Tomado de **(Mahtani, Snachéz , Fernández , & Martinez , 2016)**]

Los paquetes: Contiene los nodos, archivos de configuración entre otro contenido necesario para crear el programa usando ROS.

Manifiesto de paquete: Provee información de los paquetes como las licencias, dependencia, indicadores de compilación, etc. Se administra con un archivo llamado package.xml.

Metapaquetes: En caso de tener varios paquetes, ROS los organiza en forma de metapaquetes. Hacen la misma función que las pilas, pero de forma simple.

Manifiestos de metapaquetes: Posee una etiqueta de exportación XML y restricciones en estructuras.

Tipos de mensajes (msg): ROS posee muchos tipos de estándares de mensajes para comunicarse entre procesos. Los mensajes son guardados en my_package/msg/MyMessageType.msg.

Tipos servicios (srv): Las descripciones de los servicios son guardadas en my_package/srv/MyServiceType.srv.

El espacio de trabajo: En la carpeta de espacio de trabajo se contiene los paquetes, facilita una vía para lograr la compilación de paquetes en un mismo tiempo además de centralizar los desarrollos.

El espacio de fuente: En la carpeta src, se colocan paquetes, proyectos, clonado de paquetes y demás. Se encuentra también el archivo CMakelist.txt. que permites que se invoque el contenido de esta carpeta a través de cmake.

El espacio de compilación: En la carpeta de compilación se encuentran las carpetas cmake y catkin, mantienen la información de cache, entre otra información intermedia.

Espacio de desarrollo (devel): Se colocan en este espacio los programas compilados y sirve para probar los programas antes de ser instalados.

Características del nivel de gráfico de cálculo

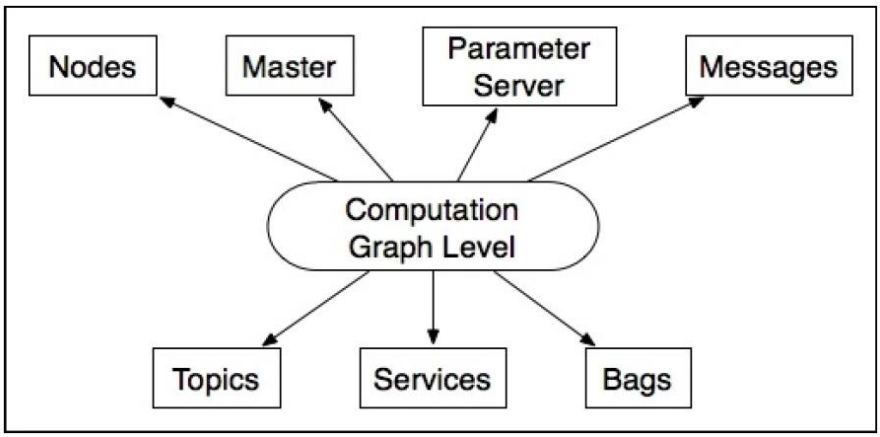


Figura 2.20 Nivel de gráfico de cálculo

[Tomado de (Lopez, 2017)]

A través de una red se conectan todos los procesos. Es de acceso para todos los nodos del sistema para interacción, intercambio de información y la transmisión de datos.

Nodos: Son procesos en los que se realizan los cálculos. Es conveniente tener varios nodos para realizar una función que tener uno o pocos nodos enfocados a una sola función.

El máster: Conoce información y la lleva en un registro de los nodos del sistema, los pone a disposición al resto de nodos. Propone una estructura de conexión entre los nodos. Permite que los nodos se localicen uno a otros

Servido de parámetros: Mediante estos parámetros es posible configurar nodos mientras están corriendo.

Mensajes: Contiene información que proveen a otros nodos.

Tópicos: El nombre que se le da a los mensajes que se envían entre los nodos. Cuando un nodo envía datos, entonces se encuentra publicando un tópico, mientras que cuando un nodo recibe datos, entonces se está subscribiendo a un tópico.

Servicios: Cuando un nodo necesita una hacer un requerimiento o recibir una respuesta de otro nodo hace uso de servicios que permite interactuar entre ellos.

Bags: Permite almacenar datos y luego reproducirlos, por ejemplo, datos de sensores.

Características del nivel comunitario de ROS

Distribuciones: Son colecciones de metapaquetes.

Repositorios: ROS se basa en repositorios de códigos donde se pueden crear nuestros propios componentes de software.

ROS wiki: Foro principal para documentación e información sobre ROS.

Google Cartographer ROS

Cartographer es un sistema utilizado para realizar SLAM 2D y 3D en tiempo real en. Al inicio de su aparición se lo implementó en una mochila equipada para mapeos de entornos interiores mientras el usuario recorría el sitio.

En octubre de 2016 Google lanza Cartographer como un proyecto de código abierto, incluida el soporte de ROS, en el repositorio de Github llamado *Cartographer _ros.*

Actualmente Google utiliza Cartographer para mapear en entornos interiores de establecimientos.

Carographer trabaja en base a dos sistemas el SLAM local y el SLAM global. El SLAM local se encarga de crear los submapas del área próxima al robot, también genera la trayectoria local el robot. En otro proceso separado el SLAM global reúne a los submapas de la forma más consistente posible, basándose en las restricciones de cierre de bucle presentes en los submapas.

Explicación del funcionamiento de Cartographer SLAM

Teniendo los datos de entrada del sensor LIDAR el primer paso es tratar los puntos sin procesar por un filtro de vóxel que los comprime en cubos de igual tamaño, interesándole para su posterior tratamiento solo el centroide de cada cubo. Luego se emplea un filtro vóxel adaptativo que genera una nube de puntos mediante el ajuste el tamaño de los vóxel con el fin de llegar a un número determinado de puntos.

Luego del proceso de filtrado viene la implementación del SLAM local que ajusta los datos filtrados en un submapa a través de un proceso emparejamiento de escaneo que “consistirá en encontrar una transformación rígida de traslación y rotación que alinee dos frames sucesivos” (Lopez, 2017).

Se debe suministrar una suposición del extrapolador de poses con el fin de pronosticar la nueva posición donde se debe insertar el siguiente escaneo dentro del submapa. El empajeramiento de escaneo se lo suele hacer con dos herramientas que proporciona Cartographer que son CeresScanMatcher y RealTimeCorrelativeMatcher. El primero es el más utilizado por su velocidad de ejecución, peros si se posee errores más grandes que la resolución de submapa no los puede ajustar. El segundo es muy costoso, pero más robusto.

CeresScanMatcher es una biblioteca que resuelve problemas de mínimos cuadrados no lineales a partir de un supuesto inicial, determina el sitio optimo en la que el emparejamiento de escaneo encuadra en el submapa.

Posteriormente un filtro de movimientos selecciona los escaneos provenientes de movimientos significativos.

Cuando el SLAM local ha recogido una cantidad explícita de datos de rango se establece como un submapa terminado. El algoritmo de SLAM global recoge los submapas y trata de organizarlos de forma consistente para que formen el mapa global.

Para almacenar los datos de rangos los submapas emplean las cuadrículas de probabilidad que divide el espacio en tablas y cada celda puede o no encontrarse ocupada.

Archivos de configuración LUA

AL trabajar con Cartographer se necesita de una descripción de la configuración del robot descrita en un script Lua. En este archivo se detalla datos como los ID de trama TF tanto del entorno como del robot.

Opciones de nivel superior de la integración de Cartographer con ROS

map_frame: nombre ID que identifica el marco de coordenadas donde se publicarán los submapas.

Tracking_frame: nombre ID que identifica el marco de coordenadas de la trama de la que se lleva un rastreo por el algoritmo SLAM.

Published_frame: nombre ID, que identifica el marco de coordenadas considerado como secundario y utilizado para publicar poses.

Odom_frame: ID que se lo utiliza si el valor de estado de provide_odom_frame es verdadero.

Provide_odom_frame**:** si es verdadero, se publica la pose local, sin bucle cerrado como odom_frame en map_frame.

Publish_frame_projected_to_2d: si es verdadero entonces la pose que se publica será solamente en 2D.

Use_odometry: si es verdadero entonces nav_msgs/Odometry se subscribe sobre el tópico odom.

Use_nav_sat: si es verdadero entonces se subscribe a sensor_msgs/NavSatFix en el tópico fix.

Use_landmarks: si está activada se subscribe a Cartographer _ros_msgs/Landmarklist en el tópico landmarks.

Use_laser_scan: cantidad de tópicos de escaneo láser a los que está subscrito, se subscribe a sensor_msgs / LaserScan sobre el tópico scan.

Num_multi_echo_laser_scan: cantidad de tópicos de escaneo láser de eco múltiple.

Num_subdivisions_per_laser_scan: cantidad de nubes de puntos en las que se dividirán los escaneos láser de entrada (eco múltiple).

Num_ponit_clouds: cantidad de tópicos de nube de puntos para subscribirse. Se subscribe a sensor_msgs / PointCloud2 sobre el tópico point2.

Look_transform_timeout_sec: tiempo en segundos que se espera para buscar un nueva trnasformación tf2.

Submap_publish_period_sec: intervalo de tiempo durante el cual se publican las poses del submapa.

Pose_publish_period_sec: intervalo para publicar poses dentro de una frecuencia de 200 Hz.

Trajectory_publish_period_sec**:** intervalo de tiempo en segundos para la publicación de marcadores de trayectoria.

Rangefinder_sampling_ratio**:** muestreo de proporción fija para mensajes de telémetros.

Odometry_sampling_ratio**:** muestreo de proporción fija para mensajes de odometría.

Fixed_frame_sampling_ratio**:** muestreo de proporción fija para mensajes de marco fijo.

Imu_sampling_ratio: muestreo de proporción fija para mensajes IMU.

landmarks_sampling_ratio: muestreo de proporción para mensajes de puntos de referencia.

Marco de coordenadas

tf2: dado que los robots contienen diferentes marcos de coordenadas 3D que cambian constantemente con el tiempo por ejemplo cada uno de los eslabones del robot posee un marco de coordenadas; la base, la cabeza, la pinza, también está el marco mundial, etc. El paquete tf2 mantiene un seguimiento de todos los marcos de coordenadas en el tiempo. Este paquete opera en un sistema distribuido permitiendo tener acceso a estas coordenadas desde cualquier parte de ROS.

Marcos de coordenadas para plataformas móviles

Base_link: Este marco de coordenadas fijo en la base del robot.

Odom: Es un marco de coordenadas fijo en el mundo lo que significa que siempre variará la ubicación de una plataforma móvil con el tiempo y sin límites. Se lo puede usar como una referencia local precisa a corto plazo, pero por la desviación es muy poco favorable como referencia global a largo plazo. Generalmente se calcula este marco en base a una fuente de edometría.

Map: Es un marco de coordenadas fijo en el mundo, la referencia de la base móvil con respecto al marco map no debe de variar significativamente con respecto al tiempo. La ubicación de la base móvil usando el marco map no es continua y se puede presentar como saltos discretos. Generalmente en la etapa de localización siempre se está calculando continuamente la pose del robot con este marco de coordenadas y depende de los datos adquiridos con los sensores lo que elimina la desviación y son culpables de generar saltos discretos en cada actualización de los datos de entrada. Este marco es preferible colocarlo como una referencia global a largo plazo.

Árbol de URDF (Universal Robot Description Format)

Estos archivos precisan en forma de una descripción, los elementos físicos de un robot. Se definen la relación y los enlaces existentes entre los sensores y otros elementos de robot.

Archivo de Lanzamiento

En él se precisan el robot a utilizar, así como ciertos valores para algunos parámetros necesarios para que se ejecute el proyecto. También se puede hacer referencia a otros archivos launch.

Se necesita de un archivo launch principal que carga todos los datos incluyendo los nodos y algunos soportes necesario para que se encuentre disponible la información y datos necesarios para correr el proyecto.

Procedimiento para crear un fichero ejemplo.launch:

1. Crear un directorio llamado launch con el comando **mkdir** launch
2. Acceder al directorio con el comando **cd** launch
3. Crear el fichero ejemplo.launch
4. Abrir y editar el texto del fichero con el comando **gedit** ejemplo.launch, en donde se agregan los múltiples nodos a ejecutar, estos pueden publicar o suscribir. Además, se procede a editar los nombres del paquete, el tipo de nodo y el nodo a ejecutar
5. Ejecutar con el comando **roslaunch** ejemplo.launch en el terminal de ROS

Comunicación/ Control de base móvil

uPy-rosserial

Método para comunicación serial, se lo categoriza como un middleware. En este proyecto permite la comunicación entre ROS y uPy (FunPythonEC, uPy-rosserial, 2020).

uPyArlo

Librería de microPython para controlar el puente H DBH- 10 de la base móvil Arlo Parallax platform (FunPythonEC, uPyArlo, 2019).

Tabla 2.5 Paquetes y nodos de ROS utilizados

Paquetes	Nodo	Función	Descripción
Cartographer _ros	Cartographer _node	mapeo	Permite el SLAM en tiempo real
	Cartographer _occupancy_grid_node	mapeo	Recibe los submapas publicados por SLAM y construye un ROS occupancy_grid y lo publica
Turtlebot_teleop	turtlebot3_teleop_keyboard	mapeo	Permite la teleoperación del robot mediante joysticks o teclado, útil para el proceso de mapeo.
robot_state_publisher	robot_state_publisher	todo	Este paquete le permite publicar el estado de un robot en marcos de coordenadas
tf2	tf2	todo	Permite al usuario realizar un seguimiento de múltiples marcos de coordenadas a lo largo del tiempo
map_server	map_server	mapeo	Los mapas tratados se guardan, generando dos archivos. El archivo YAML que posee metadatos del mapa y el archivo de la imagen que codifica datos de ocupación.
gazebo_ros_pkgs	gazebo	mapeo	Simulación del entorno
rviz	n_rviz	todo	Herramienta de visualización
move_base	move_base	navegación	El nodo move_base vincula un planificador global y local para realizar su tarea de navegación global.
amcl	amcl	localización	Permite la localización probabilística, toma un mapa basado en láser, escaneos láser y mensajes de transformación, y genera estimaciones de localización.

Programas requeridos para recrear un ambiente simulado

Onshape

Software CAD para dibujo y diseño colaborativo mediante computadora, que trabaja en la nube a través del uso de servidores de internet. Varios usuarios pueden hacer modificaciones en un mismo dibujo a la vez.

Rviz

Herramienta de visualización 3D para ROS permite observar al robot y su comportamiento. Entre las ventajas que ofrece es el hecho de registrar lo que el sistema de recepción del robot sensa y luego la reproducen.

Gazebo

Software de simulación en 3D utilizado para recrear escenarios que permitan evaluar la interacción entre el robot y su entorno.

2.3.1. Diseño de Software

Implementar una técnica de navegación usando SLAM en un entorno simulado

Para llevar a cabo la simulación de Cartographer se necesitó recrear un entorno simulado que se asemeje al lugar por donde el robot navegaría en la aplicación experimental. Con la ayuda del software Onshape se logró el dibujo en 3D del espacio, luego se realizó las conversiones y procesos necesarios para adaptar dicho escenario dentro de ROS con la ayuda Gazebo un software de simulación.

Preparación de un entorno de prueba simulado.

En Onshape se dibujó el área de la sala de un apartamento. fue necesario obtener medidas físicas de la casa como por ejemplo el área, altura, ancho, largo, etc.

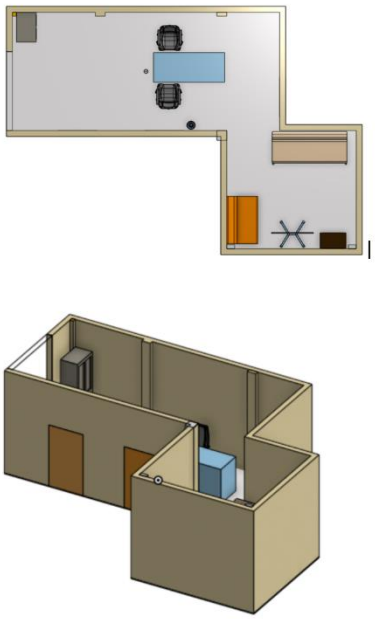


Figura 2.21 Diseño en 3D del escenario de navegación en OnShape

Conversión de archivo de extensión. stl a .sdf

OnShape ofrece entre sus opciones de archivos para exportar la extensión stl que es un archivo de dibujo 3D. Dado que Gazebo trabaja con extensión .sdf que almacena una descripción del dibujo de forma estructurada fue necesario hacer una conversión en la extensión del dibujo de stl a sdf.

Para este proceso se ha utilizado el proyecto en Github de Rhoban llamado onshape-to-motor. (Burdeos, 2020)

Se debe inició instalando los paquetes requeridos, clonando el repositorio e instalando todas las dependencias necesarias como se indican el archivo README del repositorio **onshape-to-robot**

Para poder extraer los modelos de Onshape se creó en un directorio deseado una carpeta donde almacenar los modelos de 3D a exportar, a esta carpeta se la denominado con el nombre de **casa** y dentro de esta se ha creado un archivo jason con datos necesarios para la exportación como por ejemplo el número el código de identidad, nombre del proyecto, la extensión, la URL del dibujo en Onshape, la clave de acceso necesarias.

```
{
    "documentId": "0540bfeb79ac2f02ca575fe8",
    "assemblyName": "EntornoV1",
    "outputFormat": "sdf",
    "onshape_api": "https://cad.onshape.com",
    "onshape_access_key": "2uYsXuPjLzct4vpkGN25PZae",
    "onshape_secret_key": "JCVRorQZgIMvx50QJJb0GYmf1Vzgvv9DFIvYvGGBSEI0vq5h"
```


Figura 2.22 Archivo. jason con los datos necesarios para la exportación

Desde el directorio de **casa** se abre un nuevo terminal y se corre con permisos de administrador el siguiente comando.

```
sudo onshape-to-robot casa/
```

De esta forma se realiza la extracción de los archivos sdf dentro de la carpeta seleccionada.

