Proyecto Fin de Máster
Máster en Ing. Electrónica, Robótica y Automática

Exploración de entornos no estructurados mediante técnicas de SLAM aplicadas a un robot 8x8

Autor: Carlos E. Pérez Hernández
Tutor: Ángel Rodríguez Castaño

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster
Máster en Ing. Electrónica, Robótica y Automática

Exploración de entornos no estructurados mediante técnicas de SLAM aplicadas a un robot 8x8

Autor:
Carlos E. Pérez Hernández

Tutor:
Ángel Rodríguez Castaño
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Exploración de entornos no estructurados mediante técnicas de SLAM aplicadas a un robot 8x8

Autor: Carlos Eduardo Pérez Hernádez

Tutor: Ángel Rodríguez Castaño

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos

Agradecimientos

A mis padres que siempre me han apoyado, sin ellos no hubiera podido cumplir este nuevo logro.

Al profesor Ángel Castaño por darme la oportunidad de realizar este proyecto y de invertir su tiempo en transmitirme sus conocimientos.

A todas las nuevas amistades que he hecho en este bello país.

Carlos Eduardo Pérez Hernández
Sevilla, 2020

Resumen

En el presente trabajo se exponen diversos métodos de mapeo y localización para robots móviles, así como también los métodos de navegación más conocidos y usados. Estos conocimientos servirán para poder implementar un sistema que le permita a un vehículo 8X8 poder mapear y localizarse en su entorno mientras navega de manera autónoma. Se realizarán pruebas reales y simuladas con todo el sistema desarrollado y se analizarán los resultados obtenidos.

Abstract

In the present work, some mapping and location methods for mobile robots are exposed, as well some of the best known and most used navigation methods. This knowledge will serve to implement a system that allows an 8X8 vehicle to be able to map and locate itself in its environment while navigating autonomously. Real and simulated tests will be made with the entire system developed and the results obtained will be analyzed.

Índice General

Índice de Figuras

1 Introducción

1.1 Motivación.

La robótica móvil en general ha adquirido mucha importancia en los últimos años. Este creciente interés está motivado por las numerosas tareas útiles que un robot podría realizar para los humanos en nuestra vida diaria. Para lograr lo anterior mencionado, es necesario conseguir y mantener un ambiente de trabajo seguro entre el hombre y la máquina.

A este vehículo 8x8 se le considerará como un robot debido a que estará dotado de sensores y actuadores. A pesar de la gran cantidad de aportaciones y de avances tecnológicos, programar un robot sigue siendo una tarea complicada. Esto es debido a que existen muchas aplicaciones y ambientes en los cuales pueden desempeñarse, lo que genera que cada sistema de fabricación e implementación sea diferente.

Uno de los mayores retos en el ambiente de la robótica móvil, es que estos vehículos deben poder posicionarse de manera confiable para que esto funcione eficientemente. La forma más común de lograr esto es utilizando infraestructura adicional, sin embargo, la instalación de dicha infraestructura requiere de mucho tiempo, trabajo manual, coste económico y computacional. Por lo tanto, se puede afirmar que la investigación dentro del área de métodos alternativos de localización basados en mapas es crucial para el desarrollo de este proyecto.

La motivación general de este proyecto surgió como una iniciativa de querer proporcionarle al vehículo móvil 8X8 las capacidades necesarias que le permitan realizar un mapa de su entorno, detectar los objetos que se encuentran en él y que el vehículo pueda navegar de manera autónoma dentro de ese entorno.

1. 2 Propósito y enfoque del proyecto.

Desde el inicio de este trabajo se planteó que el propósito principal es desarrollar un sistema eficiente, preciso y de bajo costo que le permita al vehículo 8x8 mapear, detectar objetos y navegar de manera autónoma.

El trabajo se realizará en ROS (Robot Operating System) [3], que es un entorno software de código libre que ha sido desarrollado para facilitar la creación e implementación de aplicaciones robóticas. Permite incorporarle al robot desde controladores simples hasta avanzados algoritmos, además de brindar el uso de potentes herramientas de desarrollo. Este *framework* ha mostrado ser bastante poderoso para poder desarrollar robots móviles tanto en el ámbito de la investigación como en el industrial, especialmente para poder simularlos y depurarlos [4].

En este trabajo primero se creará un modelo digitalizado del vehículo en ROS tratando de respetar sus características generales. A continuación se le añadirá un modelo del radar RpLidar-A3 que servirá como escáner de rango de entrada y con ello, mediante el desarrollo y presentación de un sistema de pruebas en Gazebo y Rviz, se evaluará el uso de la técnica SLAM para generar mapas 2D del entorno. El software implementado de este proyecto será una extensión de un sistema ya existente, donde ROS servirá como base para la implementación.

Posteriormente se evaluará el rendimiento de esta configuración a fin de proporcionar un punto de partida antes de realizar pruebas físicas con el vehículo 8x8. Las limitaciones encontradas durante el desarrollo de este proyecto permitirán contribuir al mejoramiento de aplicaciones de investigación sobre vehículos móviles. Cabe resaltar que en todas las pruebas que se realicen se incluirá inspección visual y medición manual de distancias, esto con el fin de obtener y mostrar resultados más factibles para los proyectos de investigación futuros.

El enfoque de este proyecto es el mapeo, detección de objetos y la navegación autónoma, sin embargo, indirectamente también incluye crear un modelo digital del vehículo y un entorno de pruebas que permitan a las investigaciones futuras evaluar otros vehículos robóticos con los que cuenta el laboratorio.

1.3 Estructura del proyecto.

Este proyecto está organizado en 6 capítulos, en el capítulo 1 se realiza una breve introducción de los temas generales que se abordan en este proyecto, así como también la motivación y el enfoque para llevarlo a cabo. En el capítulo 2 se explican las tecnologías que permitirán alcanzar el propósito de este proyecto, se detalla la estructura y funcionamiento del software de código libre ROS, y se da una breve introducción a la definición de SLAM. El capítulo 3 describe todo el hardware y software con el que cuenta el prototipo 8X8, los protocolos de comunicación, sistemas de emergencia, etc.

En el capítulo 4 se explicará de manera detallada el sistema de mapeo y localización simultáneos (SLAM), se citará varios algoritmos SLAM que existen, así como sus implementaciones en diversas investigaciones. También se mencionará el algoritmo a ocupar en este proyecto y por qué se ha seleccionado. El capítulo 5 establece la definición de varios tipos de algoritmos de navegación para vehículos autónomos, algunos de ellos se implementarán y se analizarán sus resultados.

Por último, en el capítulo 6 se mencionarán las conclusiones finales de este proyecto, los alcances obtenidos y se propondrán desarrollos futuros que podrían implementarse en el vehículo 8X8.

2 Antecedentes

En este capítulo se presentará una descripción sobre los temas principales del proyecto y las variadas tecnologías (hardware y software) que permitirían poder desarrollarlo.

2.1 Robot Operating System

ROS (*Robot Operating System*), es un conjunto de herramientas que proporcionan la funcionalidad y los servicios de un sistema operativo que permiten desarrollar y modelar sistemas robóticos. Estos servicios incluyen abstracción del hardware, intercambio de mensajes entre procesos, gestión de paquetes, entre otros [7]. El mayor beneficio que tiene ROS, es la gran cantidad de funcionalidades y herramientas con las que cuenta [8].

Debido a que diferentes tipos de robots pueden tener hardware muy variable, poder reutilizar el código entre ellos se vuelve una tarea algo complicada. Sumado a esto, crear software para robots es complicado, particularmente a medida que la escala, rapidez y alcance de la robótica aumentan. Sin embargo, ROS está diseñado para que el usuario pueda superar estos problemas [9].

Un sistema que funciona con ROS debe tener un nodo master (roscore), el cual actúa como nombre de servidor y permite que otros nodos se encuentren entre sí para formar conexiones directas (figura 1). El nodo master en ROS (roscore) es la parte esencial de cada programa y se puede describir a *roscore* como un conjunto de nodos centrales de ROS [10]. Dentro de los conceptos fundamentales en la implementación de ROS están los nodos, los mensajes y los tópicos y servicios.