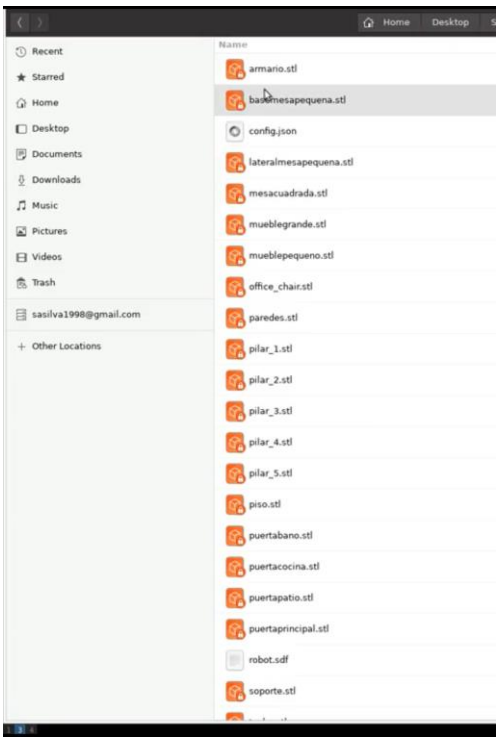


Figura 2.23 Extracción de los archivos sdf en ROS

En la imagen 2.19 se puede apreciar que se detalla parte por parte de la casa como son los pilares; el piso, paredes, entre otros, esto es debido a que todo se ha dibujado por partes y luego se lo ha ensamblado en un solo archivo sdf de Onshape.

Posteriormente estos archivos generados fueron copiados en la carpeta **.gazebo** que se encuentra en el directorio Home. Aquí se crea una carpeta a la que llamamos **entorno_sim** y dentro de esta otra que debe llamarse meshes donde se pegaron los **modelos.sdf** extraídos. De todos estos modelos guardados se cambia la ubicación del que se llama **robot.sdf** y se lo coloca fuera de la carpeta meshes es decir en el directorio: **Home>>.gazebo >> models >> entorno_sim**. Para que gazebo tenga lectura de estos archivos es necesario que lleven nombres estándares, por esta razón se cambió el nombre del archivo **robot.sdf** a **model.sdf**

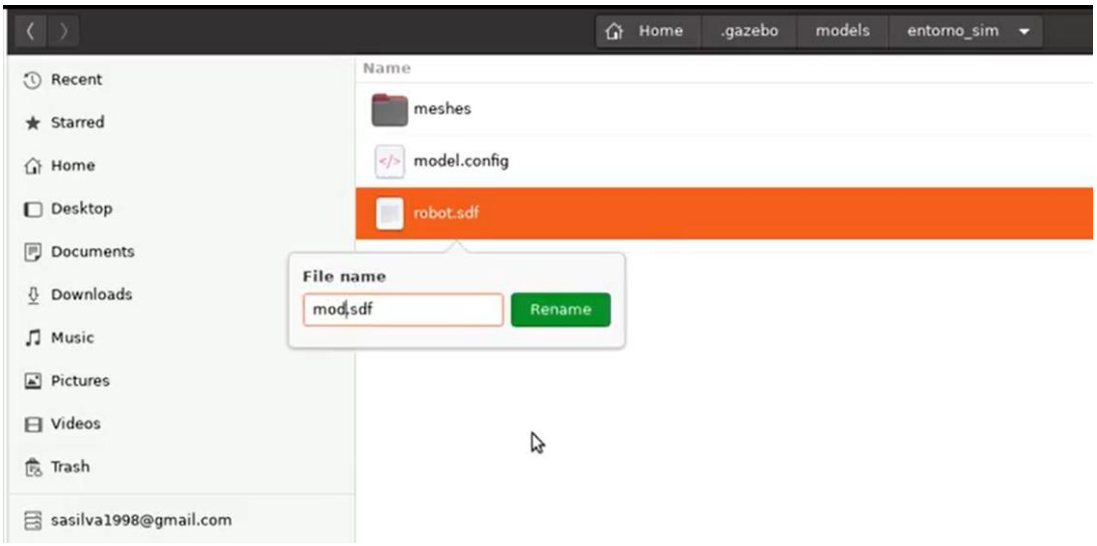


Figura 2.24 Cambio de nombre del archivo del modelo del robot sdf

También es requisito tener dentro de esta carpeta un archivo .**config** que se creó con datos como por ejemplo la versión de **sdf**, el autor y la descripción.

Archivo .config

```
<?xml version="1.0"?>
<model>
  <name>Entorno Simulación</name>
  <version>1.0</version>
  <sdf version='1.6'>model.sdf</sdf>

  <author>
   <name>My Name</name>
   <email>me@my.email</email>
  </author>

  <description>
    Entorno representando
  </description>
</model>
```

Figura 2.25 Archivo config

Desde una terminal se lanza gazebo y se elige a este escenario de simulación como el world de nuestro robot simulado.

Instalación de Cartographer _ros

Para facilitar el desarrollo del proyecto se ha creado un repositorio privado en GitLab donde se guardan las colaboraciones de todos los integrantes a medida que se actualiza el contenido de las carpetas del proceso.

Se ha compilado el proyecto de **Cartographer _ros** en la distribución de ROS melodic (google).

Dentro del repositorio llamado **amr** en la rama de **navegación** se ha colocado un documento README donde se especifica los pasos a seguir para que una persona que visite el repositorio pueda correr el proyecto de simulación desde su computador.

Antes de iniciar con el clonado del repositorio de GitLab es necesario actualizar los programas que requieren de nuevas versiones.

En una terminal de Ubuntu correr:

```
sudo apt update
```

Se instala el modelo del robot diferencial utilizado en la simulación que es el Turtlebot 3 de modelo burguer, escribiendo en la terminal:

```
sudo apt install ros-melodic-turtlebot3-* ros-melodic-ros-control ros-melodic-ros-controllers ros-melodic-dwa*
```

Para mayor facilidad se ha modificado el archivo **~/.bashrc** ubicado en el directorio HOME para que las variables de entorno de ROS estén disponible de forma automática cada vez que se abre una nueva terminal, de esta forma se evita el trabajo tedioso de escribir el mismo comando en cada nueva terminal.

```
source /opt/ros/melodic/setup.bash
```

```
source /home/maria/amr/amr_ws/devel_isolated/setup.bash
```

Además, se añade al final del archivo **~/.bashrc** la siguiente línea para seleccionar de forma automática las variables del modelo de robot predeterminado que se va a usar.

```
echo "export TURTLEBOT3_MODEL=burger" >> ~/. bashrc
```

```
source ~/.bashrc
```

Figura 2.26 Modificaciones al archivo. Bashrc

En el caso de que no añaden los comandos se debe correr lo siguiente en cada terminal a usar:

```
export TURTLEBOT3_MODEL=Burger
```

Para instalar Cartographer ROS se recomienda usar rosdep, wstool y Ninja para agilizar el proceso de instalación.

Wstool: Esta herramienta suministra comandos para gestionar y mantener un espacio de trabajo de repositorios locales SCM (Sistemas de Control de Código

Fuente) como por ejemplo git, cuyo espacio de trabajo se basa en un archivo de rosinstall. Permite el control de versiones.

Rosdep: Esta herramienta sirve para instalar dependencias del sistema.

Ninja: Este sistema permite que las compilaciones se ejecuten de forma rápida.

Detalles del archivo README del repositorio

Abriendo una nueva terminal en Ubuntu desde un directorio deseado correr:

```
git clone  https://GitLab.com/dpailla/amr.git
```

Nos ubicamos dentro de la carpeta amr que se ha creado tras correr la línea anterior y que contiene el espacio de trabajo de nuestro proyecto:

```
cd ./amr
```

Debido a que en el repositorio existen varias ramas de trabajo de otros investigadores, fue necesario cambiar hacia la rama pertinente para acceder a los archivos del árbol de trabajo que necesitamos:

```
git checkout navegación
```

```
El siguiente paso descrito en el archivo README consiste en copiar la carpeta entorno_sim dentro de la carpeta .gazebo.
```

```
cp -avr ./entorno_sim ~/.gazebo/models
```

```
Sin embargo, esta acción ya lo habíamos hecho en el paso anterior de la creación del entorno, por lo tanto, solo sería necesario actualizar la carpeta.
```

```
Ubicados dentro del espacio de trabajo amr_ws se inicializa el mismo con el siguiente comando:
```

```
wstool init src
```

```
Para fusionar archivos rosinstall se corre
```

```
wstool merge -t src
https://raw.Githubusercontent.com/googleCartographer/Cartographer_ros/master/Cartographer_ros.rosinstall
```

```
wstool update -t src
```

Inastalación de las dependencias de Cartographer_ros.

```
src/Cartographer /scripts/install_proto3.sh
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
```

Construcción e instalación de Cartographer ROS

```
catkin_make_isolated --install --use-ninja
catkin_make_isolated --install-space Cartographer _launcher --use-ninja
catkin_make_isolated --install-space my_robot_description --use-ninja
catkin_make_isolated --install-space rpLIDAR_ros --use-ninja
catkin_make_isolated --install-space Arlobot_description --use-ninja
catkin_make_isolated --install-space Arlobot_bringup --use-ninja
catkin_make_isolated --install-space differential-drive --use-ninja
```

Tabla 2.6 Datos del archivo launch del robot

my_turtlebot_launcher. launch		
Argumentos del robot	model	TURTLEBOT3_MODEL
	model type	waffle
	x_pos	0.0
	y_pos	0.0
	z_pos	0.2
Gazebo	include file	(find gazebo_ros) /launch/empty_world.launch
	world_name	(find my_robot_description) /worlds/entorno.world"/
	paused	false
	use_sim_time	true
	gui	true
	headless	false
	debug	false
Parámetros del robot	robot_descriptio n	find turtlebot3_description)/urdf/turtlebot3_$(arg model). urdf. xacro
node pkg	gazebo_ros	
type	spawn_model	
name	spawn_urdf	
args	urdf -model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description	

Para agregar submódulos o paquetes de repositorios

```
git submodule init
git submodule update
```

Correr Cartographer en simulación para el mapeo

Abriendo una nueva terminal dentro del workspace del proyecto correr:

```
source devel_isolated/setup.bash
roslaunch my_robot_description my_turtlebot_launcher. launch
```

Se inicia la simulación con el lanzamiento de un archivo para mostrar el robot integrado con sus sensores ubicado dentro del escenario simulado en 3D, todo se abarca en un mismo archivo, que necesariamente debe ser lanzado primero para identificar al modelo del robot y dar paso a la adquisición de datos registrados por los sensores según el entorno simulado.

Luego se lanza Cartographer con sus configuraciones

En otra terminal se lanza el nodo de Cartographer

```
roslaunch Cartographer _launcher Cartographer
_node_sim_maps. launch
```

Tabla 2.7 Datos del archivo launch del nodo de Cartographer

Cartographer _node_sim_maps. launch		
Argumentos	model	TURTLEBOT3_MODEL
	model type	waffle
	SLAM_methods	Cartographer
	SLAM type	Cartographer
	configuration_basename	turtlebot3_lds_pure2d_sim_maps.lua
	open_rviz	true
	file	find turtlebot3_bringup) /launch/turtlebot3_remote.launch
	model	arg model
SLAM	file	ind Cartographer _launcher)/launch/turtlebot3_$(arg SLAM_methods). launch
	model	arg model
	configuration_basename	arg configuration_basename
Rviz	node pkg	rviz
	name	rviz
	required	true
	args	find Cartographer _launcher)/rviz/turtlebot3_$(arg SLAM_methods). rviz

En otra nueva terminal se lanza el sistema para teleoperar el robot de esta forma, se mueve la base móvil mediante comandos.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key. launch
```

Cuando se consiguió mapear todo el sitio, moviendo el robot de forma teleoperada, se procede a guardar el mapa graficado en Rviz con la ayuda del paquete map_server.

Correr Cartographer en simulación para la localización / navegación

Para comprobar el proceso de localización y navegación se debe seguir los mismos pasos descritos en el apartado anterior solo cambia el archivo launch del node de Cartographer.

```
roslaunch Cartographer _launcher Cartographer
_node_sim_maps. launch
```

Para la navegación no se necesita del launch para teleoperar el robot simulado ya que se debe cargar el mapa guardado anteriormente probando la localización y la navegación del robot sobre este mapa.

2.3.2. Diseño de software para la Implementación

Así como para el software de la simulación se diseñó el software, también es necesario en la implementación tener una estructura. Para aquello es necesario instalar paquetes, crear el launcher para el nodo de Cartographer, integrar el URDF del Arlo robot, realizar la programación en micro Python para el microcontrolador ESP32, crear mensaje de ROS serial, tipo float de 16 y 32 bits para la comunicación entre la plataforma del robot y ROS Melodic. Finalmente integrar códigos para el mapeo y la navegación en el entorno interior físico.

Los pasos para construir el software para la implementación son:

Ejecutar driver RPLIDAR

Dentro de la arquitectura de la implementación se encuentra el paquete RPLIDAR que permite manejar el sensor laser RPLIDAR A1/A2 y A3. El paquete mencionado contiene un único nodo llamado rplidarNode, el cual actúa como controlador para RPLIDAR; su función es leer los datos que arroja el láser escáner haciendo uso del SDK de RPLIDAR, para transformarlo en un mensaje LaserScan de ROS. (ROS.org, 2019)

Instrucciones para correr el paquete:

1. Verificar la autoridad del puerto serial que corresponde al RPLIDAR
   ```
   ls -l /dev |grep ttyUSB
   ```
2. Añadir la autoridad de escritura denominado: (such as /dev/ttyUSB0)
   ```
   sudo chmod 666 /dev/ttyUSB0
   ```

3. Correr el nodo rpLIDAR y mostrar el resultado del escaneo en Rviz.
   `roslaunch rpLIDAR_ros view_rpLIDAR. launch`

Crear los launcher para la experimentación

En el repositorio de GitLab se crea la carpeta Cartographer _launcher, esta carpeta contiene los launch para correr la simulación y la experimentación. Los launcher son la estructura interna del sistema que contienen paquetes, nodos, Tópicos y parámetros, permitiendo ejecutar varios nodos ROS de manera local y también remota por medio de SSH. Dentro de la carpeta launch se alojan los tres launch para la experimentación, sus nombres son:

- Cartographer _node_exp_maps. launch
- Cartographer _node_exp_nav. launch
- Cartographer _node_exp_nav_amcl. launch

El primer launch contiene el nodo de Cartographer que genera un mapa del entorno interior por medio del sensor RPLIDAR. El segundo launch carga el robot en el mapa generado y permite navegar de manera autónoma en ella utilizando el paquete move_base que se encuentra dentro de este launch. El tercer launch contiene el paquete amcl que permite ubicar al robot en la poción actual y obtener su localización.

Procedimiento para crear el fichero Cartographer _node_exp_maps.launch:

1. Crear un directorio llamado launch con el comando **mkdir** launch
2. Acceder al directorio con el comando **cd** launch
3. Crear el fichero Cartographer _node_exp_maps.launch
4. Abrir y editar el texto del fichero con el comando **gedit** Cartographer _node_exp_maps.launch, en donde se agregan los múltiples nodos a ejecutar, estos pueden publicar o suscribir. Además, se procede a editar los nombres del paquete, el tipo de nodo y el nodo a ejecutar. Las configuraciones realizadas se encuentran en el siguiente enlace map
5. Ejecutar con el comando **roslaunch** Cartographer _node_exp_maps.launch en el terminal de ROS.

Procedimiento para crear el fichero Cartographer _node_exp_nav.launch:

1. Crear un directorio llamado launch con el comando **mkdir** launch
2. Acceder al directorio con el comando **cd** launch
3. Crear el fichero Cartographer _node_exp_nav.launch
4. Abrir y editar el texto del fichero con el comando **gedit** Cartographer _node_exp_nav.launch, en donde se agregan los múltiples nodos a ejecutar, estos pueden publicar o suscribir. Además, se procede a editar los nombres del paquete, el tipo de nodo y el nodo a ejecutar. Las configuraciones realizadas se encuentran en el siguiente enlace nav
5. Ejecutar con el comando **roslaunch** Cartographer _node_exp_nav.launch en el terminal de ROS.

Procedimiento para crear el fichero Cartographer _node_exp_nav_amcl.launch:

1. Crear un directorio llamado launch con el comando **mkdir** launch
2. Acceder al directorio con el comando **cd** launch
3. Crear el fichero Cartographer _node_exp_nav_amcl.launch
4. Abrir y editar el texto del fichero con el comando **gedit** Cartographer _node_exp_nav_amcl.launch, en donde se agregan los múltiples nodos a ejecutar, estos pueden publicar o suscribir. Además, se procede a editar los nombres del paquete, el tipo de nodo y el nodo a ejecutar. Las configuraciones realizadas se encuentran en el siguiente enlace amcl
5. Ejecutar con el comando **roslaunch** Cartographer _node_exp_nav_amcl.launch en el terminal de ROS.

Integrar URDF del Arlo Robot

El URDF contiene la representación del modelo del robot en formato XML con su analizador correspondiente. Se integra el paquete common_chrisl8.urdf.xacro tomado de GitLab (Lofland, 2019) al árbol de la implementación , ya que contiene la descripción en XML del modelo Arlo robot, y de los sensores que están integradas en ella. Este archivo se encuentra en el siguiente enlace urdf.

Descripción en XML del argumento integrado en los launcher de la implementación:

```
<arg name="urdf_file" default="xacro '$(find Arlobot_description) /urdf/common_default.urdf.xacro'" />
```

Programar el microcontrolador ESP32 y crear el mensaje ROS serial float 32 bits, int 16 bits.

El desarrollo del programa en micro Python permite comunicar el controlador ESP32 con los encoders para adquirir datos de los pasos por metro y asignar velocidades a las llantas de la plataforma robótica, y así producir el movimiento en el robot. Los mensajes creados en ese tipo de datos y el número de bits son imprescindibles para comunicar los nodos que intervienen en la comunicación serial.

Realizar código de mapeo mainTeleop.

El desarrollo del programa en micro Python permite establecer velocidades a la plataforma del robot a través de los encoders permitiendo controlar al robot de manera manual por medio del virtual joystick, el cual es un nodo que forma parte del paquete differential drive.

Realizar código de navegación mainNav.

El desarrollo del programa en micro Python permite establecer velocidades a la plataforma del robot a través de los encoders permitiendo que el robot pueda navegar dentro del entorno físico, luego de asignar una trayectoria al robot a través de la herramienta 2D Nav Goal.

Cinemática del robot diferencial integrado en el paquete differential_drive

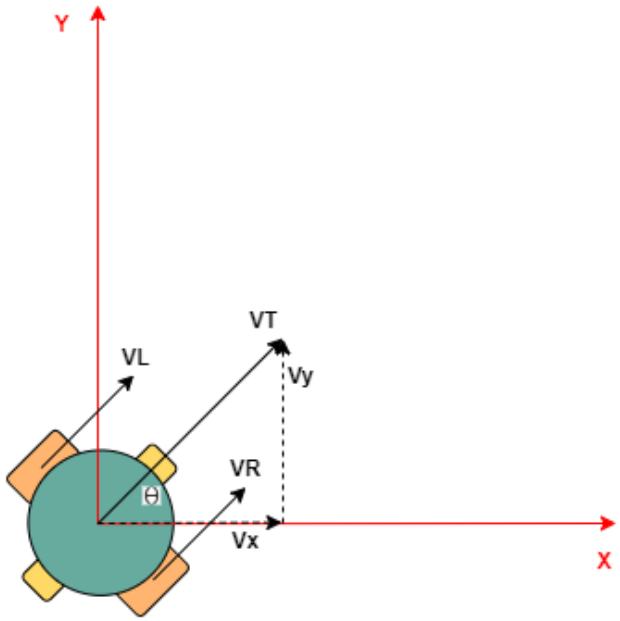


Figura 2.27 Análisis de la velocidad lineal del Arlo Robot en un plano 2D

Establecer un modelo cinemático para la plataforma Arlo robot, que cumple con las características de un robot móvil diferencial no holónomico es imprescindible para el control del movimiento de la plataforma. Para aquello se realiza el siguiente análisis de vectores de velocidad a través de ecuaciones matemáticas en base a la figura 2.23

Análisis para hallar las expresiones de las velocidades lineales

$$Vt = \frac{Vr + Vl}{2} \tag{2.11}$$

$$Vx = Vtcos\theta \tag{2.12}$$
$$Vy = Vtsen\theta$$

$$Vx = \frac{Vr + Vl}{2} cos\theta \tag{2.13}$$

$$Vy = \frac{Vr + Vl}{2} sen\theta$$

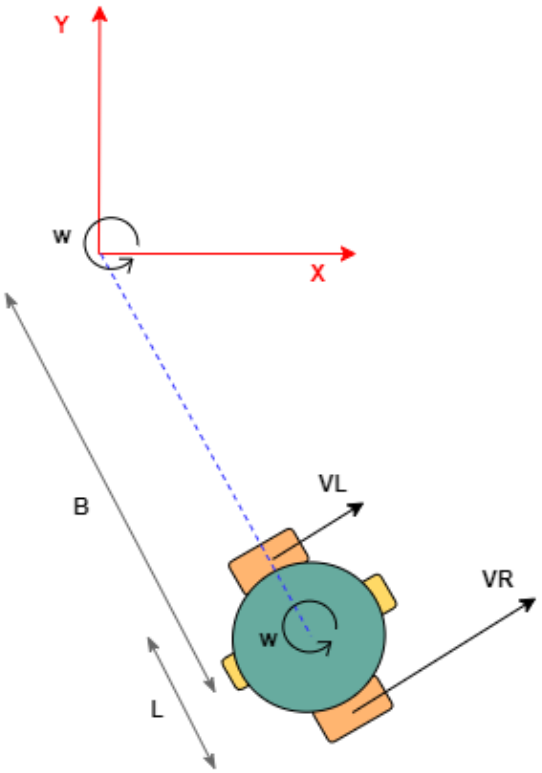


Figura 2.28 Análisis de la velocidad angular del Arlo Robot en un plano 2D

Análisis para hallar la expresión matemática de la velocidad angular

$$v = wr; dondeVl = w\left(B - \frac{L}{2}\right) \quad (2.14)$$

$$Vr = w\left(B + \frac{L}{2}\right) \quad (2.15)$$

$$B = \frac{\left(Vl + \frac{wL}{2}\right)}{w} \quad (2.16)$$

$$Vr = w\left(\frac{Vl + \frac{wL}{2}}{w} + \frac{L}{2}\right) \quad (2.17)$$

$$Vl = w\left(\frac{Vl + \frac{wL}{2}}{w} - \frac{L}{2}\right) \tag{2.18}$$

$$Finalmente, w = \frac{Vr - Vl}{L} \tag{2.19}$$

En el modelo cinemático desarrollado, las variables de entrada del sistema son las velocidades de las llantas de tracción izquierda y derecha. Las variables de salida son las velocidades lineales y angular de la plataforma Arlo robot en la posición correspondiente con respecto al sistema referido.

Las expresiones matemáticas antes encontradas son un poco complejas para introducirlas en el paquete differential_drive que contiene los nodos y Tópicos para el control del movimiento de la plataforma Arlo robot; por esta razón se simplifico el análisis de las velocidades de las llantas de tracción a una sola velocidad lineal que mueve la plataforma. Ahora las variables de entrada serán la velocidad lineal y rotacional de la plataforma; y las variables de salida serán las velocidades de la llanta izquierda y derecha, permitiendo observar a través del terminal estos cambios de velocidades de cada llanta.

A partir de las siguientes ecuaciones:

$$V = \frac{Vr + Vl}{2}$$
$$w = \frac{Vr - Vl}{L} \tag{2.20}$$

Se desarrolla algebraicamente, para obtener la expresión matemática utilizada en la programación.

$$Vl = V - \frac{wL}{2}$$
$$Vr = V + \frac{wL}{2} \tag{2.21}$$

Las ecuaciones 2.11 representan de manera matemática el comportamiento cinemático del robot móvil diferencial.

Costo de los recursos de Hardware

Tabla 2.8 Análisis de costo de dispositivos que integran el sistema

ID de producto	Items	Cantidad	Precio unitario	Subtotal
28231	DHB-10 Dual H-Bridge Motor Controller	1	$ 49.99	$ 49.99
28960	Arlo Base Kit	1	$ 39.00	$ 39.00
28961	Caster Wheel Kit	2	$ 49.00	$ 98.00
28962	Motor Mount & Wheel Kit – Aluminum	1	$ 299.00	$ 299.00
28965	Arlo Top Deck	1	$ 49.00	$ 49.00
28996	Arlo Power Distribution System	1	$ 99.99	$ 99.99
32201	Propeller Plug programming tool	1	$ 14.99	$ 14.99
32912	Propeller Activity Board WX	1	$ 79.99	$ 79.99
570-35000	Hardware Pack for Arlo	1	$ 9.99	$ 9.99
700-00245	Arlo Battery Charger - 2.1 mm Plug	1	$ 31.47	$ 31.47
752-00007	Battery, 12-volt, 7.2 amp-hour, sealed lead acid	1	$ 24.99	$ 24.99
A1M8	RPLIDAR A1	1	$ 99.99	$ 99.99
29321	36-Position Quadrature Encoder Set	1	$ 39.99	$ 39.99
3-01-1287	HiLetgo ESP-WROOM-32	1	$10.99	$10.99
FMP05B-i7-4500U/5500U	Mini PC	1	$ 745.45	$ 745.45
	Monitor	1	$ 100	$ 100.00
	Total			$ 1792.83

Nota. Fuente: Arlo Complete Robot System v1.2 del fabricante PARALLAX INC, Amazon y OLX Ecuador

El precio de los dispositivos que integran el hardware del sistema es presentado a través de la tabla 2.8, donde se detallan los precios por unidad y su valor total dependiendo de la cantidad. Al final se muestra el precio total por la adquisición

de todos los dispositivos. Presentar el costo total permite que otros usuarios tengan un conocimiento previo de la inversión, en el caso de recrear o mejorar el prototipo presentado en este proyecto; sin embargo, los costos de software no son presentados porque son plataformas libres y de código abierto.

CAPÍTULO 3

3. RESULTADOS

En la presente sección se detallarán y analizarán los resultados obtenidos de simular e implementar la técnica SLAM para la navegación autónoma de un robot móvil. En la simulación e implementación los resultados se ven reflejados en los siguientes ítems: mapeo, localización, planificador de trayectoria y seguimiento de trayectoria.

3.1. Simulación de la técnica SLAM para la navegación autónoma.

Mapeo

A través de un diagrama y gráficos se mostrará el proceso para mapear un entorno interior recreado en Onshape y simulado en Gazebo; así también el mapa final es visualizado en Rviz. Al final se evaluará el mapa en base a métricas de evaluación.

Diagrama del proceso de mapeo simulado

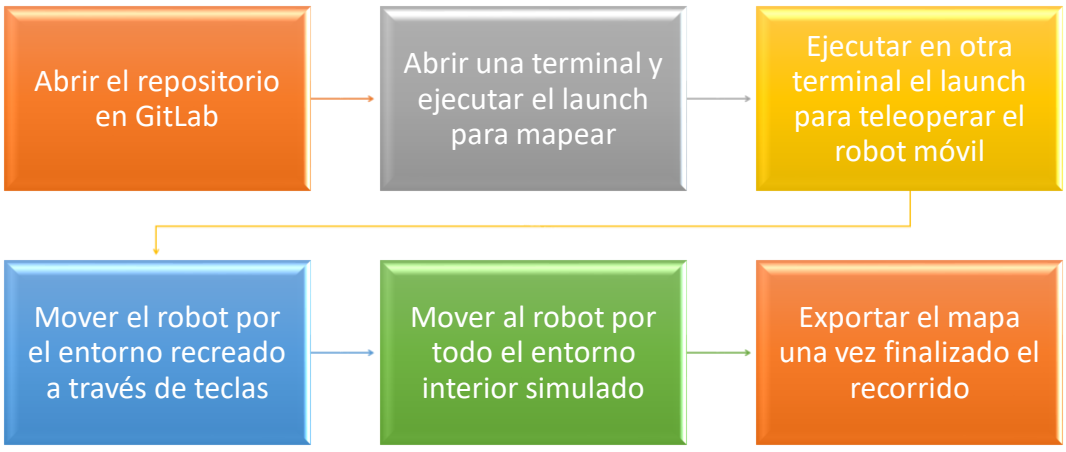


Tabla 3.9 Diagrama del proceso de mapeo simulado

El diagrama muestra el proceso a seguir para obtener el mapa que será luego utilizado en la navegación autónoma del robot móvil Turtlebot 3.

A continuación, se presenta mediante ilustraciones el inicio del mapeo, el desarrollo del mapa y finalmente el dibujo del mapa realizado en Rviz.

Gráficas del proceso de mapeo simulado

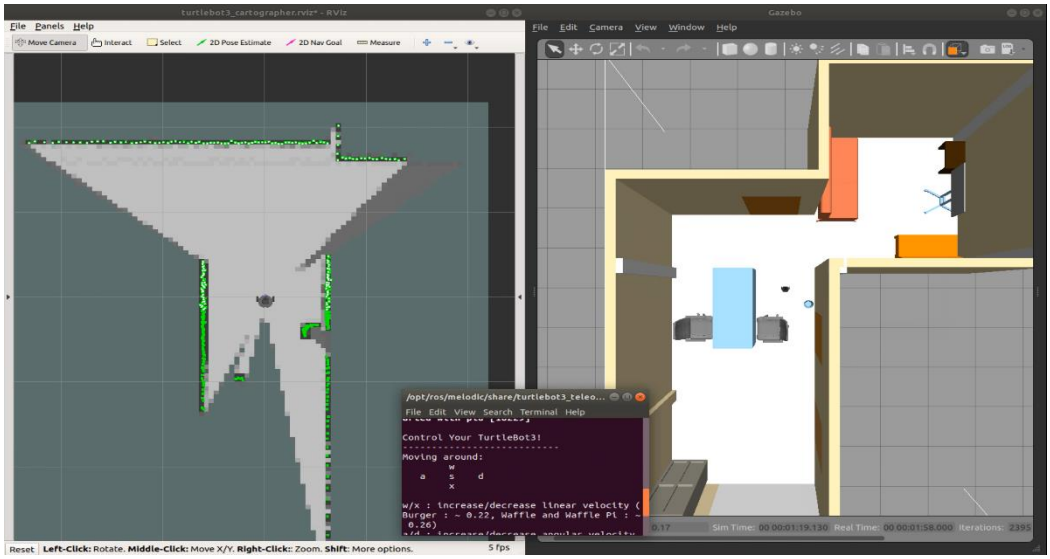


Figura 3.29 Inicio del proceso de mapeo

La figura 3.2 muestra al robot móvil Turtlebot 3 ubicado en el interior del entorno simulado en Gazebo. El launch catographer_node_sim_maps.launch permite ejecutar los múltiples nodos integrados en este launch, en consecuencia se abre el simulador de entorno 3D Gazebo y la herramienta de visualización Rviz.

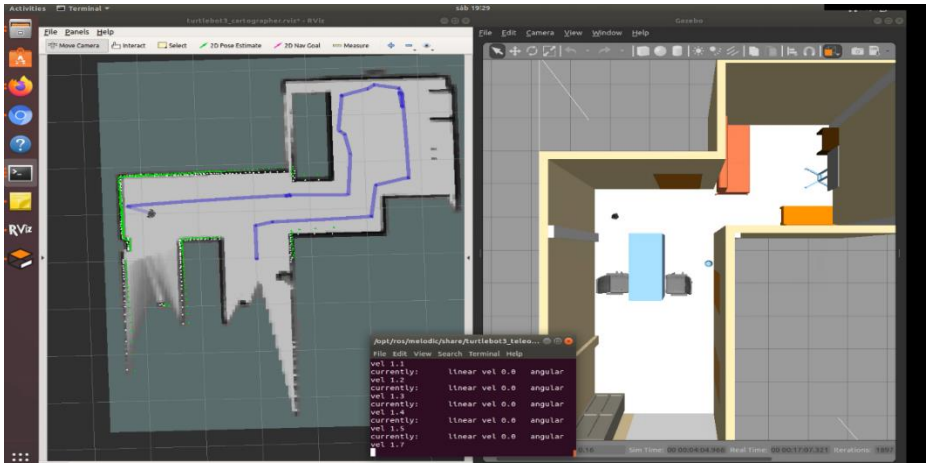


Figura 3.30 Desarrollo del proceso de mapeo de la sala.

La figura 3.3 muestra al robot móvil Turtlebot 3 recorriendo el entorno simulado y su vez la trayectoria marcada por el trazo de color lila. Para mover el robot de forma manual a través del entorno simulado se tuvo que ejecutar el launch turtlebot3_teleop_key.launch.

Figura 3.31 Desarrollo del proceso de mapeo del comedor

La figura 3.4 muestra al robot móvil Turtlebot 3 recorriendo el área del comedor y su movilidad manual se ejecutó a través de las teclas a, d, w, x y s, donde la tecla **a** representa giro hacia la izquierda, la tecla **d** giro hacia la derecha, la tecla **w** movimiento hacia adelante, la tecla **x** movimiento hacia atrás y la tecla **s** detiene el movimiento del robot.

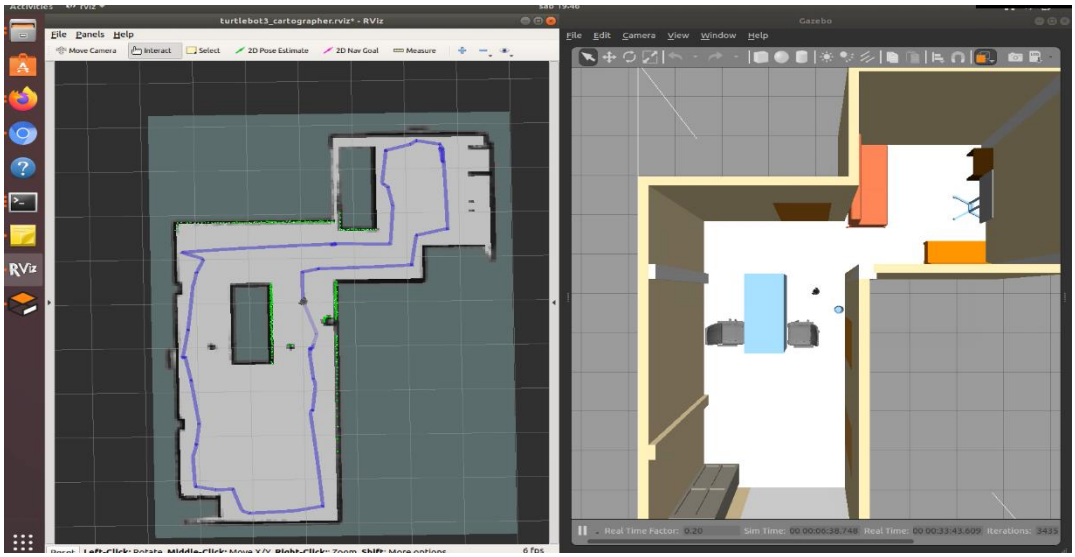
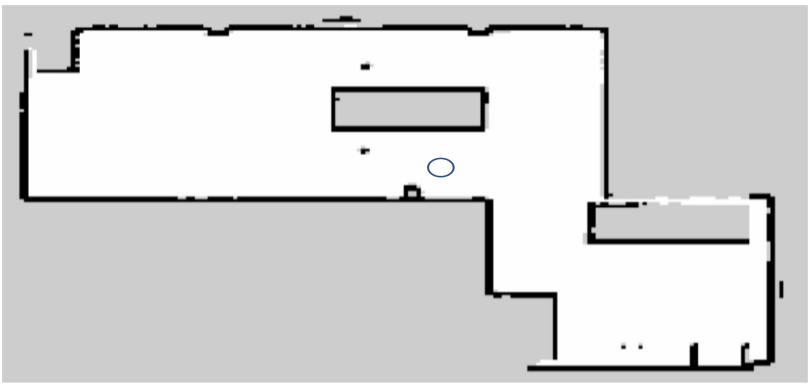


Figura 3.32 Finalización del proceso de mapeo.

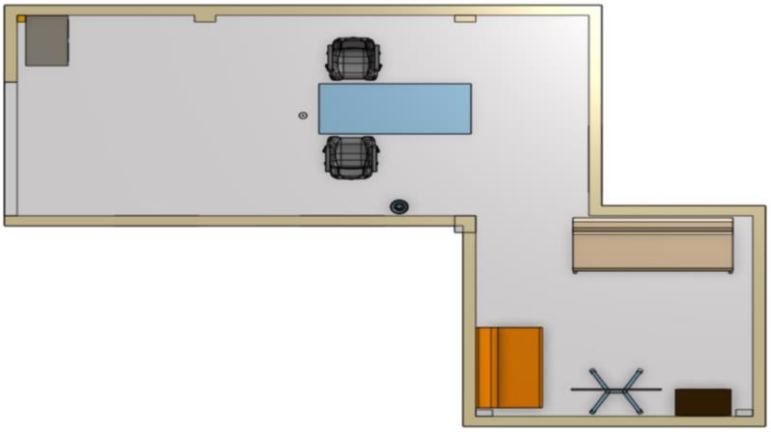
La figura 3.5 muestra al robot móvil Turtlebot 3 detenido en el punto donde empezó su trayectoria y el trazo lila representa todo el recorrido realizado dentro del entorno simulado. Una vez completado la construcción del mapa se procede a exportar el mismo para que el robot móvil navegue de manera autónoma en el utilizando la herramienta 2D Nav Goal, que sirve para fijar metas u objetivos, de esta forma el robot ejecute el planeo de la trayectoria que lo lleve a su destino final de forma óptima.

Evaluación del mapa simulado

La evaluación de forma visual de la calidad de la imagen del mapa obtenido a través de la técnica de SLAM Cartographer, se basa en las siguientes métricas: proporción de celdas ocupadas y libres, cantidad de esquinas en un mapa y cantidad de áreas cerradas. (A. Filatov, 2017, pág. 121)



a) Mapa de la simulación realizado por la técnica Cartographer SLAM.



b) Entorno de simulación realizado en OneShape

Figura 3.33 Comparación entre el mapa obtenido (a) y el entorno de simulación (b)

Proporción de celdas ocupas y libres

La métrica analiza el desenfoque, efecto borroso, líneas paralelas que se puedan hallar en las paredes que delimitan el entorno de navegación, así como objetos o artefactos. En la matriz 2D que es representada por una cuadrícula de ocupación, el algoritmo se encarga de asignar a cada celda la probabilidad de estar ocupada, donde las celdas de color blanco representan celdas libres y las celdas color oscuro son las celdas ocupadas.

Partiendo de este análisis, las paredes del mapa obtenido en un gran porcentaje están bien definidas y parecidas al mapa estimado, a pesar de que existen algunos espacios de la pared incompletos, y dos líneas paralelas, producto del error del algoritmo SLAM en la toma de datos como muestra la figura 3.7 En general las paredes y objetos dentro del mapa no poseen efectos borrosos, ni desenfoque y la proporción de celdas ocupadas es menor en comparación con las celdas libres; por lo tanto, la calidad del algoritmo es mayor.

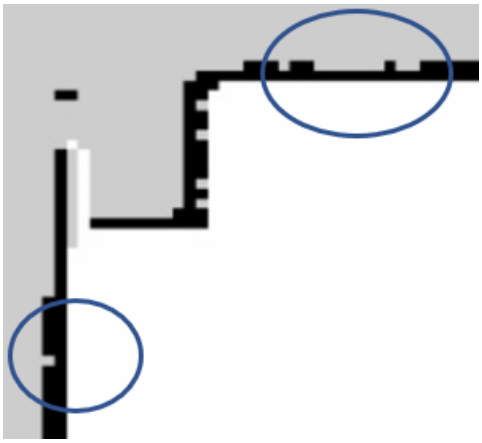


Figura 3.34 Proporción de celdas ocupadas y libres en el mapa generado por la técnica SLAM Cartographer.

Cantidad de esquinas en un mapa

La métrica analiza la cantidad de esquinas falsas que se produjeron en el mapa, paredes duplicadas, curvaturas erróneas que no forman parte del modelo original.

La cantidad de esquinas falsas es mínima, pero si existen inconsistencia en la forma de algunas de las esquinas reales que forman parte del entorno de simulación, debido a que ciertos datos no fueron tomados por el algoritmo

mientras se realizaba la creación del mapa como muestra la figura 3.8 En general hubo una cantidad mínima de esquinas falsas, lo que indica que la calidad del mapa es aceptable.

Figura 3.35 Cantidad de esquinas falsas e indefinidas en el mapa generado por la técnica SLAM Cartographer.