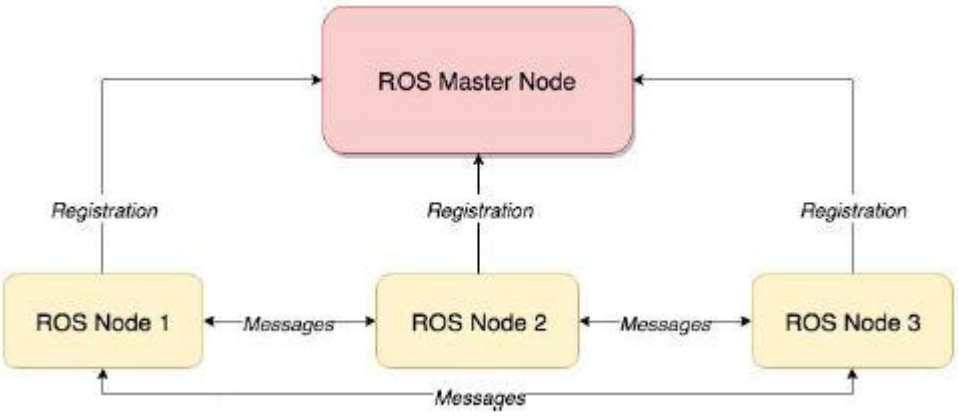


Figura 1. Distribución de nodos y mensajes en ROS [11].

Los nodos son una especie de librería en donde cada uno de ellos se encarga de controlar una pequeña parte del sistema. Estos nodos están conectados en una red por medio de tópicos y servicios, los cuales publican información para que cualquier interesado pueda suscribirse y recibir todos los mensajes generados (figura 2).

Figura 2. Configuración de ROS con nodos publicadores y suscritos compartiendo datos mediante tópicos. [11]

En resumen, el nodo master (roscore) debe estar siempre ejecutándose para que los nodos ROS se comuniquen, los nodos pueden intercambiar mensajes publicando y suscribiéndose a ciertos temas o invocando directamente los servicios y acciones de los otros nodos. [10].

Sin embargo, existen dos inconvenientes que son necesarios tomar en consideración cuando se pretende desarrollar un proyecto en ROS [12]:

1.- Para que se ejecute el programa en ROS, roscore debe estar ejecutándose en todo momento. Por lo tanto, si roscore termina o falla, todo el programa basado en ROS fallará, aunque el resto del sistema esté funcionando sin problema alguno y esté escrito correctamente.

2.- El acceso a la red en ROS no está totalmente segura, ya que la comunicación entre nodos no está asegurada y esto genera que el sistema sea vulnerable, ya que alguien puede acceder y alterar el comportamiento del robot.

A pesar de estos inconvenientes se pretende realizar este proyecto en ROS ya que continúa siendo una muy buena solución para poder desarrollarlo e implementarlo. Permitirá un fácil acceso a la información proporcionada por los sensores, procesarla y finalmente enviar las instrucciones a los actuadores del robot móvil.

Para poder navegar dentro de un entorno, un robot móvil necesita algoritmos para planificar una ruta desde su posición actual hasta el objetivo y calcular los comandos de control que dirigen al robot hacia ese objetivo. Estos algoritmos deben tener en cuenta los obstáculos circundantes (estáticos y dinámicos) y generar una trayectoria que conecte la configuración inicial del robot con la futura configuración final (meta) sin chocar con ninguno de estos obstáculos.

Los sistemas existentes tradicionales de navegación robótica intentan calcular las rutas hacia el objetivo final haciendo uso principalmente de los criterios de rendimiento, tales como la longitud de la ruta, la eficiencia en el tiempo y el costo computacional [5]. Lo que obligará a que los movimientos del robot móvil puedan ser no socialmente adecuados, sin embargo, este proyecto no está enfocado en la interacción humano-robot ya que no se pretende que el vehículo 8x8 interactúe en un ambiente rodeado de personas.

En la robótica, es de suma importancia que el robot sepa en todo momento la posición en la que se encuentra en relación al mundo en el que opera (entorno). Así como también, el robot debe conocer dónde están ubicados los diferentes sensores con los que cuenta. Cada sensor ocupa su propio marco de coordenadas y es trabajo del robot transformar todas estas coordenadas, compararlas y establecer un marco de coordenadas único.

Para ello existe la librería TF en ROS, la cual provee una manera de rastrear las diferentes coordenadas que los sensores van emitiendo [13]. Esto facilitará poder conocer y ocupar las coordenadas específicas necesarias sin tener que recabar la información de todos los marcos de

coordenadas en el sistema. Esta transformación es simplemente un mensaje que contiene una traslación y rotación, proporcionando una relación entre dos marcos de coordenadas.

Odom y *Base_Link* son usualmente los marcos que se utilizan (figura 3). El marco *Base_Link* está unido a la base del robot, en este caso el chasis del vehículo, y el marco *Odom* representará el origen de la posición inicial del robot. La transformación de *Odom* a *Base_Link* especificará la traslación y rotación que lleva el vehículo desde su posición inicial hasta la posición en que se encuentra en el momento de la lectura de coordenadas.

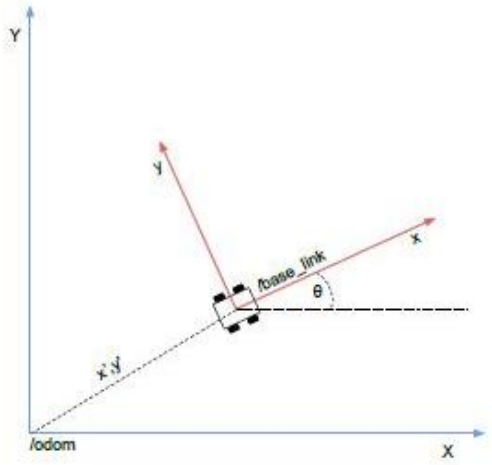


Figura 3. Relación entre *Base_Link* y *Odom_Frame*. [11]

Estas referencias de posicionamiento del vehículo utilizando la transformación *Base_Link – Odom* pueden tener sus problemas de malas lecturas, por lo tanto, es recomendable agregar la transformación *Odom – Map*, la cual es comúnmente utilizada en la robótica. Dicha transformación permite especificar dónde está colocado el marco *Odom* dentro del marco del mapa global.

Para la visualización del modelo del vehículo se utilizarán dos poderosas herramientas que proporciona ROS: Gazebo y RViz. Gazebo ofrece la capacidad de simular robots de forma precisa y eficiente en entornos complejos tanto en interiores como en exteriores, esto permitirá ver el comportamiento del robot en un ambiente real. RViz (ROS Visualization), permite visualizar en 3D toda la información del software del robot (sensores, algoritmos, etc), y con ello poder ver cómo el robot percibe el mundo donde está siendo simulado.

Ambas herramientas serán de mucha ayuda durante el mapeo y localización resultante del algoritmo SLAM.

2. 2 Sensores

Para el desarrollo de este proyecto se hará uso de un láser que permitirá medir la distancia que existe entre el robot y cualquier objeto que se encuentre a su alrededor. Los láseres miden esta distancia al calcular el tiempo que tarda viajando un láser de luz hacia a un objeto y de vuelta al sensor. Se utilizará un RPLIDAR-A3 como sensor láser para el vehículo móvil.

LIDAR (*light detection and ranging*) es un tipo de láser comúnmente utilizado en la robótica para hacer representaciones muy precisas en 2D O 3D de los diversos objetos que se encuentran en el entorno. Este tipo de láser se usa muchísimo para vehículos móviles debido a su gran eficiencia para mapear el entorno y detectar obstáculos [6].

Se ha optado por el RPLIDAR-A3 debido a las grandes prestaciones que tiene para ofrecer:

- Tamaño de 4 cm de espesor.
- Rango de distancia de 10-25 m.
- Frecuencia de muestreo de 10000-16000 veces por segundo.
- Velocidad de escaneo de 10-20 Hz.
- Exploración omnidireccional láser de 360 grados.
- Resolución angular de 0.3375°-0.54°.
- Efectivo tanto para interiores, como para exteriores.