Cantidad de áreas cerradas

La última métrica consiste en la cantidad de regiones dentro del mapa que es encuentran delimitadas por una pared que rodea un espacio abierto. Las celdas ocupadas e indefinidas actúan como parte de una limitación de una región específica.

La cantidad de área encerrada es mayor, aunque dos regiones que pertenece a un mueble y a un ropero no se encuentran totalmente delimitadas, debido a que el robot móvil no pudo acceder a esos espacios reducidos como muestra la figura 3.8 En general la cantidad de áreas encerradas es mayor y toda la región que conforma el entorno de simulación se encuentra delimitado por las paredes; por lo tanto, la calidad del mapa es alta.

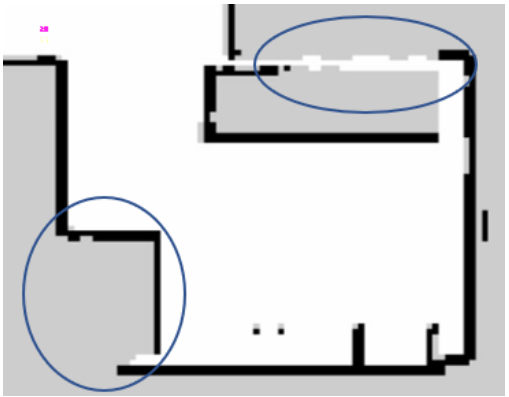


Figura 3.36 Cantidad de áreas encerradas en el mapa generado por la técnica SLAM Cartographer.

Localización

A través de un diagrama y gráficos se mostrará el proceso para localizar al robot móvil y establecerlo en su ubicación actual dentro del mapa generado por el algoritmo de Cartographer. Este proceso podrá ser observado a través de la herramienta de visualización Rviz.

Diagrama del proceso de localización

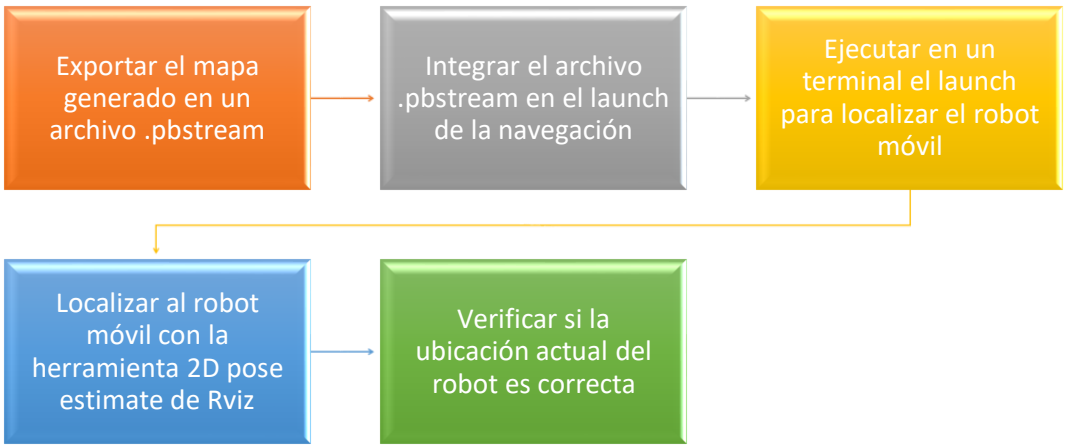


Figura 3.37 Diagrama del proceso de localización

El proceso de la localización es dependiente de un proceso previo, porque es necesario contar con el mapa generado, consecuencia de la exploración del robot móvil Turtlebot 3 por el entorno simulado de forma manual.

A continuación, se muestra el proceso para la estimación de la ubicación actual del robot móvil Turtlebot 3.

Gráficas del proceso de localización

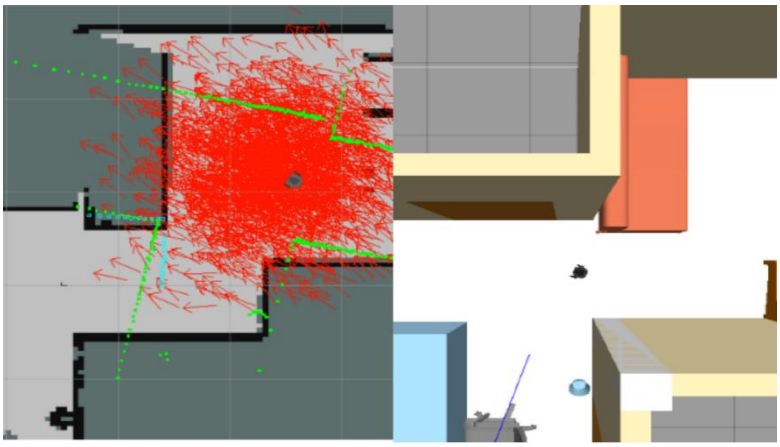


Figura 3.38 Inicio de la localización del robot móvil Turtlebot 3

La figura 3.10 muestra en el lado izquierdo al robot dentro del mapa que ha sido cargado en Rviz ubicado de forma aleatoria y en el lado derecho la ubicación actual del robot dentro del entorno de simulación en Gazebo. Al correr el launch Cartographer _node_sim_nav_amcl.launch se ejecturán los múltiples nodos que permitirán la localización gracias al paquete amcl de ROS.

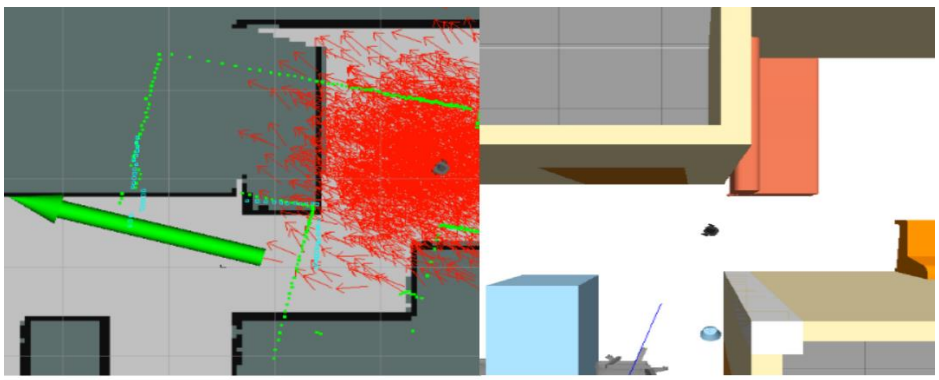


Figura 3.39 Desarrollo de la localización del robot móvil Turtlebot 3.

La figura 3.11 muestra el uso de la herramienta **2D pose estimate** para ubicar al robot móvil Turtlebot 3 en la posición actual dentro del mapa generado.

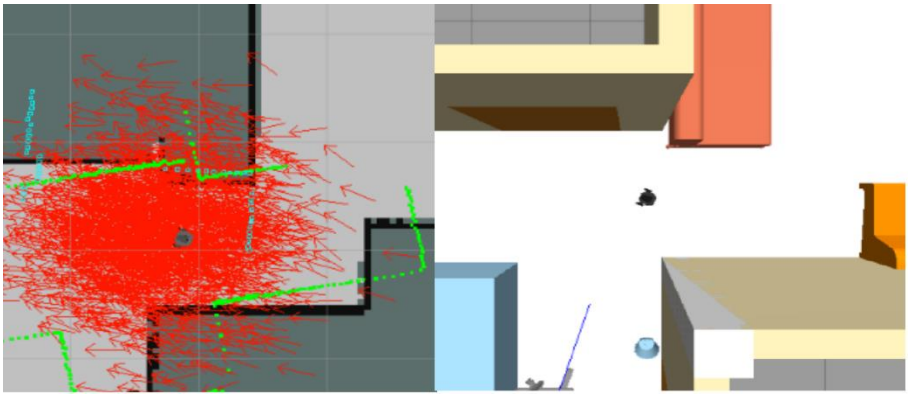


Figura 3.40 Localización actual del robot móvil Turtlebot 3

La figura 3.12 muestra la estimación de la ubicación actual de robot móvil dentro del mapa generado. La ubicación correcta del robot se sustenta en la coincidencia entre el contorno del mapa y el área de escaneo que se encuentra delimitado por la franja verde. Se evidencia en la imagen que aún no se ha hecho una correcta estimación de la pose del robot, debido a lo cual no coincide el mapa y el área escaneada.

Para que se efectúe el ajuste de las áreas se utiliza la herramienta de Rviz llamada 2D Nav Goal, representado por la flecha color purpura, que permite enviar un punto como meta para que sea alcanzado por el robot, de esta manera se pone en evidencia la ejecución de la planificación dentro de la navegación y a medida que el robot navegue automáticamente, gracias al paquete de amcl, se reajustan la ubicaciones del robot tanto en gazebo, el entorno de referencia y la ubicación estimada por el usuario dentro de Rviz, coincidiendo así el mapa cargado y las franjas verdes del escaneo del sensor LIDAR en tiempo real.

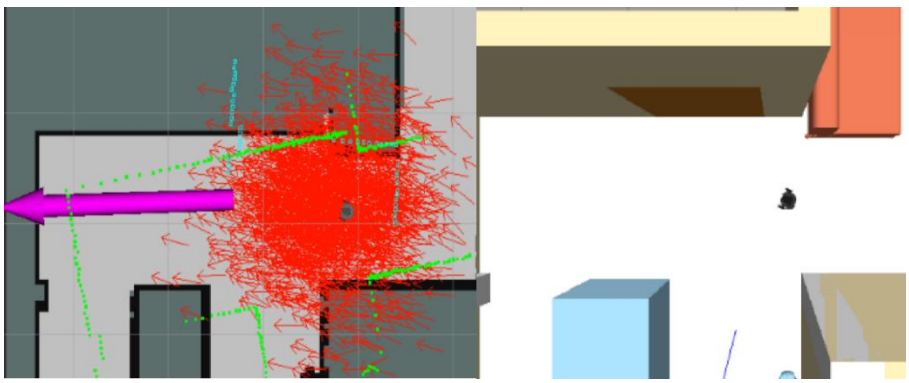


Figura 3.41 Probando el reajuste de la pose del robot

Evaluación de la localización

La evaluación de forma visual y numérica de la ubicación correcta del robot móvil dentro del mapa obtenido y visualizado en Rviz, se basa en las siguientes métricas: coincidencia del borde del mapa y comparación de la posición actual del robot móvil.

Coincidencia del borde del mapa

La métrica se basa en que el área de escaneo delimitado por el sensor laser representado por las franjas verdes debe coincidir con el esquema del mapa, esto demuestra que la ubicación del robot móvil dentro del mapa en Rviz es correcto.

La figura 3.14 muestra al robot móvil ubicándose en la posición correcta, luego de haberse desplazado para alcanzar un punto objetivo. Ahora la franja de escaneo se ajusta al esquema del mapa en un gran porcentaje, es decir, la estimación de la localización produce un error mínimo.

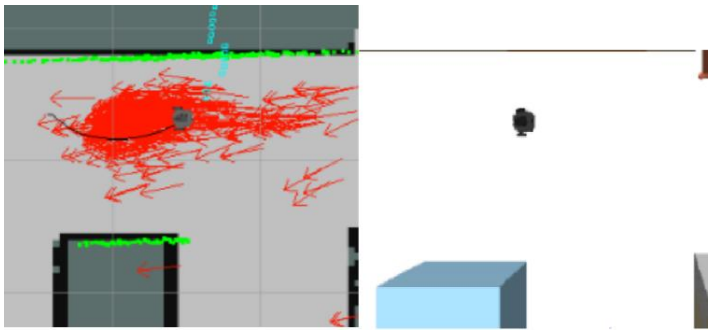


Figura 3.42 Estimación de la posición actual del robot móvil

Planificador de trayectoria

La figura 3.15 muestra el proceso para la planificación de la trayectoria del robot móvil dentro del mapa generado por la técnica de Cartographer SLAM que es observado por medio de la herramienta de visualización Rviz.

Con la integración del paquete dwa_local_planer de ROS es posible conectar el planificador de rutas con el robot móvil. El planificador de rutas crea una trayectoria cinemática directa desde la posición de partida hasta la posición objetivo de llegada.

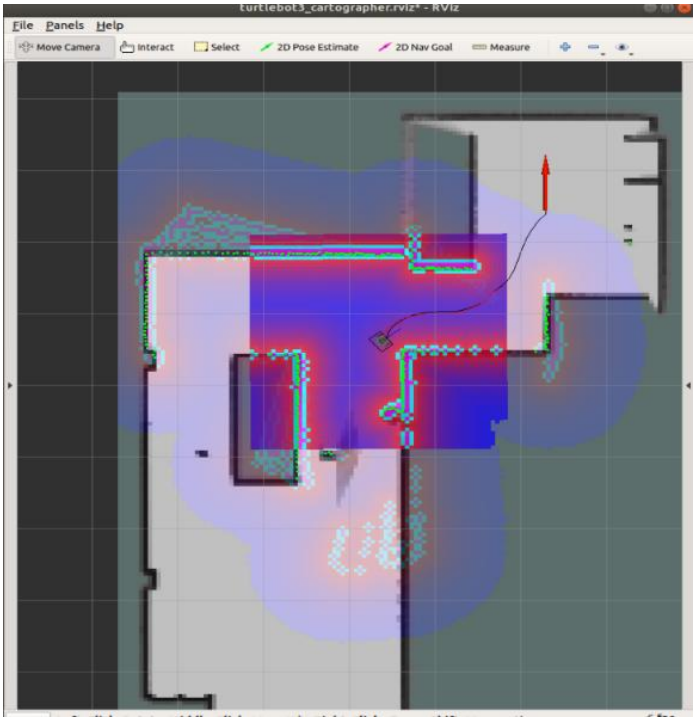


Figura 3.43 Planificación de una trayectoria cinemática

Evaluación de la planificación de la trayectoria

Luego de trazar varias trayectorias se pudo apreciar que el planificador de trayectorias toma la ruta más corta y directa, sin la necesidad de utilizar otro paquete de ROS que permita ajustar las trayectorias, ya que el planificador las genera de manera aleatoria.

Seguimiento de trayectoria

El proceso de seguimiento de una trayectoria o navegación autónoma del robot móvil y evasión de obstáculos se muestra a través de gráficas.

Con la integración del paquete move_base de ROS es posible que el robot navegue de forma autónoma a través del mapa, en conjunto con la herramienta 2D Nav Goal que marca la posición de llegada del robot móvil.

Gráficas del seguimiento de trayectoria

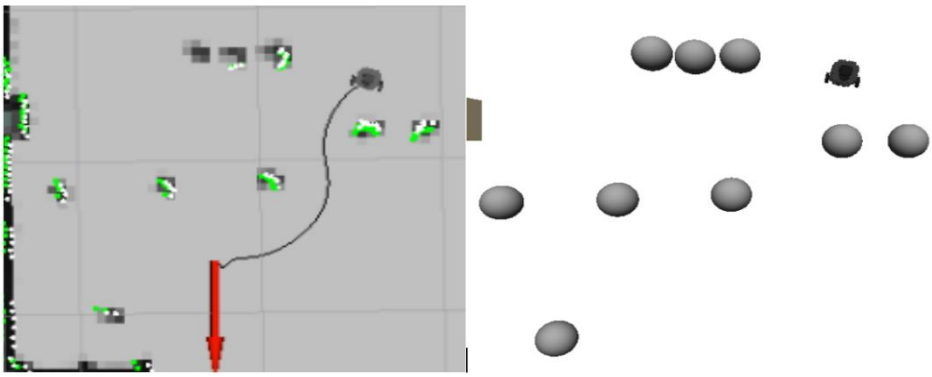


Figura 3.44 Inicio de la trayectoria asignada por el planificador de trayectorias

La figura 3.16 muestra la ruta que debe recorrer el robot móvil Turtlebot 3 a través del mapa. En el entorno de simulación se agregaron objetos que actúan como obstáculos y se ven reflejados en la herramienta de visualización Rviz.

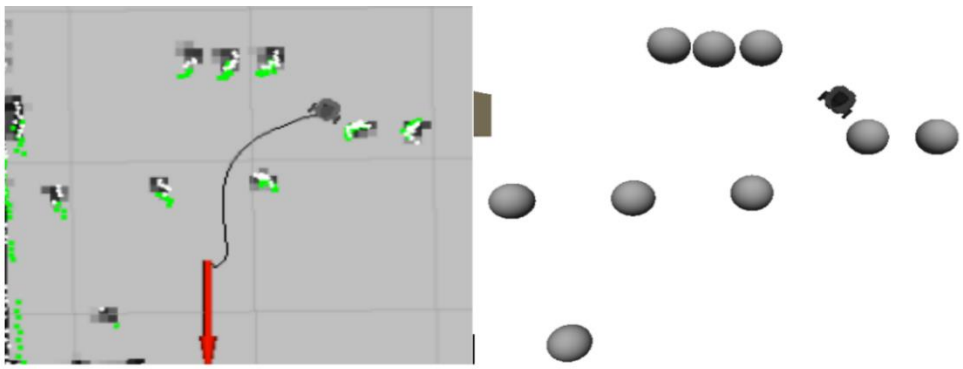


Figura 3.45 Robot móvil evadiendo obstáculos

La figura 3.17 muestra al robot móvil iniciando el seguimiento de la trayectoria, donde a medida que avanza se encuentra con los objetos que simulan los obstáculos. El robot móvil evade el obstáculo y esto provoca que el planificador asigne una nueva trayectoria al robot.

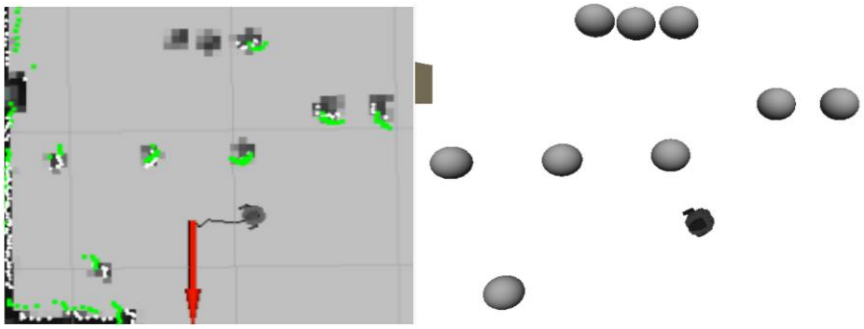


Figura 3.46 Robot móvil llegando a la posición objetivo

La figura 3.18 muestra al robot móvil llegando a la posición final, donde previamente encontró otro obstáculo y nuevamente el planificador tuvo que cambiar la trayectoria. Finalmente, el robot llego a la posición final sin ninguna dificultad.

Evaluación del seguimiento de trayectoria

Luego de varias pruebas se verificó que el robot móvil evadió con facilidad los obstáculos y no existió ninguna colisión entre el robot y los objetos que se encontraban dentro del entorno de simulación; así también, el planificador de rutas se iba actualizando a medida que el robot móvil iba avanzando, tomando la mejor trayectoria para que él llegue a su posición final sin ningún problema. La distancia entre la posición final y la posición a la que llegó el robot móvil tiene una tolerancia de x(+0.10,-0.00), es decir, entre ambas posiciones existe una diferencia de 10 cm.

En base a la simulación se comprobó que la asignación de varias posiciones objetivos dentro del mapa es posible, permitiendo que el robot pueda moverse de manera libre por toda el área.

3.2. Implementación de la técnica SLAM para la navegación autónoma.

Mapeo

A través de un diagrama y gráficos se mostrará el proceso para mapear un entorno interior físico conformado por la sala y el comedor de $9.84m^2$ y $20.31m^2$ respectivamente; así también el mapa final es visualizado en Rviz. Al final se evaluará el mapa en base a métricas de evaluación.

Diagrama del proceso de mapeo experimental

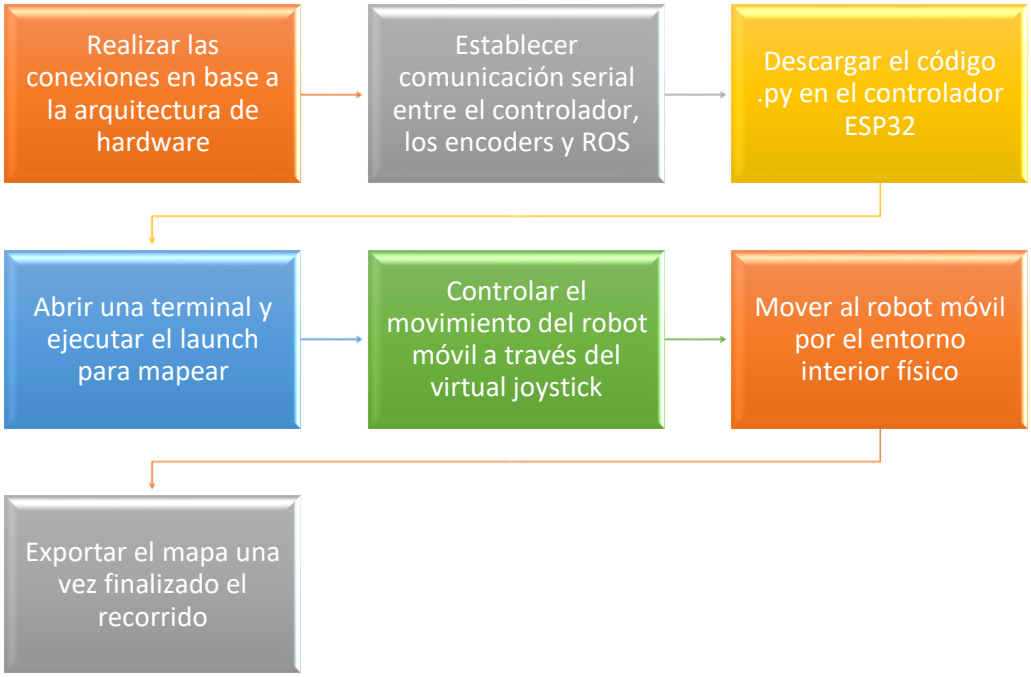


Figura 3.47 Diagrama del proceso de mapeo experimental

Mediante el diagrama se muestra el procedimiento paso a paso para construir el mapa del entorno. Los paquetes, tópicos, nodos que integran el launch para crear el mapa se encuentran en un repositorio creado en GitLab, así también, los archivos antes mencionados se encuentran en la mini PC. Un paso clave es el ajuste físico de la plataforma Arlo robot y los sensores, donde inicialmente se verifica de manera individual el estado de los equipos, el funcionamiento, la precisión de medición, alimentación eléctrica, entre otros. En la ejecución del

software, la revisión del árbol de programación construido con anterioridad es otro paso importante antes de ejecutar el procedimiento para la construcción del mapa.

Los resultados obtenidos del mapeo se obtuvieron luego de varias pruebas y el proceso de la creación del mapa y su mapa final se ven reflejados a través de gráficas.

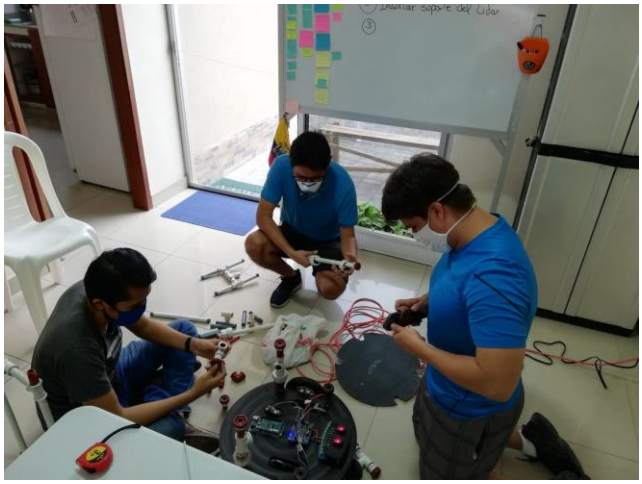


Figura 3.48 Ensamble de la plataforma Arlo robot

Gráficas del proceso de mapeo experimental

La figura 3.19 muestra el proceso de ensamble de la plataforma del robot móvil para colocar los sensores, batería y otros dispositivos que hacen posible el movimiento del Arlo robot y adquisición de datos para el mapeo.

Figura 3.49 Estado final de la plataforma Arlo robot.

La figura 3.20 muestra el ensamble final de la plataforma Arlo robot, donde se encuentran instalados los siguientes dispositivos: controlador ESP32-WROOM, sensor RPLIDAR A1, encoders, DHB-10, mini PC, batería para la plataforma y la estructura de la plataforma Arlo robot. Una vez finalizado esta etapa, se procede a establecer la comunicación serial entre el controlador y los sensores, el software ROS; así también se descarga en el controlador el programa realizado en micro Python **main.py**, que se encuentra también en el repositorio para el manejo de los encoders y el movimiento de la plataforma del robot móvil.

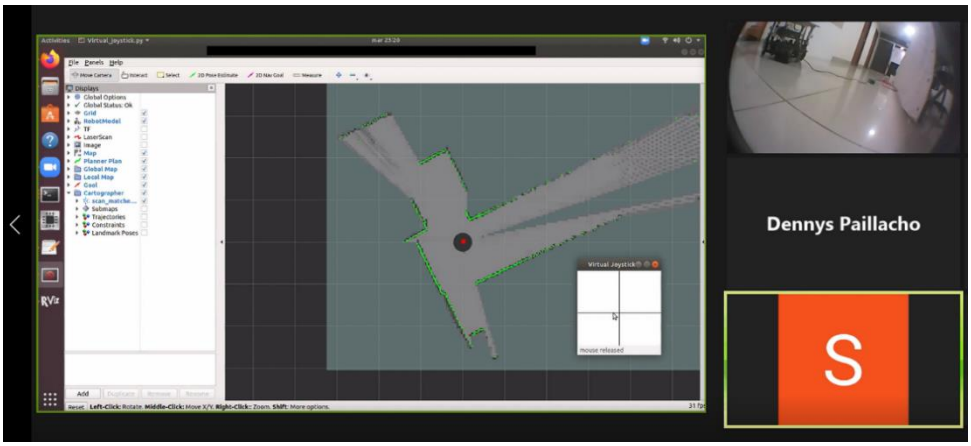


Figura 3.50 Software y hardware combinados para el desarrollo del mapeo

La figura 3.22 muestra a la plataforma Arlo robot preparada dentro del entorno físico para el inicio del mapeo, donde a través de la herramienta de visualización Rviz se observa la ubicación dentro del entorno. En la gráfica también se observa la interfaz gráfica de virtual joystick para mover de manera manual el robot móvil a través del entorno interior. El launch utilizado para mapear y operar al robot móvil de manera manual través del entorno es **Cartographer _node_exp_maps.launch**.