Se utilizará la paquetería RPLIDAR-A3 [14] que se encuentra disponible en ROS para ejecutar este láser.

2. 3 Procesador.

Es necesario definir el alcance del proyecto para poder seleccionar el tipo de procesador adecuado que llevará el vehículo (PC a bordo). Las características del procesador dependerán de la cantidad de datos que se tengan que manejar del sensor RPLIDAR-A3 y la eficiencia que queramos que se tenga al mapear, al detectar objetos y al navegar de manera autónoma.

El procesador hará uso de los datos que emita el sensor, recabará la información del mapa que se vaya formando, de los objetos que vaya detectando y enviará información para controlar los actuadores del vehículo. Este control determinará el ángulo de posición de la dirección, la aceleración y frenado del vehículo, entre otras características.

Se ha optado por un procesador Intel NUC modelo NUC7i7DNKE, el cual tiene las especificaciones necesarias para poder desarrollar el proyecto, sus características son:

- Intel Core i7 de 8° generación.
- 4 núcleos, 8 hilos.
- Memoria interna de 16 Gb.
- Conexión Bluetooth.
- 2 puertos HDMI.
- 4 puertos USB.

2. 4 Mapeo y localización del vehículo.

Es posible automatizar el proceso de reconocimiento del entorno y reducir significativamente el tiempo y el costo de los procesos de mapeo mediante el uso de una técnica llamada localización y mapeo simultáneo (SLAM) [1].

SLAM resuelve el problema computacional de mapear un entorno desconocido y al mismo tiempo permite encontrar y realizar un seguimiento de la ubicación del robot dentro de este mapa, esta técnica en dos dimensiones se ha utilizado con frecuencia en la investigación durante ya varios años [2] [15] [16]. Toda la información se calcula en tiempo real y no es necesario tener un conocimiento previo del entorno que rodea al vehículo.

Los principales propósitos al ocupar SLAM son:

1. Crear de manera precisa un mapa del ambiente que rodea al robot.
2. Calcular y mantener el seguimiento de la localización del robot dentro del mapa en todo momento.

Cuando se usa la técnica de SLAM, es necesario que en todo momento el robot conozca las medidas del ambiente que lo rodea. Para ello, se ha implementado el uso de un sensor LIDAR, el cual mide en tiempo real las distancias de los objetos más cercanos al robot en todas las direcciones.

La posición real del robot que se estima en cada momento generalmente es representada como un vector tridimensional, el cual contiene una coordenada en dos dimensiones (x,y) y un valor rotacional que representa la orientación del robot (*yaw*). Cada nueva posición estimada del vehículo tiene una relación con la posición estimada anterior y con la siguiente.

"Cierre de bucle", es un problema que existe al implementar SLAM y se presenta cuando el vehículo móvil vuelve por alguno de los caminos antes recorridos. Esto es que el robot reconoce una ubicación que visitó anteriormente y por consecuencia, debido a los nuevos datos que obtiene en tiempo real, reestablece los datos de esa posición que conocía [16].

Para este proyecto se hará uso del algoritmo "Hector SLAM", el cual es un algoritmo desarrollado en la Universidad Técnica de Darmstadt. La gran ventaja de este algoritmo es que no requiere conocer la odometría del robot para realizar el mapeo y la estimación de la posición en tiempo real, ya que aprovecha al máximo todos los datos que obtiene del láser (RPLIDAR-A3) y provee una estimación en 2D del robot. Una desventaja de este algoritmo, es que no tiene una gran habilidad de respuesta ante el problema del "cierre de bucle" [17], sin embargo, los resultados que se obtienen al implementar este algoritmo son bastante precisos.

3 Prototipo 8X8.

En este capítulo se describirá de manera general el vehículo 8X8, los componentes hardware y software implementados para que trabaje de manera correcta.

3. 1 Descripción general del vehículo.

A los vehículos 8X8 también se les se les conoce como vehículos 8WD, esto es debido a que sus 8 ruedas son consideradas "ruedas motrices". La gran capacidad de carga útil que estos vehículos tienen es una de las características que los distingue de otro tipo de vehículos pesados con menor número de ejes. El hecho de contar con más ejes y por tanto con más ruedas, le permite soportar mayor peso. Asimismo, al tener más fuerza motriz, su movilidad aumenta considerablemente, asemejándose en este aspecto a los vehículos orugas.

La dinámica del vehículo se ha dispuesto de tal manera que para que se mueva con una determinada velocidad y en una determinada dirección, deben ser especificados los siguientes parámetros:

- **Par aplicado a cada rueda.** Dado que las ruedas son independientes en cuanto a tracción, debemos especificar el par que se aplica a cada una de las 8 ruedas en cada momento.
- **Ángulo con el que se gira cada rueda.** Las ruedas, además de ser independientes en cuanto a tracción, también lo son en cuanto a dirección, lo que abre un amplio campo de posibilidades en cuanto a maniobrabilidad. Por ello, también se especificará el ángulo con el que debe girar cada una de las 8 ruedas con respecto a su eje longitudinal.

Los 3 modos de direccionamiento que se han implementado en vehículo son los siguientes:

- Direccionamiento tipo Ackerman.
- Direccionamiento diferencial o *skid steer*.
- Sistema mixto resultado de la combinación de los 2 anteriores.

3.1.1 Direccionamiento tipo Ackerman.

Para este modo de direccionamiento, el par aplicado es el mismo para todas y cada una de las 8 ruedas del vehículo, siendo el ángulo de deriva el factor determinante para el giro del vehículo.

Dado el radio de giro deseado, debemos obtener el ángulo de deriva de cada una de las ruedas que, en esta ocasión, suele ser diferente para cada una de ellas. Sabiendo donde está situado el centro de giro, para que el vehículo describa una trayectoria circular alrededor de dicho centro hay que colocar las ruedas de tal manera que el eje longitudinal de las mismas sea perpendicular a la línea imaginaria que une el centro de gravedad de las ruedas con el propio centro de giro. De esta forma, a una velocidad constante y no excesivamente elevada, el vehículo girará en torno al centro de giro deseado (figura 4).

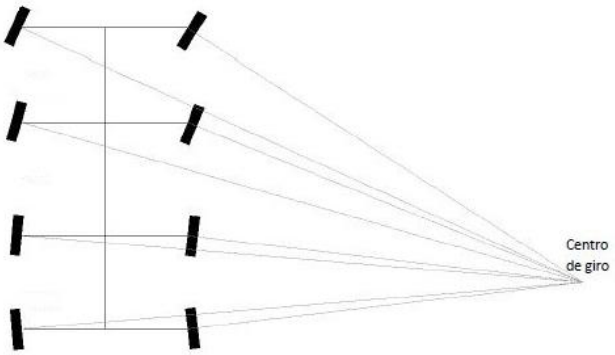


Figura 4. Vista del giro de las ruedas con su centro de giro [18].

En todo momento se ha tenido en cuenta que el centro de giro del vehículo se encuentra siempre en la línea perpendicular al eje longitudinal del vehículo que pasa por el centro geométrico del mismo. Es interesante mencionar que es posible cambiar la posición del centro de rotación dependiendo de la velocidad del vehículo con el objetivo de evitar deslizamientos.

3.1.2 Direccionamiento diferencial o *skid steer*.

El giro en *skid* se basa en la realización de giro de un determinado radio pero sin girar las ruedas, produciéndose el giro por la diferencia de par aplicada a las ruedas de un lado y otro del vehículo.

El sistema de direccionamiento *skid steer* (figura 5) consiste en variar el par aplicado a las ruedas de cada lado del vehículo para que describa trayectorias circulares manteniendo sus ruedas rectas respecto a su eje longitudinal. Las entradas son nuevamente la velocidad y el radio de giro que deseamos obtener. Y en esta ocasión tendremos dos salidas: el par aplicado en el lado izquierdo de los ejes y el par aplicado en el lado derecho de los ejes para describir el radio de giro deseado.

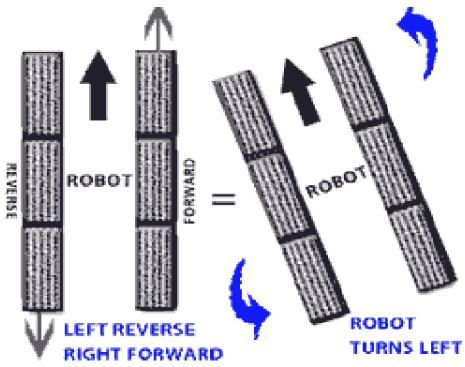


Figura 5. Comportamiento del direccionamiento *Skid Steer*.

Dado que conocemos el radio de giro que debe describir el vehículo y la velocidad con la que se moverá, podemos obtener de manera simple la velocidad angular. Conocida la velocidad angular con la que debe describir la trayectoria el vehículo y el radio de giro, podemos obtener la velocidad necesaria en cada lado del vehículo utilizando la distancia de la rueda de cada lado al centro de masas del propio vehículo, de manera que en dicho centro de masas se alcance la velocidad lineal previamente establecida.

Con estos valores ya es posible conocer qué velocidad lineal deben describir las ruedas de cada lado para que el vehículo responda a las entradas del sistema. Ahora sólo queda formar las salidas del sistema, es decir, transformar dichas velocidades en el par a aplicarle a las ruedas de cada lado del vehículo para ello se ocupa la constante que sirve para transformar velocidad lineal en par motor para nuestro modelo en concreto.

3.1.3 Direccionamiento mixto.

En este tipo de direccionamiento se hará una combinación del direccionamiento Ackerman con el direccionamiento *skid* y así poder optimizar el comportamiento del vehículo.

Si el ángulo de las ruedas marca un determinado giro y además de eso se aplica un par diferente a las ruedas de ambos lados del vehículo, se está aplicando un mayor giro absoluto, por ello mediante un análisis de distintas configuraciones se llegan a determinar los factores que permiten distribuir el giro según los dos métodos, para que el giro global sea el que se quería al inicio.

La implementación de este direccionamiento tiene sus ventajas y desventajas. Es cierto que al tener que calcular en un momento determinado tanto los ángulos de las ruedas como la diferencia de par estamos cargando computacionalmente la tarea del vehículo, pero por el contrario, la carga hardware será menor al tener que realizar menos movimiento, existirá menos diferencia entre las ruedas lo cual provocará un menor desgaste de las mismas, etc.

3.2 Especificaciones del vehículo.

El vehículo 8X8 deberá estar alimentado por baterías LiPo y tener un sistema de mando electrónico. Su tren de movimiento deberá estar controlado por servo-actuadores, contar con una suspensión pasiva y las 8 ruedas deberán tener tracción independiente.

3.2.1 Hardware

El hardware implementado en el vehículo es el siguiente:

- **Motor sin escobillas Crawler** 18.5 que proporciona mayor rendimiento, potencia de frenado y mejor comportamiento a altas temperaturas.
- **Variador Goat 2S Crawler** sin escobillas, que permite el ajuste de los parámetros de zona muerta, velocidad mínima y freno motor. Para regular la velocidad angular se implementó un controlador PI.
- **Transmisión AX10 Scorpion** que permite un amplio rango de relaciones de transmisión sin acción diferencial.
- **Servomotores Hitec HSR-5980SG** de tipo coreless, engranajes de acero y una velocidad de comunicación de 19, 2 kbps.
- **5 Baterías LiPo** 2S (7.4 v), 5000 mAh, 4 por tren de ruedas y una para la placa master.
- **1 placa master** de propia fabricación que se encargará de recolectar la telemetría de cada una de las placas esclavo, y establecerá las consignas de velocidad, dirección y frenado del vehículo.
- **4 placas esclavo**, que tendrán comunicación vía SPI (Serial Peripheral Interface) con la placa master.
  Cada placa esclavo se encargará de:
  - Movimiento del ángulo de la dirección por cada 2 ruedas.
  - Velocidad de las ruedas (control de los motores).
  - Control de los servos de frenado.
  - Las lecturas de las tensiones de las baterías, lo cual permitirá detectar niveles críticos o errores y poder generar alarmas o paradas de emergencia.
  - Modificar el firmware de dicha placa.

Cada placa esclavo contará con 1 led de estado que permite conocer el estado de las comunicaciones y el nivel de la batería. Si todo está correcto parpadeará a una frecuencia de 1 Hz, si la tensión es baja parpadeará a una frecuencia de 4 Hz. Si la tensión de las baterías es crítica quedará encendido sin parpadeo alguno.

Para poder conocer la posición lineal y angular, se le ha incorporado al vehículo una **IMU X-Sens**.

Las magnitudes que se obtendrán para su almacenamiento y tratado son las siguientes:

- **Marca de tiempo** (formato time) desde el inicio de lectura de la IMU. Muy útil a la hora de la representación de las demás magnitudes.
- **Temperatura.** (formato temperature) Si las temperaturas son demasiado altas o demasiado bajas pueden ser causantes de averías. Es interesante obtener esta magnitud para comprobarla en caso de que pudieran producirse averías de tipo mecánico.
- **Orientación.** (formato oe_X) Puede obtenerse en base a 3 modos: quaternion, ángulos de Euler o matriz de orientación. El modo debe definirse durante la inicialización del dispositivo. Por defecto, la orientación se obtiene mediante ángulos de Euler (yaw, pitch, roll).
- **Posición.** (formato p_gps_X) La posición se obtiene en formato LLA. La altitud, latitud y longitud se obtiene en grados con decimales en la precisión que se desee. En la inicialización del programa se establecen 8 decimales de precisión para esta magnitud. La conversión a minutos y segundos de los decimales debe hacerse mediante tratamiento posterior.
- **Velocidad.** (formato v_X) Velocidad en x, y, z, del dispositivo en m=s.

3.2.2 Software

Para el software se estableció una arquitectura que consta de 3 módulos:

- **PC emisor:** el cual suministra las consignas al PC de a bordo (receptor).
- **PC a bordo (receptor):** es el procesador que va en el vehículo, recibe las consignas del PC emisor y las transforma en consignas de bajo nivel para los servos y motores.
- **Vehículo móvil:** recibe las consignas necesarias del PC a bordo para su movimiento, de igual manera, envía información de regreso con su estado actual al PC de a bordo.

El PC emisor se comunica con el PC a bordo por medio de una LAN en la que ambos están conectados, dicha conexión es inalámbrica. Este intercambio de información se da de 2 maneras:

- **Consignas base de direccionamiento:** velocidad, radio de curvatura, modo de direccionamiento e incidencia del sistema de frenado deseado en cada instante.
- **Información de disponibilidad en la LAN** en la que ambos se encuentran conectados, de manera que se pueda controlar la conexión y parar el vehículo en caso de que exista una pérdida de conexión.

Ambas PC's estarán conectadas mediante sockets debido a que es la solución más rápida para activar el modo emergencia si se detectan pérdidas en la conexión. La IP de cada componente será estática con el objetivo de que no varíe durante una ejecución y otra.

Si se detectará alguna pérdida en la conexión entre ambas PC's, la PC a bordo estaría en todo momento tratando de volver a conectarse a la PC emisora, y una vez que lo consiga, el modo emergencia del vehículo se desactivará. La PC a bordo y el vehículo se comunicarán mediante una transmisión serie bidireccional, lo que hace que sea un sistema retroalimentado.

Al vehículo se le ha implementado un sistema de sonido que permitirá reconocer cuando el vehículo se encuentre parado esperando una consigna o también cuando el vehículo se encuentre en movimiento. Cuando el 8X8 se encuentre parado imprimirá una capa de sonido en la que se reproducirá un motor a ralentí, cuando este se mueva, producirá un sonido a efecto de aceleración. El software utilizado para el sonido de motor ralentí fue la librería SDL-Mixer y para el efecto de aceleración fue OpenAL.

4 Localización y mapeo simultáneos (SLAM)

En este capítulo se detallará una descripción general del funcionamiento de SLAM, se abordarán algunas de las diferentes técnicas SLAM que se ocupan comúnmente y se definirá la técnica SLAM a ocupar en este proyecto. Se sustentará esa decisión al mostrar cómo se ha implementado durante el desarrollo del proyecto, así como también se mostrarán las pruebas realizadas al modelo en ROS y al vehículo físico.