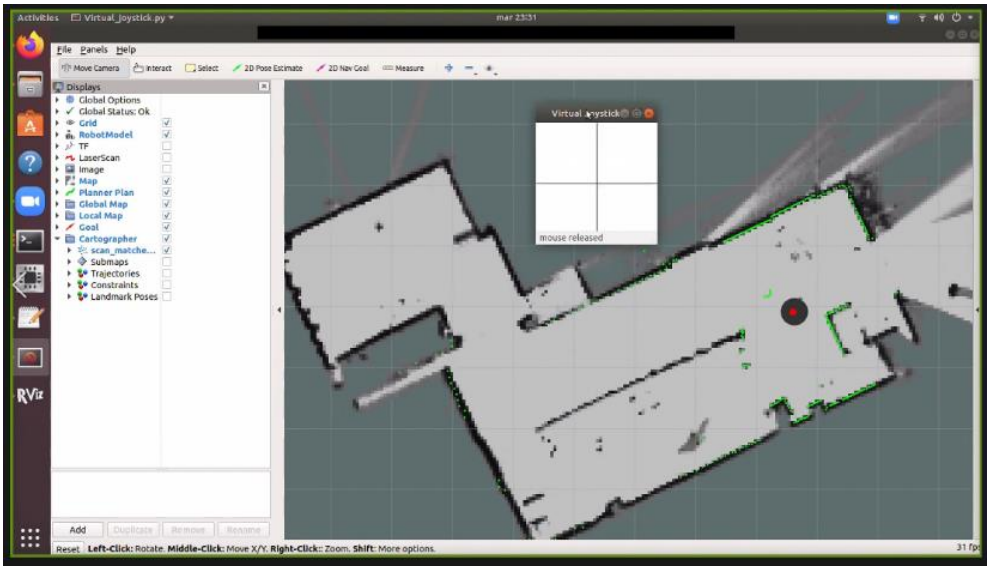
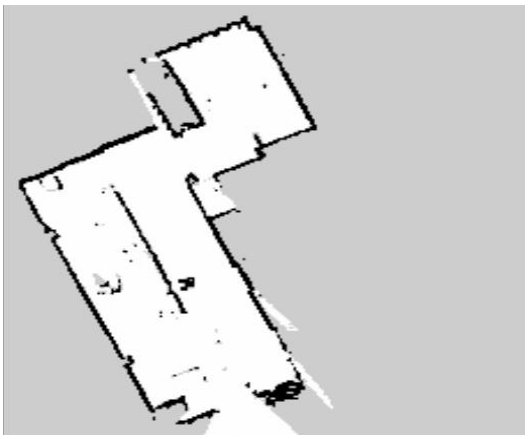


Figura 3.51 Finalización de la construcción del mapa.

La figura 3.23 muestra la construcción final del mapa correspondiente al entorno interior físico, producto de mover la plataforma Arlo robot por todo el área a través de la interfaz gráfica virtual joystick. Al finalizar esta etapa se procede a exportar el mapa en un archivo **.Pbstream** para utilizarlo en el proceso de la localización, planificación de trayectoria y seguimiento de la trayectoria.

Evaluación del mapa experimental

Al igual que en la simulación, la evaluación del mapa experimental es de forma visual para analizar la calidad de imagen del mapa obtenido a través de la técnica de SLAM Cartographer. Los criterios de evaluación se basan en las siguientes métricas: proporción de celdas ocupadas y libres, cantidad de esquinas en un mapa y cantidad de áreas cerradas. (A. Filatov, 2017, pág. 121)



Mapa experimental

(b) entorno físico

Figura 3.52 Mapa experimental obtenido(a) y entorno interior físico (b).

Proporción de celdas ocupas y libres

La métrica analiza el desenfoque, efecto borroso, líneas paralelas que se puedan hallar en las paredes que delimitan el entorno de navegación, así como objetos o artefactos. En la matriz 2D que es representada por una cuadrícula de ocupación, el algoritmo se encarga de asignar a cada celda la probabilidad de estar ocupada, donde las celdas de color blanco representan celdas libres y las celdas color oscuro son las celdas ocupadas.

Partiendo de este análisis, las paredes del mapa obtenido en un 70 % están definidas y parecidas al mapa estimado, a pesar de que existen algunos espacios de la pared incompletos y espacios libres fuera del límite del entorno, producto del error del algoritmo SLAM y del sensor RPLIDAR A1 en la toma de datos como muestra la figura 3.25. En general las paredes y objetos dentro del mapa no poseen efectos borrosos, poco desenfoque y la proporción de celdas ocupadas es menor en comparación con las celdas libres, pero existen espacios libres que no corresponden al entorno; por lo tanto, la calidad del mapa es media.

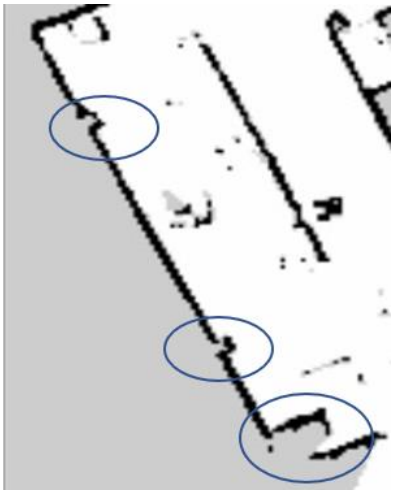


Figura 3.53 Proporción de celdas ocupadas y libres en el mapa experimental generado por la técnica SLAM Cartographer.

Cantidad de esquinas en un mapa

La métrica analiza la cantidad de esquinas falsas que se produjeron en el mapa, paredes duplicadas, curvaturas erróneas que no forman parte del modelo original.

La cantidad de esquinas falsas es mínima, pero si existen inconsistencia en la forma de algunas de las esquinas reales que forman parte del entorno de física, debido a que ciertos datos no fueron tomados por el algoritmo mientras se realizaba la creación del mapa como muestra la figura 3.26. En general hubo una cantidad mínima de esquinas falsas, lo que indica que la calidad del mapa es aceptable.

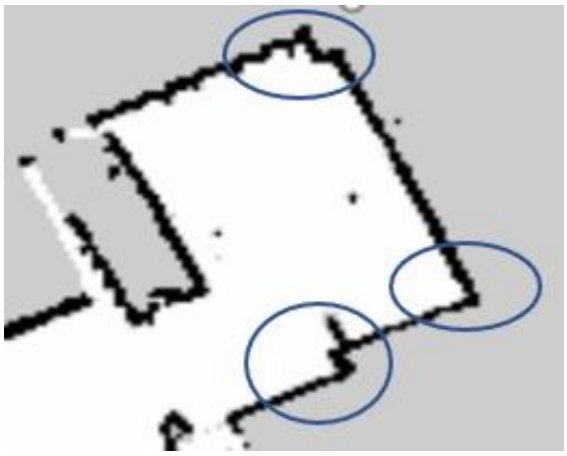


Figura 3.54 Cantidad de esquinas falsas e indefinidas en el mapa experimental generado por la técnica SLAM Cartographer.

Cantidad de áreas cerradas

La última métrica consiste en la cantidad de regiones dentro del mapa que es encuentran delimitadas por una pared que rodea un espacio abierto. Las celdas ocupadas e indefinidas actúan como parte de una limitación de una región específica.

La cantidad de área encerrada es menor, debido a que algunas regiones no estaban bien adecuadas y el sensor no pudo detectarla, debido a la ubicación del sensor. Además, existían espacios reducidos a los que el robot móvil no pudo acceder y los resultados se reflejan en la figura 3.27. En general la cantidad de áreas encerradas es menor y casi toda la región que conforma el entorno de simulación se encuentra delimitado por las paredes; por lo tanto, la calidad del mapa es media.

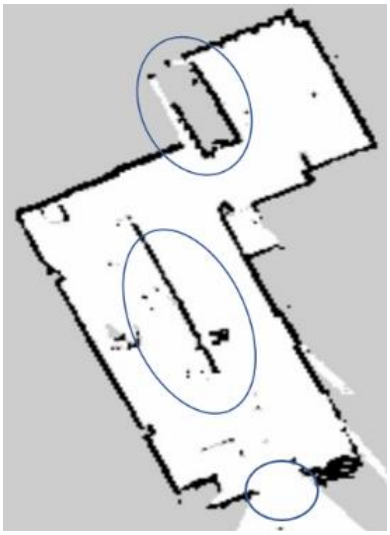


Figura 3.55 Áreas no encerradas en el mapa experimental generado por la técnica SLAM Cartographer.

Localización

A través de un diagrama y gráficos se mostrará el proceso para localizar al robot móvil y establecerlo en su ubicación actual dentro del mapa generado por el algoritmo de Cartographer. Este proceso podrá ser observado a través de la herramienta de visualización Rviz.

Diagrama del proceso de localización experimental

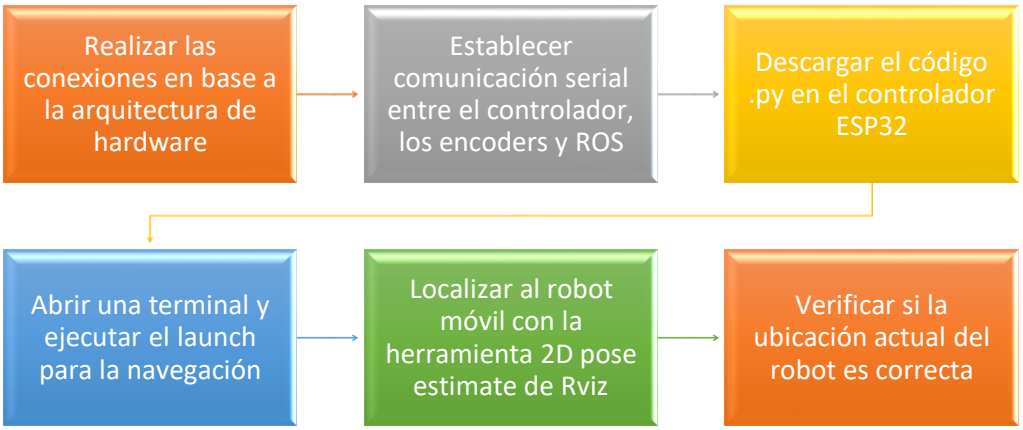
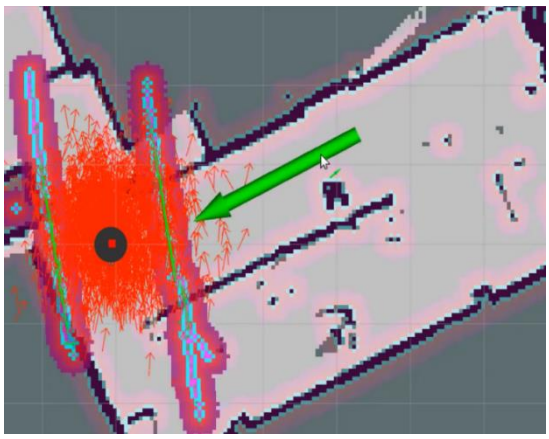


Figura 3.56 Diagrama del proceso de localización experimental

Una vez adquirido el mapa del entorno se lo utiliza para hacer ahora que el robot navegue de forma teleoperada por medio de un joystick sobre este mapa.

Gráficas del proceso de localización

En el proceso de navegación, para comprobar la localización, se situará al robot, de forma intensional en una posición que no es la que en la realidad ocupa en el entorno físico, con el fin de validar posteriormente que se vuelva a localizar correctamente.



Localización errónea del robot de forma intensional

(b) localización real del robot en el entorno físico

Figura 3.57 (a) Localización falsa, (b) localización real

En la figura 3.29 se muestra como el robot en la vida real no está en el lugar que se señala en la herramienta de visualización Rviz, sino que se encuentra una distancia más atrás y como diferente orientación.

Las flechas rojas representan distribuciones de probabilidad de poses cuando estas distribuciones están modeladas como partículas y donde cada flecha apunta hacia donde cree que está mirando el robot. Estas flechas se muestran en la figura 3.29, dispersas y con diferentes direcciones entre ellas debido al no estar correctamente ubicado el robot.

Evaluación de la localización

Coincidencia del borde del mapa

Posteriormente como se indica en el diagrama 3.30 se utiliza la herramienta 2D pose para indicar un estimado de la posición real del robot. Se espera que una vez el robot se encuentre en el proceso de alcanzar un punto meta u objetivo, tanto el mapa cargado como el área del escaneo se vayan acomodando y coincidiendo a medida que el robot se mueve hacia su destino, por lo que las flechas también deben cada vez más irse acomodando, agrupándose y apuntando todas a una sola dirección, que sería la correcta en la que se encuentra el robot.

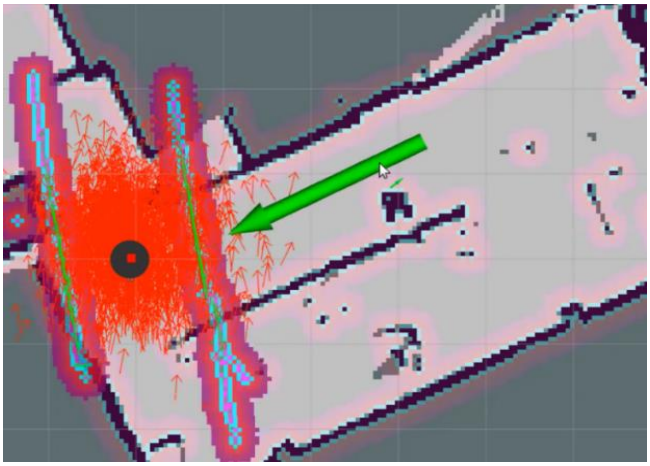


Figura 3.58 Uso de herramienta 2D pose (flecha verde)

Coincidencia del borde del mapa

La métrica se basa en que el área de escaneo delimitado por el sensor laser, representado por las franjas verdes sobre las áreas púrpuras debe coincidir con el borde delimitado de color negro del mapa cargado, esto demuestra que la ubicación del robot móvil dentro del mapa en Rviz es correcto.

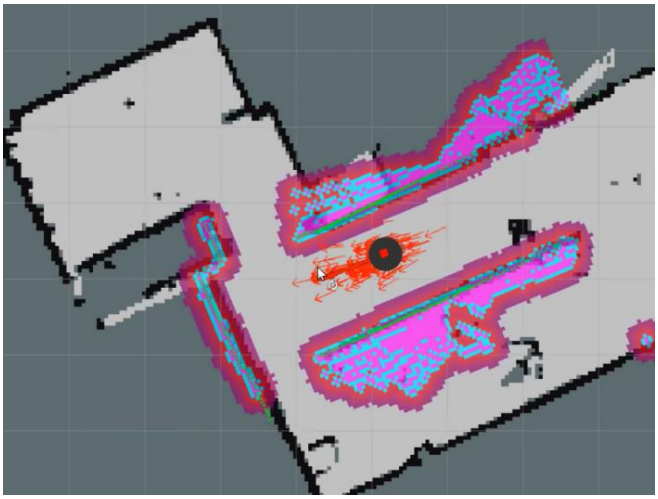


Figura 3.59 Coincidencia del área escaneada y los límites del mapa

Se puede observar en la figura 3.31 que en el proceso experimental no es tan exacta la localización como en la simulación, ya que no se produce la coincidencia exacta, entre el escaneo del sensor y los bordes del mapa cargado al programa.

Durante la evluación de la localización en la implementación se sucitaron problemas, repetitivos, ya que en el primer intento no se consiguió de imediato la cincidencia de los bordes entre el mapa y lo que el sensor veía, incluso si el robot se ponía en camino hacia otro objetivo, este perdía su referencia y se distorcionaba su localización.

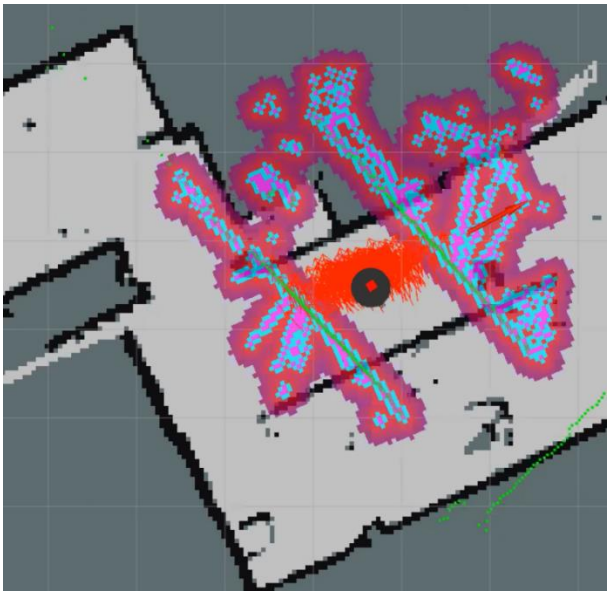


Figura 3.60 pérdida de la referencia y localización

Nos fijamos que hasta que alcance a acomodarse el robot permanecía dando vueltas en el mismo lugar, que es lo que se muestra en la figura 3.32

Para solucionar este inconveniente se realizaron pruebas quitando el aporte de los encoders, ya que al obviarlos del grupo de los sensores del robot, no se producía dicho error. Se evidenió entonces que existe un problema en la calibración de los encoders ya que al hacer pruebas previo a la navegación en las que se movía el robot desde una terminal, através de un program de microPython que enviaba ordenes al controlador y este a su vez al puente H para mover los motores, cuando se le asignaba el robot que se mueva una cierta distancia en valores de pasos del encoder se observaba que al terminar la ejecución de dicha orden siempre había una llanta que recorría más pasos de encoder que la otra llanta.

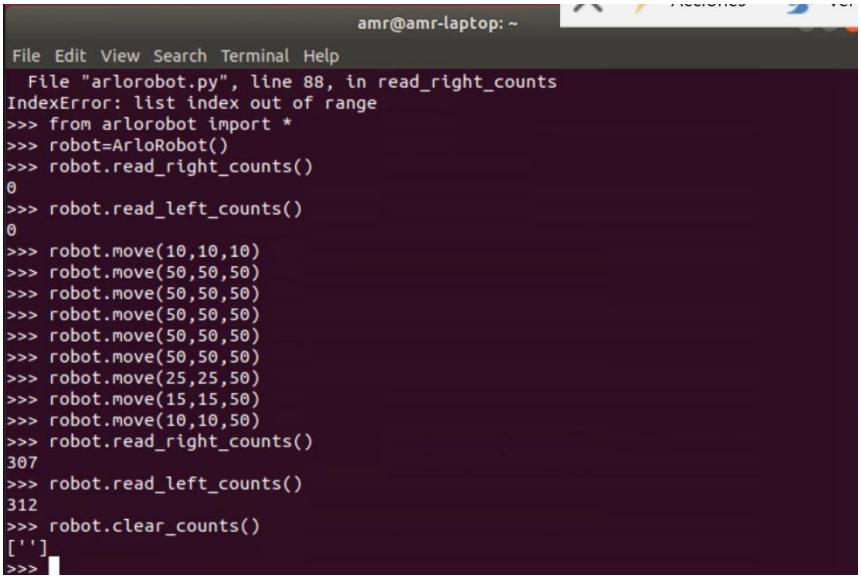


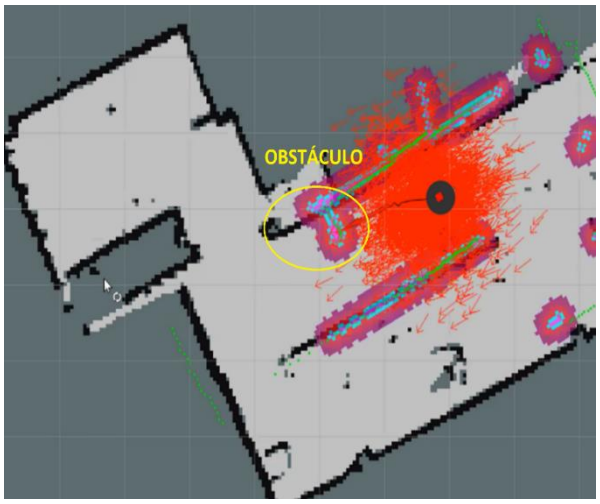
Figura 3.61 pruebas previo a la navegación

Se puede observar en la figura 3.33 como el conteo de pasos del encoder de la rueda derecha es de 307 y la de la rueda izquierda es 312, cuando ambas llantas han recorrido en varios pasos hasta alcanzar el objetivo.

Planificador y seguimiento de trayectoria

Para la planificación de la trayectoria se coloca un obstáculo frente del robot con el fin de evaluar como se generan rutas para lograr evadir dicho obstáculo. En los primero intentos sucedió que el robot colisionó con el obstáculo, sin embargo se evidenciaba que la trayectoria planificada por el algoritmo si evadía el obstáculo. Para solucionar esto se propuso hacer un ajuste de los parámetros involucrados en la navegacipon entre ellos los tiempos de respuesta.

Colocación de obstáculo en el entorno



Visualización del obstáculo en Rviz

Figura 3.62 (a) colocación del obstáculo en el entorno, (b) visualización del obstáculo en Rviz

Evaluación de la planificación de la trayectoria

Para probar la navegación se coloca una nueva meta para que sea alcanzada por el robot con la herramienta **2D Nav Goal** que señala un nuevo objetivo como se señala en la figura 3.35.

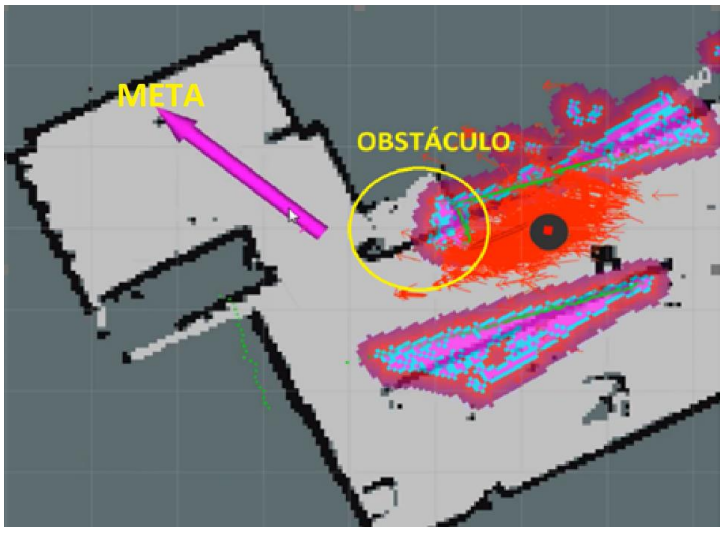


Figura 3.63 Uso de herramienta 2D Nav Goal que señala un nuevo objetivo

Se observará en la figura 3.36 que se plantea una trayectoria que efectivamente evade el obstáculo.

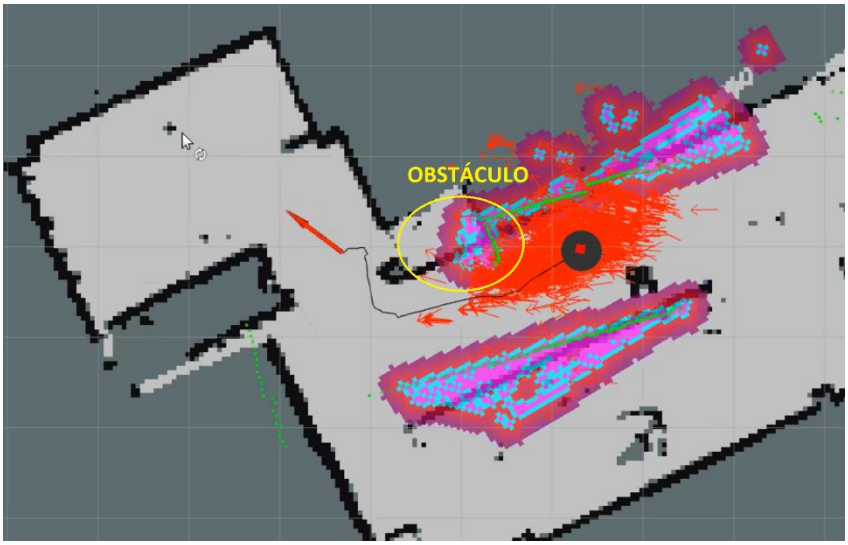


Figura 3.64 planteo de una trayectoria

Como se mencionó anteriormente en varios intentos se efectuó la colisión del robot con el obstáculo, pero el algoritmo seguía calculando trayectorias para evitarlo, aunque por la proximidad de contacto, el esfuerzo del cálculo de posibles trayectorias fue en vano porque se colisionó como se muestra en la figura 3.37

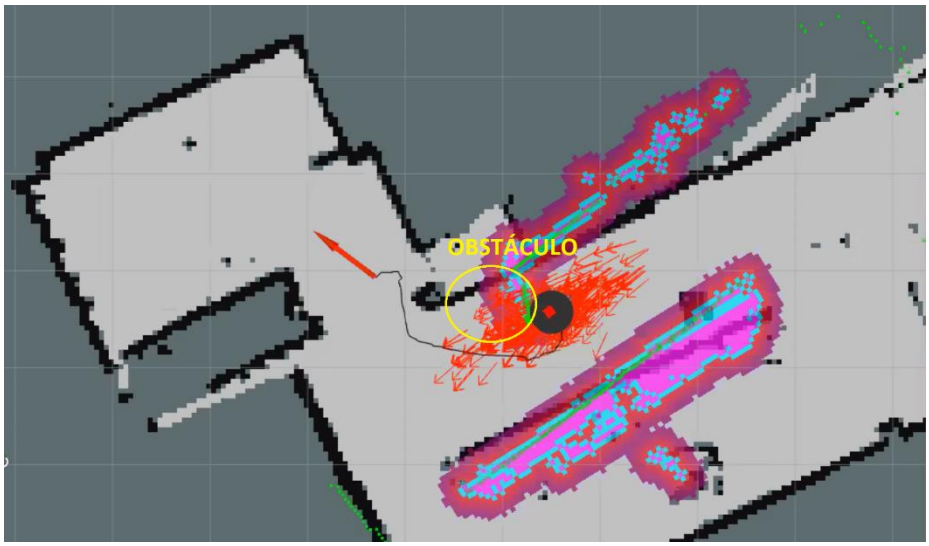


Figura 3.65 colisión entre el robot móvil y el obstáculo

Debido a los problemas presentados se realizaron varios cambios a parámetros que hacen referencia a la frecuencia de publicación de los tópicos, pero después

de varios intentos y de indagar se encontró que tras el proceso de afinación y sintonización de parámetros para maximizar el rendimiento de la pila de navegación de ROS se requiere algunos ajustes y no es tan simple como parece, incluso hay papers enfocados solo a ese tema.

Comparación entre resultados de la simulación y de la implementación