4.1 Descripción general.

La característica principal de un robot móvil autónomo es el poder generar un mapa del entorno que lo rodea y simultáneamente poder localizarse dentro de ese mapa, para ello, una de las técnicas más potentes que existen es la de localización y mapeo simultáneos (SLAM).

La implementación de sensores son clave para las aplicaciones que involucran SLAM, sin embargo, estos tienen ciertas limitaciones a la hora de ocuparlos, y debido a eso, se tendrán que establecer diversas soluciones para aplicarlos sin que afecten al desarrollo del sistema. Dentro de la gran gama de sensores que se encuentran disponibles, los más usados son las cámaras, sonares, radares, láseres, sensores cartográficos, sensores ultrasónicos, IMU, etc.

Al implementar SLAM se estima continuamente el estado del robot basándose en el control, mediciones y estimaciones antiguas. Por ello, se han desarrollado diversos algoritmos que permiten realizar SLAM siguiendo diferentes metodologías. Algunas de las técnicas SLAM más usadas son [25] [26]:

Filtro Kalman Extendido (EKF). El filtro Kalman es usado en numerosas aplicaciones, y es comúnmente implementado en la navegación y control de vehículos autónomos. La primera implementación del EKF en SLAM se dio en un artículo de 1986 [19], una detallada descripción de EKF-SLAM y varias implementaciones del mismo pueden ser consultadas en [20] [22] [23].

Debido a que EKF aplica modelos lineales a observaciones y modelos no lineales, esto debido a que la convergencia y la consistencia solo podría ser garantizada en el caso lineal. Esto no generará un mayor problema si el modelo real puede interpretarse como un modelo cercano a lo lineal [21].

Otro detalle que se debe tomar en cuenta cuando se pretende usar EKF-SLAM, es que a medida que el mapa crezca, la complejidad de la representación del EKF aumentará tanto en memoria como en tiempo computacional. Esto es debido a que la matriz de covarianza tiene que ser actualizada en cada observación que hace el robot a fin de que se mantenga una relación entre todas las variables de estado [24].

Sin embargo, hoy en día existen diversas variantes del EKF-SLAM que permiten procesar grandes cantidades de datos de manera más eficiente.

Filtro de Partículas. Esta técnica presenta una solución para modelos no lineales [15], es un algoritmo que permite estimar distribuciones de probabilidad de estado en forma de "partículas", estas partículas pueden considerarse como muestras de una distribución posterior desconocida [25].

Esta técnica básicamente está basada en diversos pasos. Durante la primera medición las partículas tendrán un determinado "peso" basado en cuan bien estas coincidan con lo que dice la medida tomada. De esta manera, las áreas que tengan una alta probabilidad tendrán más partículas que las áreas con menos probabilidad. En un principio las partículas son arbitrariamente escogidas y durante su implementación su "peso" se irá asignando debido a la probabilidad que tengan.

Un buen ejemplo de este filtro es el algoritmo de localización "Monte Carlo", su implementación es rápida y precisa cuando se conoce el ambiente que rodea el robot. Sin embargo, para generar mapas de grandes extensiones se requiere un alto coste computacional debido a la cantidad de partículas que se deben generar.

En [27] se menciona que el filtro de partículas puede ser implementado como una solución para mejorar la velocidad del algoritmo basado en EKF-SLAM.

Odometría y mapeo LIDAR de baja deriva y en tiempo real (LOAM). Este algoritmo realiza un escaneo de características de puntos los cuales clasificará en "bordes afilados" o "superficies planas" para encontrar similitudes entre cada escaneo. Las características son extraídas calculando la rugosidad de cada punto en su región local. Los puntos que tengan un valor de rugosidad elevado serán seleccionados como "bordes afilados", y por el contrario, los puntos que tengan rugosidad baja serán considerados como "superficies planas".

Para tener un buen rendimiento durante la implementación del sistema se debe ejecutar el algoritmo en dos partes, una parte se ejecutará a alta frecuencia y se hace una estimación a baja precisión y otra parte se ejecutará a baja frecuencia, pero se hará una estimación a alta precisión. Al fusionar las dos estimaciones se producirá una sola estimación a alta y baja frecuencia [28].

Básicamente, una parte del algoritmo ejecuta la odometría a alta frecuencia y baja fidelidad para estimar la velocidad del LIDAR. La otra parte del algoritmo se ejecuta a baja frecuencia pero alta fidelidad para el registro de la nube de puntos. Este método está basado en el método de extracción y coincidencia de características.

La extracción de características es limitada y esto permitirá que el algoritmo se desarrolle de manera rápida y no consuma mucha memoria computacional.

SLAM basado en gráficos (Graph-SLAM). Graph-SLAM permite realizar en tiempo real SLAM 2D o 3D utilizando una amplia gama de sensores y plataformas. Este método tiene como idea básica que las observaciones y localizaciones del robot se almacenen como nodos. Los bordes representarán una relación entre los nodos, la cual puede ser el movimiento entre dos posiciones o la posición en la que se realizó una observación. Dichos bordes pueden verse como limitaciones, y al relajarlos, permitirán tener una mejor perspectiva de la estimación del mapa y de la trayectoria [25].

Este sistema ha sido extensamente explorado y experimentado, sin embargo, no se han conseguido resultados deseables debido al problema de "cierre de bucles" (figura 6), dicho problema afecta a gran parte de los algoritmos de SLAM.

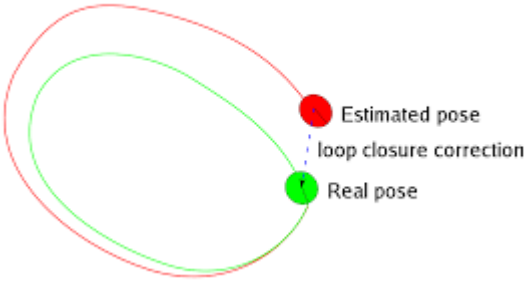


Figura 6. Ejemplo de error de posición al no corregir el problema de "cierre de bucles".

Detectar los "cierres de bucles" es crucial para agregarle robustez al algoritmo SLAM. Este problema consiste en detectar cuando un robot ha vuelto a visitar un punto conocido después de haber navegado por terreno nuevo. Este problema es considerado como uno de los más importante a superar a la hora de implementar SLAM.

Existen algoritmos que logran corregir un poco el problema de los "cierres de bucles", como en [29] [30] [31]. Sin embargo, algunos algoritmos no toman en cuenta este problema y confían en la precisión de dicho algoritmo para evitar este problema.

Los "cierres de bucles" son fácilmente detectables debido que durante la ejecución de SLAM, la misma representación física del mapa se presenta en dos ubicaciones diferentes. Esto provoca que el mapa tenga inconsistencias como dobles paredes, la posición real del robot no sea la correcta, la planificación de la ruta se vuelva más complicada, se presenten de manera errónea los mismos objetos en varias ocasiones pero en diferentes ubicaciones, etc.

4.2 Hector-SLAM.

El algoritmo que se utilizará para hacer SLAM es la librería de código abierto *hector_slam*, este algoritmo no utiliza la odometría del vehículo para crear un mapa de su entorno, sino que utiliza los datos que recibe del sensor LIDAR. Este algoritmo implementa el llamado FastSLAM [27] (SLAM rápido), el cual es un algoritmo que de forma recursiva estima la distribución posterior sobre la posición del robot y las ubicaciones de los puntos de referencia, y los escala de manera logarítmica con el número de puntos de referencia en el mapa.

En SLAM para desarrollar el mapa se utiliza la técnica llamada "mapeo de cuadrícula ocupada" (*occupancy grid mapping*), este tipo de mapas se definen como una secuencia de celdas donde cada celda tiene su ocupación designada por la ecuación aleatoria binaria:

$$m_{x,y}: \{libre, ocupad\} \rightarrow \{0,1\}$$

Donde m se define como la ocupación dada por las coordenadas en 2D de x, y, dicha ecuación puede tener el valor *libre* (0) u *ocupada* (1) y estos valores se mapean dentro de un espacio real (figura 7).

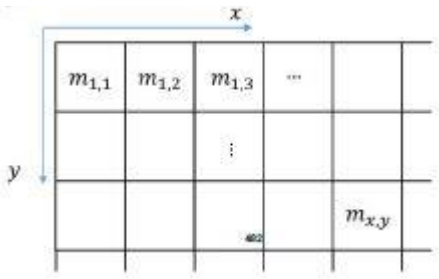


Figura 7. *Occupancy Grid Map*

Por medio del teorema de Bayes a cada celda se le asigna una probabilidad de estar o no ocupada, esto permitirá que el robot puede conocer si hay o no una obstrucción en su camino, dicha probabilidad irá cambiando mientras el robot vaya explorando su entorno. A esta probabilidad se le agregará la medida que va tomando en cada momento el sensor LIDAR, dicha medida será representada como:

$$z \sim \{0,1\} \rightarrow \{libre, ocupada\}$$

El sensor LIDAR considerará una *celda ocupada* a aquella celda que detecte que se encuentre frente a él y que no permita que el láser del sensor la atreviese, lo que llevará a que construya una pared y confirmará que en ese lugar hay una oclusión. Y considerará como *celda libre* a aquella donde el escaneo (láser) del LIDAR tiene libertad para pasar entre ella, ya que no detecta ningún obstáculo que se lo impida (figura 8).

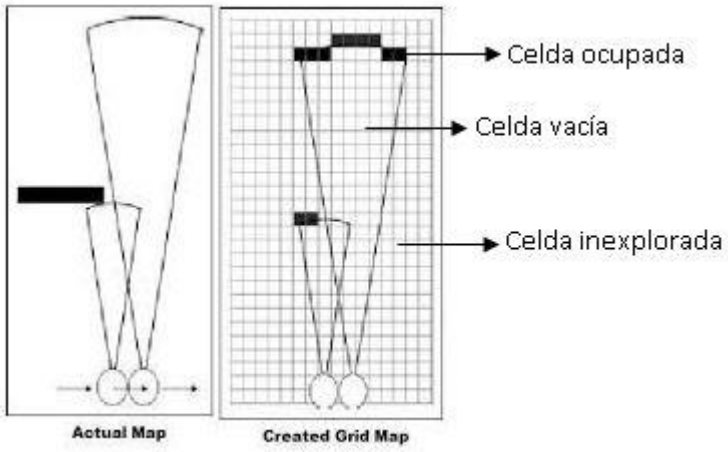


Figura 8. Celdas ocupadas, vacías y sin explorar

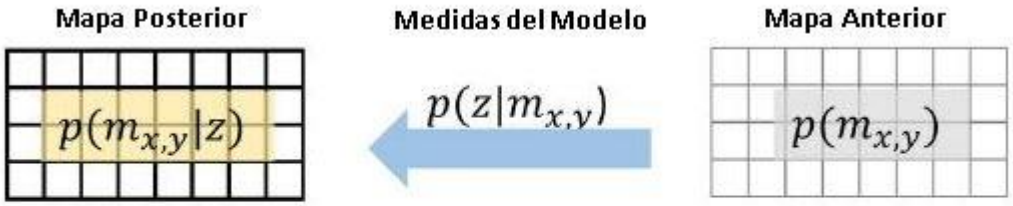
Dicho lo anterior, se puede representar la ecuación del modelo como:

$$p(z|m_{x,y})$$

Donde p será la probabilidad obtenido al aplicar el filtro Bayesiano.

En la figura 9 se detalla el proceso de "mapeo por cuadrícula ocupada", en un principio se tendrá un mapa con la probabilidad de encontrar celdas libres y ocupadas (mapa anterior), posteriormente haciendo uso de las medidas obtenidas por el sensor LIDAR se ocupará la ecuación del modelo (medidas del modelo). Esta ecuación del modelo tendrá una probabilidad al observar un 1 o 0 en z, dicha probabilidad estará condicionada del punto de coordenadas donde

se encuentre ($m_{x,y}$). Al final se utilizará esta medida para actualizar el conocimiento que se tenía acerca de la ocupación del mapa (mapa posterior).



$$p(m_{x,y}|z) = \frac{p(z|m_{x,y}) * p(m_{x,y})}{p(z)}$$

Figura 9. Proceso de *Occupancy Grid Mapping*

Sin embargo, no es bueno confiar completamente en este modelo ya que puede mostrar sus inconvenientes mientras el análisis del mapa va aumentando considerablemente, por ello se implementa una manera simplificada y recursiva para mejorar estas probabilidades. Dicha simplificación se basa en probabilidades logarítmicas, se tendrá una probabilidad logarítmica para definir una celda ocupada y otra para definir una celda libre, y se realizará una actualización del mapa siguiendo estas probabilidades:

$$Caso\ 1: celdas\ con\ z = 1\ (ocupadas)\ \ \log odd\_occ := log\frac{p(z=1|m_{x,y}=1)}{p(z=1|m_{x,y}=0)}$$

$$Caso\ 2: celdas\ con\ z = 0\ (libres)\ \ \log odd\_free := log\frac{p(z=0|m_{x,y}=0)}{p(z=0|m_{x,y}=1)}$$

Actualización del mapa:

Celdas con z=1: log odd = log odd_sensor + log odd_occ

Celdas con z=0: log odd = log odd_sensor - log odd_free

El principio fundamental del algoritmo Hector SLAM es hacer la transformación entre 2 diferentes medidas del láser. Esta técnica de escaneo básicamente está basada en el método de registro de imágenes para visión por computadora.

El algoritmo inicia recibiendo el primer escaneo del mapa, y los subsecuentes escaneos se irán incorporando al mapa para encontrar la transformación entre ellos (figura 10).

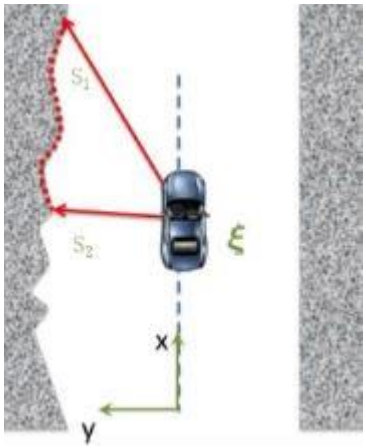


Figura 10. Escaneo inicial del vehículo

$$Posición\ del\ Robot \rightarrow\ \xi = \left(p_x, p_y, \Psi\right)^T$$

$$\xi^* = argmin_\xi \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2$$

Donde $\boldsymbol{\xi}$ representa la posición del vehículo, la función $\boldsymbol{S_i}$ representa las coordenadas finales del vehículo en el punto $\boldsymbol{S_i = (p_{,x}, p_{,y,}\ \psi)^T}$, $\boldsymbol{n}$ representa el número de escaneo, la función $\boldsymbol{M}$ representa el valor del mapa dadas las coordenadas por $\boldsymbol{S_i}$, como se mencionó anteriormente este valor será 1 o 0 según corresponda .

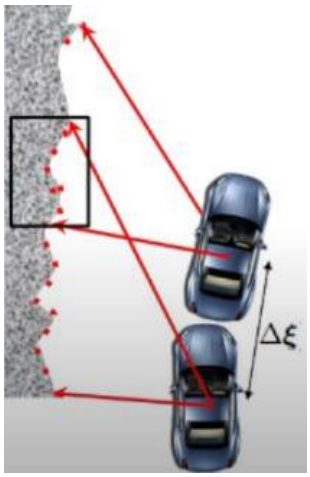


Figura 11. Incremento de posición inicial $\Delta\xi$

Con una $\boldsymbol{\xi}$ inicial estimada, el algoritmo necesitará estimar un $\boldsymbol{\Delta\xi}$ (figura 11) el cual permitirá optimizar el error de medida de acuerdo a:

$$\sum_{i=1}^{n} [1 - M\left(S_i(\xi + \Delta\xi)\right)]^2 \rightarrow 0$$

Se expande la ecuación utilizando la expansión de Taylor de primer orden para la función $\boldsymbol{M(S_i(\xi+\Delta\xi))}$, y resolviéndola en función de $\Delta\xi$ obtendremos la ecuación Gauss-Newton, la cual nos dará la evaluación del cambio de posición con respecto a los instantes anteriores de tiempo.

$$\sum_{i=1}^{n}[1 - M(S_i(\xi))-\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\Delta\xi\ ]^2 \to 0$$

$$\Delta\xi = H^{-1}\sum_{i=1}^{N}[\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}]^T[1 - M(S_i(\xi))]$$

Donde la matriz Hessiana denota:

$$H = [\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}]^T\ [\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}]$$

La ecuación de la matriz Hessiana dependerá del gradiente del mapa en alguna coordenada P_m: $\nabla M(P_m)$.