Se presenta a continuación una tabla donde se compara los resultados obtenidos en la parte simulada y la parte implementada del proyecto, de acuerdo con las tareas de la navegación autónoma. Se ha seleccionado distintos factores en cada tarea, que se consideran entre los más importantes para hacer una valoración del rendimiento de la técnica.

Tabla 3.10 Comparación entre los resultados de la simulación e implementación de la navegación autónoma del robot

Tarea	Factor	Simulación	Implementación
Mapeo	Actualización en tiempo real al añadir nuevas características en el mapa	si	si
	Calidad del mapeo	bueno	medio
	Proporción de celdas ocupadas y libres	mínima	media
	Cantidad de esquinas en un mapa	15	10
	Cantidad de áreas cerradas	5	2
Localización	Visualización de la distribución de pose del robot (Flechas rojas)	si	si
	Reajuste de la ubicación del robot a medida que navega	si	no
	Coincidencia entre el borde del mapa cargado y el área de escaneo en tiempo real	buena	medio
Planificación de la trayectoria	Se crea una trayectoria desde la posición de partida hasta la posición objetivo de llegada	si	si
	Ajuste de la trayectoria frente al dinamismo del entorno	si	si
	Se elige y muestra la ruta más corta y directa	si	si
Seguimiento de la trayectoria	Tiempos de respuesta del robot frente a cambios en la trayectoria	bueno	malo

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- Es necesario la generación de un mapa previo para poder representar una estimación de la localización de un robot, referenciado a este mapa, caso contrario se debería hacer uso de herramientas como inteligencia artificial para localizar al robot en un entorno.
- Cartographer SLAM se encuentra entre las mejores técnicas SLAM para la generación de mapas tanto en 2D como en 3D, arrojando mapas con mejores definiciones y calidad que otros métodos que usan SLAM con sensores laser, por ejemplo, Gmaping o Hector SAM, según el estudio de arte realizado.
- A pesar de la buena eficiencia de Cartographer SLAM para el proceso de mapeo, debido a ser una técnica reciente, los servicios ofrecidos para la localización y navegación que ofrece Cartographer ROS no poseen suficiente información para su implementación, por ello se ha integrado paquetes externos exclusivamente para robots diferenciales que con varios años de vigencia son populares y se mantienen actualizados constantemente.
- Cuando se realiza SLAM 2D no es necesario usar la IMU debido a que el robot solo se desplaza en los ejes (x,y) y no realiza movimientos referentes al eje perpendicular al plano de movimiento.
- La navegación autónoma del robot móvil abre puertas a posibles aplicaciones que pueden ser desarrolladas.
- En la parte de simulación del proyecto no se utilizó el mismo modelo de robot que en la vida real, es decir al momento de la implementación, debido a que el modelo URDF que se disponía del Arlobot, obtenido de internet, no estaba completo porque carecía de las ruedas locas que sirven para mantener equilibrada a la plataforma con respecto al suelo, por lo que al lazar la simulación con el Arlobot este se inclinaba arrojando mediciones erróneas con el sensor LIDAR, en vista a esto se decidió utilizar otro modelo

de robot diferencial con característica semejantes, que es el turtlebot3 waffle.

- Debido a las restricciones por la emergencia sanitaria por el COVID 19 se optó por utilizar herramientas que permitan teleoperar de manera remota al robot desde nuestras casas, con lo que se añade la característica de acceso remoto al robot para la navegación autónoma.
- La afinación y sintonización de los parámetros, es un tema que conlleva varias pruebas y estudio, debido a esto en el proyecto se avanzó a comprobar de manera efectiva hasta la tarea de mapeo experimental. Las tareas de localización y planificación de trayectoria experimental requieren mayor enfoque que próximamente se seguirá tratando.

4.2. Recomendaciones

- Realizar un correcto ajuste de los parámetros para los paquetes de navegación y localización es indispensable, ya que, en los ejemplos encontrados en repositorios de internet, no todos los parámetros mostrados se ajustan para diferentes tipos de robots.
- Al momento de trabajar en la conversión del archivo. stl a una extensión .sdf fue necesario correr el comando para la creación del proyecto, pero con permiso de administrador, esto no suele ser así, pero de esta forma funcionó.
- Cuando se modificaron los archivos que serían leído por el simulador gazebo, el que tenía el nombre de model.sfd se debe abrir para editar la ubicación de sus componentes para que apunten al directorio en los que actualmente se encuentran ubicados ya que estos archivos antes habían sido cambiados de lugar y si no se modifica no es posible visualizar en gazebo el entorno simulado.
- Tener en cuenta la correcta ubicación del sensor, específicamente de la altura a la que se encuentra el sensor LIDAR, esta debe ser lo más cercana a la especificada en el modelo URDF del robot invocado en el archivo launch.

BIBLIOGRAFÍA

(s.f.).

A. Filatov, A. F. (2017). Métodos de evaluación de calidad 2D SLAM. *21ª Conferencia de la Asociación de Innovaciones Abiertas (FRUCT) de 2017*, 120-126.

Abdelrasoul Y, S. A. (2016). Un estudio cuantitativo sobre el ajuste de los parámetros de mapeo de ROS y su efecto en la realización de SLAM 2D en interiores. *2016 II Simposio Internacional IEEE sobre Robótica y Automatización de Manufactura (ROMA)* , 1-6.

Afanasyev, M. F. (2018). Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment. Portugal.

Andrade , F., & Llofriu, M. (s.f.). *Universidad de la República.* Obtenido de https://www.fing.edu.uy/inco/grupos/mina/pGrado/pgSLAM/documentos/eda.pdf

Anónimo. (s.f.). *webpersonal.* Obtenido de http://webpersonal.uma.es/~VFMM/PDF/cap2.pdf

Barchesi S, D. M. (2012). *Proyecto Faceval.* Uruguay: UR. FI. Obtenido de http://iie.fing.edu.uy

Burdeos, G. d. (mayo de 2020). onshape to motor. Obtenido de https://github.com/Rhoban/onshape-to-robot

Caverzasi A, S. F. (2014). *Robot móvil autónomo para crear mapas 3D en un ambiente acotado.* Argencon.

Chan S, W. P. (2018). Robust 2D Indoor Localization through Laser SLAM. 6.

Concha , A., & Civera , J. (2015). DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence. Hamburg.

F., F. E. (2017). Localización Probabilística en Drones, para aprendizajes cooperativos.

Fernández A. (2019). *Robust 2D Indoor Localization through Laser SLAM.* Oviedo.

Filipenko, M., & Afanasyev, I. (2018). Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment.

FunPythonEC. (7 de noviembre de 2019). uPyArlo. *MicroPython library to control DBH-10 for Arlo Parallax platform.* Obtenido de https://github.com/FunPythonEC/uPyArlo

FunPythonEC. (marzo de 2020). uPy-rosserial. Obtenido de https://github.com/FunPythonEC/uPy-rosserial

Github. (s.f.). Obtenido de https://github.com/mgtm98/uRos

google. (s.f.). cartographer project. Obtenido de https://github.com/cartographer-project/cartographer_ros

J. Engel, V. K. (1 de Marzo de 2018). Direct Sparse Odometry.

Kuang H, Z. K. (2018). Algoritmo SLAM monocular basado en la estimación mejorada del mapa de profundidad y la selección de fotogramas clave. *décima conferencia internacional de 2018 sobre tecnología de medición y automatización de la mecatrónica (ICMTMA)*, 350-353.

Lazea Gh. (2001). *Aspects on path planning for mobile robots.* Cluj-Napoca.

Lofland, C. (3 de Septiembre de 2019). *Github*. Obtenido de https://github.com/chrisl8/ArloBot

López Torres, P. (2016). Análisis de algoritmos para localización y mapeado simultáneo de objetos. Sevilla.

Lopez, Á. P. (22 de Septiembre de 2017). Estudio y evaluación de los. Madrid, Leganés, España. Obtenido de https://e-archivo.uc3m.es/bitstream/handle/10016/28649/TFG_Angel_Palacio_Lopez_2017.pdf?sequence=1

Mahtani, A., Snachéz , L., Fernández , E., & Martinez , A. (2016). Effective Robotics programming with ROS.

Martínez Q, T. E. (2018). *IMPLEMENTACIÓN DE UN SISTEMA DE NAVEGACIÓN AUTÓNOMO BASADO EN SLAM Y NAVEGACIÓN REACTIVA.* Sangolquí.

Mir. (2019). AMR robots. Obtenido de https://www.mobile-industrial-robots.com/es/insights/report/

Mur-Artal, R., Montiel, J., & Tardós, J. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System.

Newcombe, R. L. (2011). Densse tracking anda mapping in real-time.

Nuchter, A., Bleier, M., Schauter, J., & Janotta, P. (01 de 03 de 2017). MPROVING GOOGLE'S CARTOGRAPHER 3D MAPPING BY CONTINUOUS-TIME SLAM. Obtenido de https://www.researchgate.net/publication/313115101_IMPROVING_GOOGLE%27S_CARTOGRAPHER_3D_MAPPING_BY_CONTINUOUS-TIME_SLAM

openwebinars. (s.f.). Obtenido de https://openwebinars.net/blog/que-es-ros/

Pineda-Torres, F. (2019). Técnicas de slam con filtros probabilísticos; caracterización y resultados en robots móviles.

Pire, T., Fischer, T., Civera , J., De Cristóforis , P., & Jacobo Berlles, J. (2015). Stereo parallel tracking and mapping for robot localization. Hamburg.

Reinoso García, O. (18 de abril de 2016). *Canal UNED.* Obtenido de https://canal.uned.es/video/5a6f5bf3b1111f967a8b45ac

ROS.org. (2 de Agosto de 2019). *rplidar*. Obtenido de rplidar: http://wiki.ros.org/rplidar

Satio, I. (12 de 03 de 2018). *ROS.org.* Obtenido de http://wiki.ros.org/openni2_launch

Savkin, A. V. (2015). Safe robot navigation among moving and steady obstacles. Obtenido de from https://ebookcentral.proquest.com

SLAMTEC. (s.f.). Obtenido de http://www.slamtec.com/en/Lidar/A1

Stephan, J. (5 de 10 de 2014). *ROS.org.* Obtenido de https://wiki.ros.org/differential_drive

Syahrul F, D. S. (2019). Research Study of Occupancy Grid map Mapping. *International Electronics Symposium (IES)*, 238-241.

TheoKaning. (2 de agosto de 2019). *ROS.org*. Obtenido de http://wiki.ros.org/rplidar

Thrun, S. (2003). Robotic mapping: A survey. In Exploring Artificial Intelligence in the New Millenium. Kaufmann Publishers.

Valencia J, M. A. (2009). MODELO CINEMÁTICO DE UN ROBOT MÓVIL TIPO DIFERENCIAL Y NAVEGACIÓN A PARTIR DE LA ESTIMACIÓN ODOMÉTRICA. *Scientia et Technica Año XV, No 41, Mayo de 2009. Universidad Tecnológica de Pereira. ISSN 0122-1701* , 191-196.

Wang J, C. X. (2018). Método de posicionamiento automático basado en codificador rotatorio para robot móvil. *2018 5a Conferencia Internacional de Ciencias de la Información e Ingeniería de Control (ICISCE)* , 500-504.

Wei W, S. B.-J. (2019). Corrección de orientación para Hector SLAM en la etapa inicial. *7ma Conferencia Internacional sobre Tecnología y Aplicaciones de Inteligencia de Robots (RiTA) , 2019*, 125-129.

Wonnacott, D. (8 de Septiembre de 2013). *Github*. Obtenido de https://github.com/dawonn/vectornav

Zhan, W. S. (2009). Simultaneous Localization and Mapping Exactly Sparce Information Filter. Australia: Wold Scientific.

5. ANEXOS

ANEXO 1.- FORMATOS DE MENSAJES UTILIZADOS

sensor_msgs / Mensaje de LaserScan

Archivo: sensor_msgs / LaserScan.msg

Definición de mensaje sin procesar