Hector-SLAM ocupa un método de visión por computadora llamado pirámide de imágenes, en la cual los mapas se almacenan con diferentes resoluciones (figura 11). La generación de varias resoluciones se da al producir versiones escaladas en cada escaneo, lo cual proporciona una mayor efectividad computacional y brinda mayor consistencia entre las capas. Mientras mayores escaneos existan, la representación final será más precisa.

Figura 12. Resoluciones de la representación de un mapa (17)

4.3 Desarrollo e implementación.

Habiendo descrito en los capítulos anteriores cómo funciona la paquetería de software libre ROS, la técnica SLAM, el algoritmo Hector-SLAM, el software y hardware a implementar, etc., se procede a desarrollar e implementar el modelo a computadora con todas las características antes mencionadas, para después realizar test en un ambiente real con un vehículo de pruebas.

Este modelo del vehículo 8X8 se ejecutará en ROS para ver cómo se comporta con el algoritmo Hector-SLAM junto con la implementación de un láser RPLIDAR-A3, se decidió por este modelo de láser debido a su alta efectividad sensorial y bajo costo.

La fabricación del modelo digital del vehículo 8X8 se hizo desde cero, el chasis es un rectángulo al cual se le añadieron valores de peso e inercia, a las ruedas se les añadió la geometría de las ruedas de la paquetería de software libre del robot móvil Summit-XL de Robotnik [33] ya que tienen características muy similares a las ruedas del vehículo 8X8, de igual forma se les agregó valores de peso e inercia a cada una y una transmisión para unirlas (figura 13).

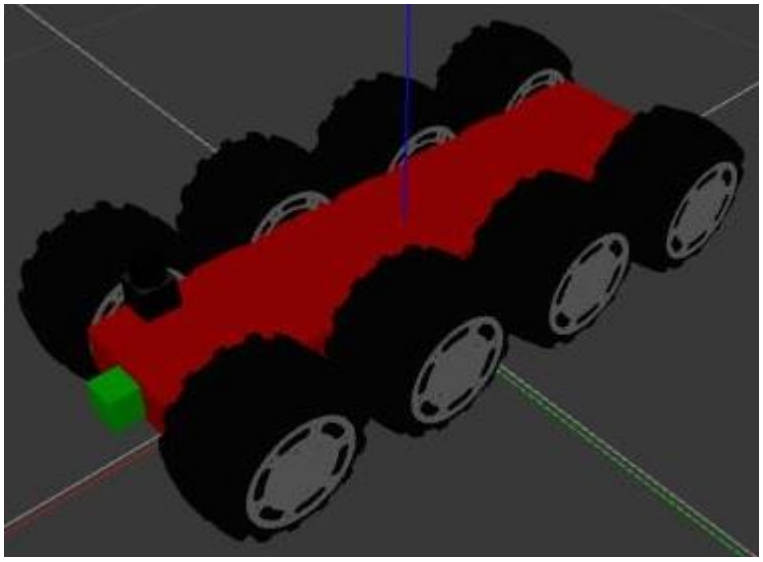


Figura 13. Modelo a computadora del vehículo 8X8.

Gazebo proporciona diversos *plugins* para el accionamiento del diferencial y con ello generar una configuración específica para el movimiento del vehículo. Dichos *plugins* aceptan comandos de velocidad y publican información sobre la odometría del vehículo. Durante el desarrollo de este proyecto se hicieron pruebas con los diferentes *plugins* que ofrece la librería de Gazebo (*Diff Drive*, *Skid Steer* y *Planar Move*), optando por el controlador "*Skid Steer*" ya que es el que más se asemeja al movimiento real del 8X8.

Para el láser se ocupó la geometría de la paquetería libre del láser Hokuyo [34] y para su controlador se ocupó la paquetería libre que ofrece Slamtec del láser RPLIDAR [35]. Para realizar las pruebas en ROS se utilizó el mundo en Gazebo que proporciona la paquetería libre del TurtleBot [32] ya que permite un marco cercano a la realidad y cuenta con diversas variables en su entorno (figura 14).

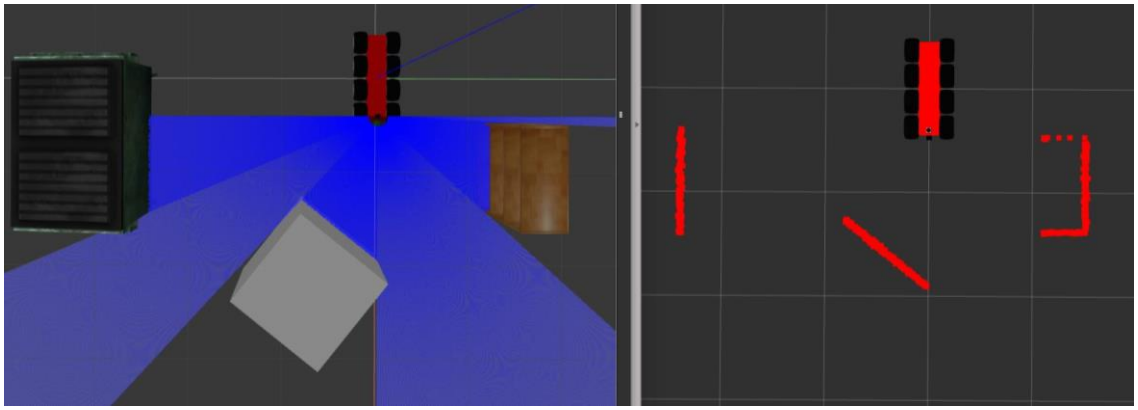


Figura 14. Visualización del LIDAR en Gazebo y Rviz.

4.3.1 Pruebas en simulación.

Antes de realizar cualquier prueba en simulación, se comprobó que las 8 ruedas del vehículo se movieran adecuadamente y de manera independiente (figura 15). Se comprobó que todos los ejes y centros de gravedad estuvieran bien ubicados, que los elementos visuales y de colisión del vehículo estuvieran bien descritos y simulados (figura 16).

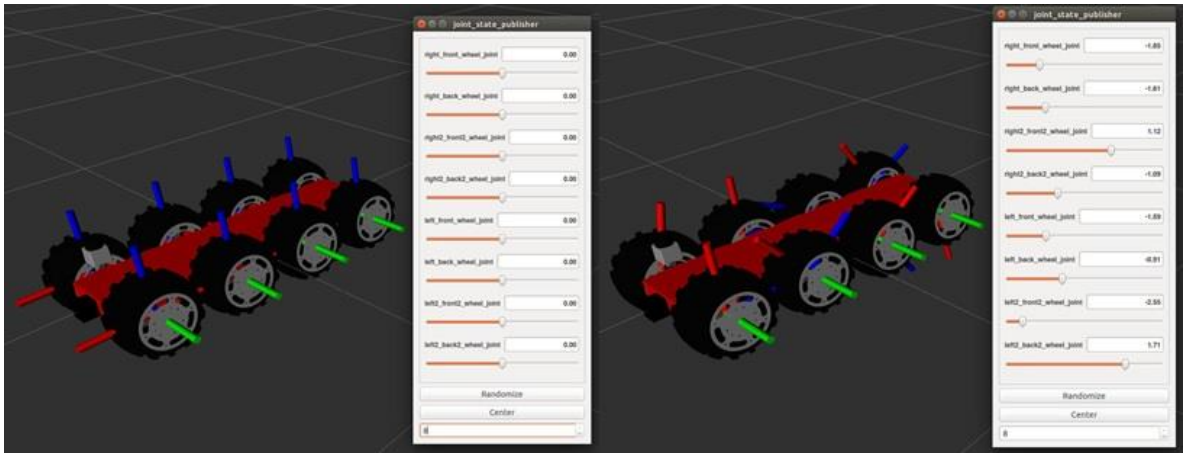


Figura 15. Revisión del movimiento independiente de cada rueda ocupando el *Joint State Publisher* [39].