```
# Escaneo único desde un telémetro láser plano
#
# Si tiene otro dispositivo de rango con un comportamiento diferente (p. Ej
., Una matriz de sonda #), busque o cree un mensaje diferente, ya que las aplicaciones
# harán suposiciones bastante específicas de láser sobre estos datos

Encabezado encabezado              # marca de tiempo en el encabezado es el tiempo de adquisición de
                          # el primer rayo en el escaneo.
                          #
                          # en el cuadro frame_id, los ángulos se miden alrededor del
                          # eje Z positivo (en sentido antihorario, si Z está arriba)
                          # con el ángulo cero hacia adelante a lo largo del eje x

float32 angle_min          # ángulo de inicio de la exploración [rad]
float32 angle_max          # ángulo final de el escaneo [rad]
float32 angle_increment    # distancia angular entre mediciones [rad]

float32 time_increment     # tiempo entre mediciones [segundos] - si su escáner
                          # se está moviendo, esto se usará en la posición de interpolación
                          # de puntos 3d
float32 scan_time          # tiempo entre escaneos [segundos]

float32 range_min          # valor de rango mínimo [m]
float32 range_max          # valor de rango máximo [m]

float32 [] rangos           # datos de rango [m] (Nota: los valores <range_min o> range_max deben descartarse)
float32 [] intensidades      # datos de intensidad [unidades específicas del dispositivo ] Si su
                          # dispositivo no proporciona intensidades, deje
                          # la matriz vacía.
```

Definición de mensaje compacto

std_msgs / Encabezado de encabezado
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32 [] rangos
float32 [] intensidades

sensor_msgs/Imu Message

File: sensor_msgs/Imu.msg

Raw Message Definition

```
# This is a message to hold data from an IMU (Inertial Measurement Unit)
#
# Accelerations should be in m/s^2 (not in g's), and rotational velocity should be in rad/sec
#
# If the covariance of the measurement is known, it should be filled in (if all you know is the
# variance of each measurement, e.g. from the datasheet, just put those along the diagonal)
# A covariance matrix of all zeros will be interpreted as "covariance unknown", and to use the
# data a covariance will have to be assumed or gotten from some other source
#
# If you have no estimate for one of the data elements (e.g. your IMU doesn't produce an orientation
# estimate), please set element 0 of the associated covariance matrix to -1
# If you are interpreting this message, please check for a value of -1 in the first element of each
# covariance matrix, and disregard the associated estimate.

Header header

geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z
```

Definición de mensaje compacto

```
std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
```

sensor_msgs/PointCloud2 Message

File: sensor_msgs/PointCloud2.msg

Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool    is_bigendian # Is this data bigendian?
uint32  point_step   # Length of a point in bytes
uint32  row_step     # Length of a row in bytes
uint8[] data         # Actual point data, size is (row_step*height)

bool is_dense        # True if there are no invalid points
```

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

nav_msgs/Odometry Message

File: nav_msgs/Odometry.msg

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Compact Message Definition

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

ANEXO 2.- GÁFICOS DE NODOS GENERADOS POR ROS

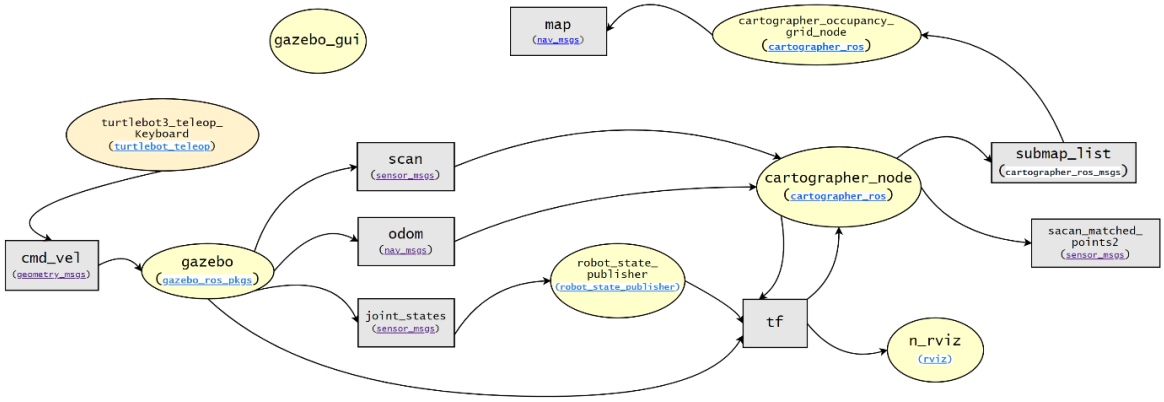


Figura 5.66 Gráfico de nodos del proceso de mapeo

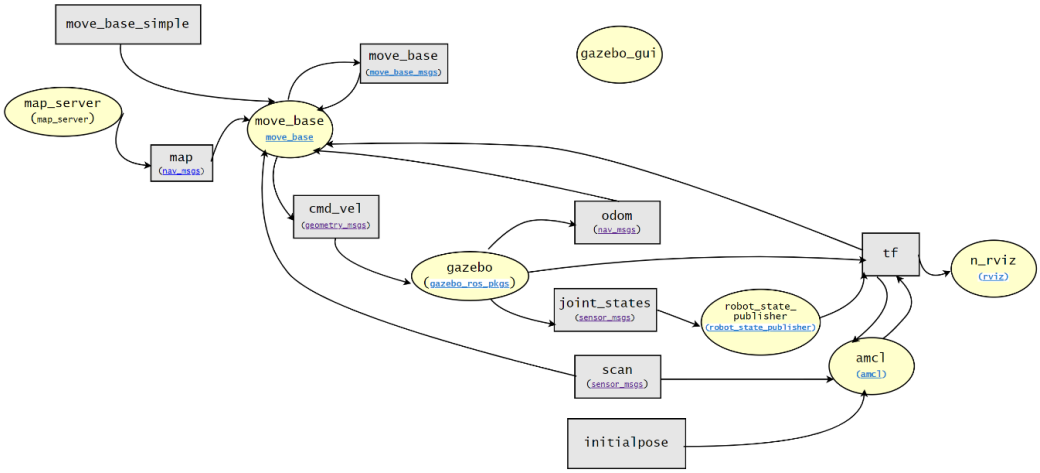


Figura 5.67 Gráfico de nodos del proceso de seguimiento de trayectoria

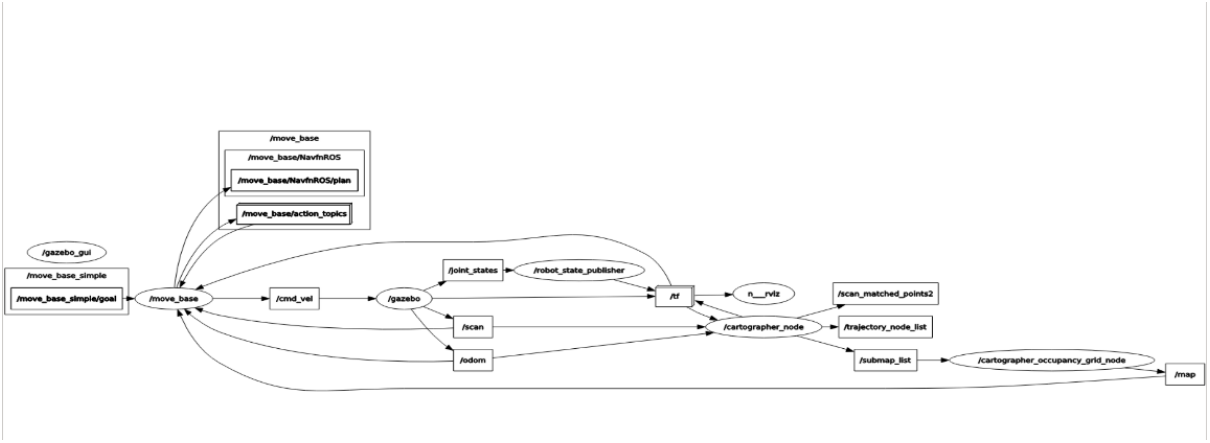


Figura 5.68 Prueba de navegación y planificación con paquete DWA planner

ANEXO 3.- CÓDIGOS DE ARCHIVOS LAUNCH

cartographer_node_sim_maps.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type
[burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="cartographer" doc="slam type [gmapping,
cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename"
default="turtlebot3_lds_pure2d_sim_maps.lua"/>
  <arg name="open_rviz" default="true"/>

  <include file="$(find
turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)"/>
  </include>

<include file="$(find
my_robot_description)/launch/my_turtlebot_launcher.launch">
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration,
RTAB-Map -->
  <include file="$(find cartographer_launcher)/launch/turtlebot3_$(arg
slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg
configuration_basename)"/>
  </include>
  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true" args="-d
$(find cartographer_launcher)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

cartographer_node_sim_nav_amcl.launch

```
<?xml version="1.0"?>
<launch>

  <arg name="load_state_filename" default="$(find
cartographer_launcher)/maps/mapa_sim.pbstream" />
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type
[burger, waffle, waffle_pi]"/>
<arg name="open_rviz" default="true"/>
 <arg name="slam_methods" default="cartographer" doc="slam type [gmapping,
cartographer, hector, karto, frontier_exploration]"/>

  <include file="$(find
turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)"/>
  </include>

  <arg name="map_file" default="$(find
cartographer_launcher)/maps/mapa_sim.yaml"/>

  <!-- Map server -->
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg
map_file)"/>

  <!-- AMCL -->
  <include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>

  <!-- move_base -->
  <include file="$(find turtlebot3_navigation)/launch/move_base.launch">
    <arg name="model" value="$(arg model)" />
  </include>
<include file="$(find
my_robot_description)/launch/my_turtlebot_launcher.launch">
    <arg name="model" value="$(arg model)"/>
  </include>

    <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true" args="-d
$(find cartographer_launcher)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>

</launch>
```

cartographer_node_exp_maps.launch

```
<?xml version="1.0"?>
<launch>
  <arg name="open_rviz" default="true"/>
  <arg name="configuration_basename"
default="turtlebot3_lds_pure2d_exp_maps.lua"/>
  <arg name="urdf_file" default="xacro '$(find
arlobot_description)/urdf/common_default.urdf.xacro'"/>
  <param name="robot_description" command="$(arg urdf_file)"/>
  <rosparam param="ticks_meter">300</rosparam>
  <rosparam param="base_width">0.393</rosparam>
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"/>

<node name="rosserial_node" pkg="rosserial_arduino" type="serial_node.py"
output="screen">
    <param name="port" type="string" value="/dev/ttyUSB0"/>
<param name="baud" type="int" value="115200"/>
  </node>

  <!-- Rplidar node -->
  <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"
output="screen">
    <param name="serial_port" type="string" value="/dev/ttyUSB2"/>
    <param name="serial_baudrate" type="int" value="115200"/>
    <!--A1/A2 -->
    <param name="frame_id" type="string" value="neato_laser"/>
    <param name="inverted" type="bool" value="false"/>
    <param name="angle_compensate" type="bool" value="true"/>
  </node>

  <!-- diff drive nodes -->
  <node pkg="differential_drive" type="diff_tf.py" name="diff_tf_node">
    <rosparam param="base_frame_id">"base_footprint"</rosparam>
    <rosparam param="rate">30</rosparam>
    <remap from="rwheel" to="rwheel"/>
    <remap from="lwheel" to="lwheel"/>
  </node>

  <node pkg="differential_drive" type="virtual_joystick.py"
name="virtual_joystick" output="screen"/>

  <node pkg="differential_drive" type="twist_to_motors.py"
name="twist_to_motors" output="screen"/>

  <!-- cartographer_node -->
  <node pkg="cartographer_ros" type="cartographer_node"
name="cartographer_node" args="-configuration_directory $(find
cartographer_launcher)/config               -configuration_basename $(arg
configuration_basename)" output="screen">
  </node>

  <!-- cartographer_occupancy_grid_node -->
  <node pkg="cartographer_ros" type="cartographer_occupancy_grid_node"
name="cartographer_occupancy_grid_node" args="-resolution 0.05"/>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true" args="-d
$(find cartographer_launcher)/rviz/turtlebot3_cartographer.rviz"/>
```

```
  </group>
</launch>
```

cartographer_node_exp_nav_teb_amcl.launch

```
<?xml version="1.0"?>
<launch>
  <arg name="open_rviz" default="true"/>
<arg name="load_state_filename" default="$(find
cartographer_launcher)/maps/exp_lidar_map.pbstream" />
  <arg name="configuration_basename"
default="turtlebot3_lds_pure2d_exp_nav.lua"/>
  <arg name="urdf_file" default="xacro '$(find
arlobot_description)/urdf/common_default.urdf.xacro'"/>
  <param name="robot_description" command="$(arg urdf_file)"/>
  <rosparam param="ticks_meter">300</rosparam>
  <rosparam param="base_width">0.393</rosparam>
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"/>
<node name="rosserial_node" pkg="rosserial_arduino" type="serial_node.py"
output="screen">
    <param name="port" type="string" value="/dev/ttyUSB2"/>
<param name="baud" type="int" value="115200"/>
  </node>
  <arg name="map_file" default="$(find
cartographer_launcher)/maps/exp_lidar_map.yaml"/>

  <!-- Map server -->
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg
map_file)"/>

<!-- Arguments -->
  <arg name="scan_topic"     default="scan"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>

  <!-- AMCL -->
  <node pkg="amcl" type="amcl" name="amcl">

    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="3000"/>
    <param name="kld_err"                   value="0.02"/>
    <param name="update_min_d"              value="0.20"/>
    <param name="update_min_a"              value="0.20"/>
    <param name="resample_interval"         value="1"/>
    <param name="transform_tolerance"       value="0.5"/>
    <param name="recovery_alpha_slow"       value="0.00"/>
    <param name="recovery_alpha_fast"       value="0.00"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <param name="gui_publish_rate"          value="50.0"/>

    <remap from="scan"                      to="$(arg scan_topic)"/>
    <param name="laser_max_range"           value="-1"/>
    <param name="laser_max_beams"           value="360"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
```

```
    <param name="laser_z_rand"               value="0.5"/>
    <param name="laser_sigma_hit"            value="0.2"/>
    <param name="laser_lambda_short"         value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type"           value="likelihood_field"/>

    <param name="odom_model_type"            value="diff"/>
    <param name="odom_alpha1"                value="0.2"/>
    <param name="odom_alpha2"                value="0.2"/>
    <param name="odom_alpha3"                value="0.2"/>
    <param name="odom_alpha4"                value="0.2"/>
    <param name="odom_frame_id"              value="odom"/>
    <param name="base_frame_id"              value="base_footprint"/>
    <param name="tf_broadcast"              value="true"/>

  </node>

  <!-- Rplidar node -->
  <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode"
output="screen">
    <param name="serial_port" type="string" value="/dev/ttyUSB1"/>
    <param name="serial_baudrate" type="int" value="115200"/>
    <!--A1/A2 -->
    <param name="frame_id" type="string" value="neato_laser"/>
    <param name="inverted" type="bool" value="false"/>
    <param name="angle_compensate" type="bool" value="true"/>
  </node>
  <!-- diff drive nodes -->
 <node pkg="differential_drive" type="diff_tf.py" name="diff_tf_node">
    <rosparam param="base_frame_id">"base_footprint"</rosparam>
    <rosparam param="rate">30</rosparam>
    <remap from="rwheel" to="rwheel"/>
    <remap from="lwheel" to="lwheel"/>
  </node>


  <include file="$(find
cartographer_launcher)/launch/arlobot_move_base_teb.launch">
  </include>

  <node pkg="differential_drive" type="twist_to_motors.py"
name="twist_to_motors" output="screen">
<remap from="twist" to="cmd_vel"/>
</node>

  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true" args="-d
$(find cartographer_launcher)/rviz/turtlebot3_cartographer.rviz"/>
  </group>
</launch>
```

my_turtlebot_launcher.launch

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type
[burger, waffle, waffle_pi]"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.2"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
my_robot_description)/worlds/entorno.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder
$(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf
-model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg
z_pos) -param robot_description" />
</launch>
```

ANEXO 4.- MAPAS GENERADOS

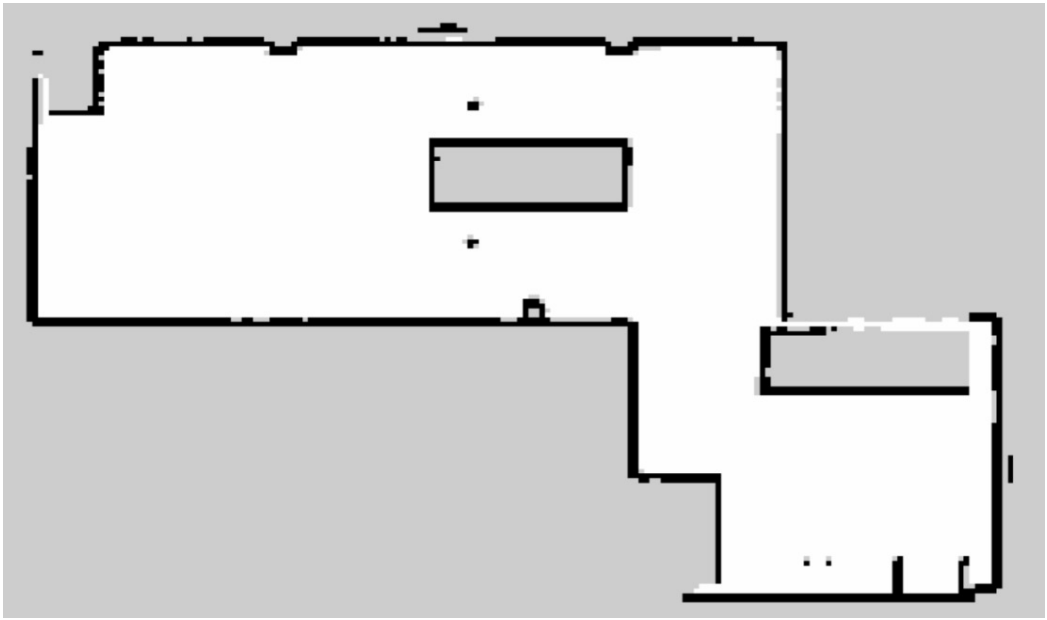


Figura 5.69 Mapa generado de la simulación

Archivo yalm de parámetros del mapa generado en la simulación

```
image: mapa_sim.pgm
resolution: 0.050000
origin: [-5.253112, -3.985279, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

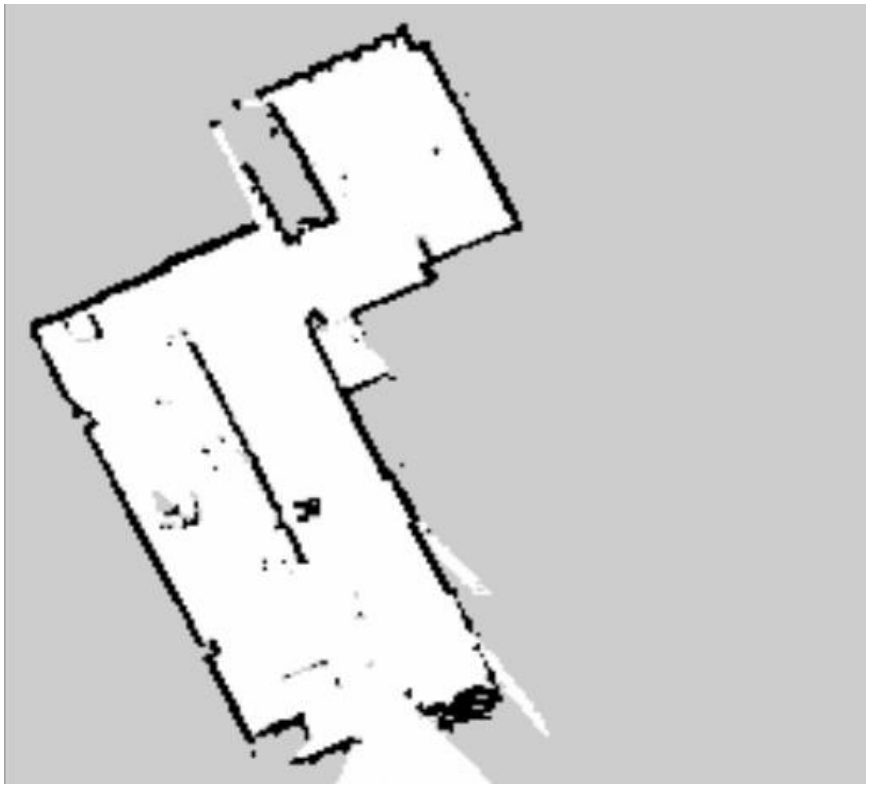


Figura 5.70 Mapa generado de la implementación

Archivo yalm de parámetros del mapa generado en la implementación

```
image: exp_lidar_map.pgm
resolution: 0.050000
origin: [-3.367338, -12.056153, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```