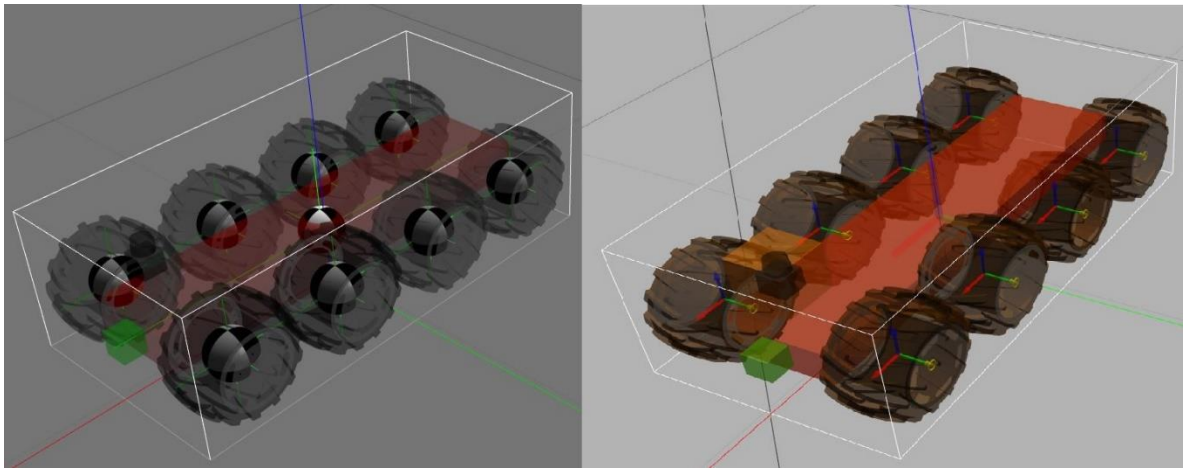


Figura 16. Centros de masa, ejes y elementos visuales y de colisión.

Como se mencionó en capítulos anteriores, para la ejecución del método SLAM se utilizó el algoritmo Hector-SLAM, el cual puede ser consultado en [36]. En la figura 17 se puede observar al vehículo en su posición inicial ejecutando el algoritmo Hector-SLAM. En las primeras pruebas el vehículo será teleoperado por medio del teclado haciendo uso del *Trutlebot keyboard teleop node* [37], posteriormente se harán 2 programas en Python en los cuales se le enviarán comandos al robot y se comprobará que este los ejecute de manera correcta.

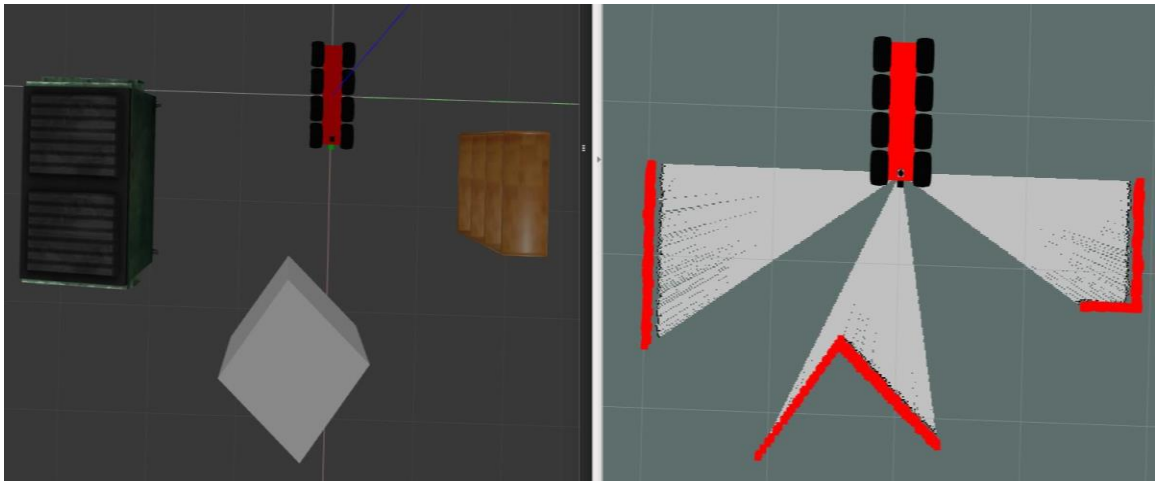


Figura 17. Vehículo en posición inicial ejecutando Hector SLAM.

En las figuras 18, 19 y 20 se puede observar el proceso de creación del mapa 2D hecho por el robot al teleoperarlo con el teclado, el mapa creado con Hector-SLAM es bastante preciso, tomando en cuenta que este algoritmo no pone especial atención al problema de "cierre de bucles". Dicho problema se notó más durante las pruebas en un ambiente real, las cuales se mencionarán más adelante.

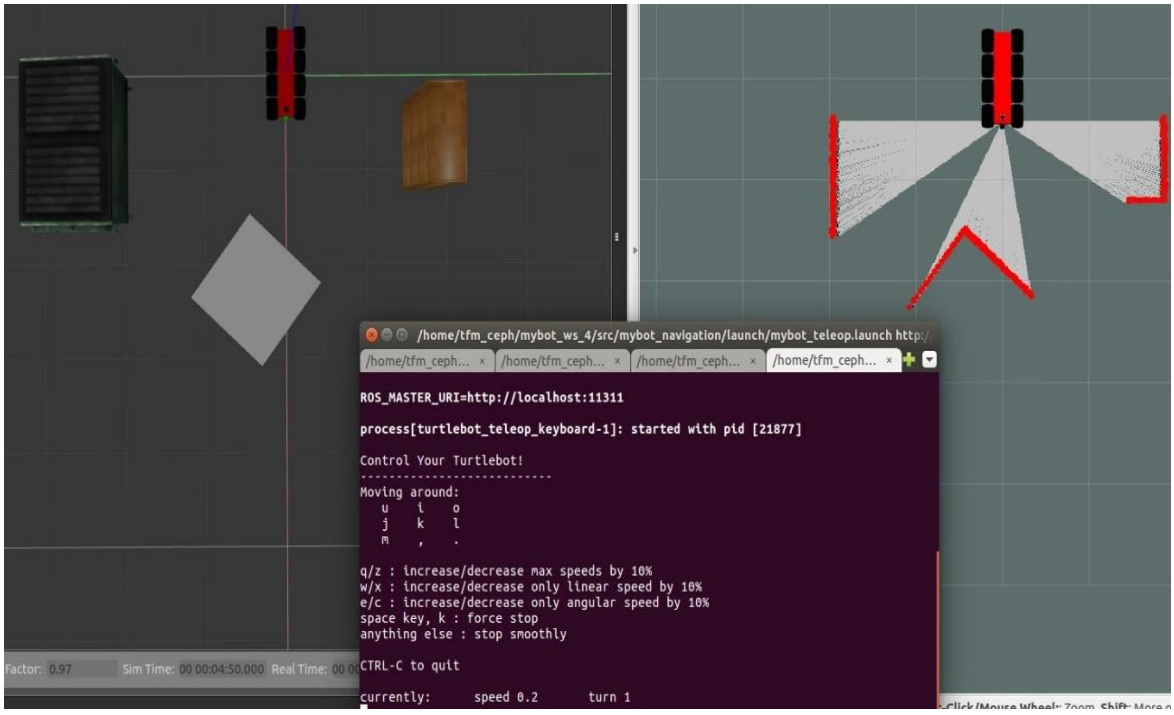


Figura 18. Vehículo teleoperado posición inicial ejecutando Hector SLAM

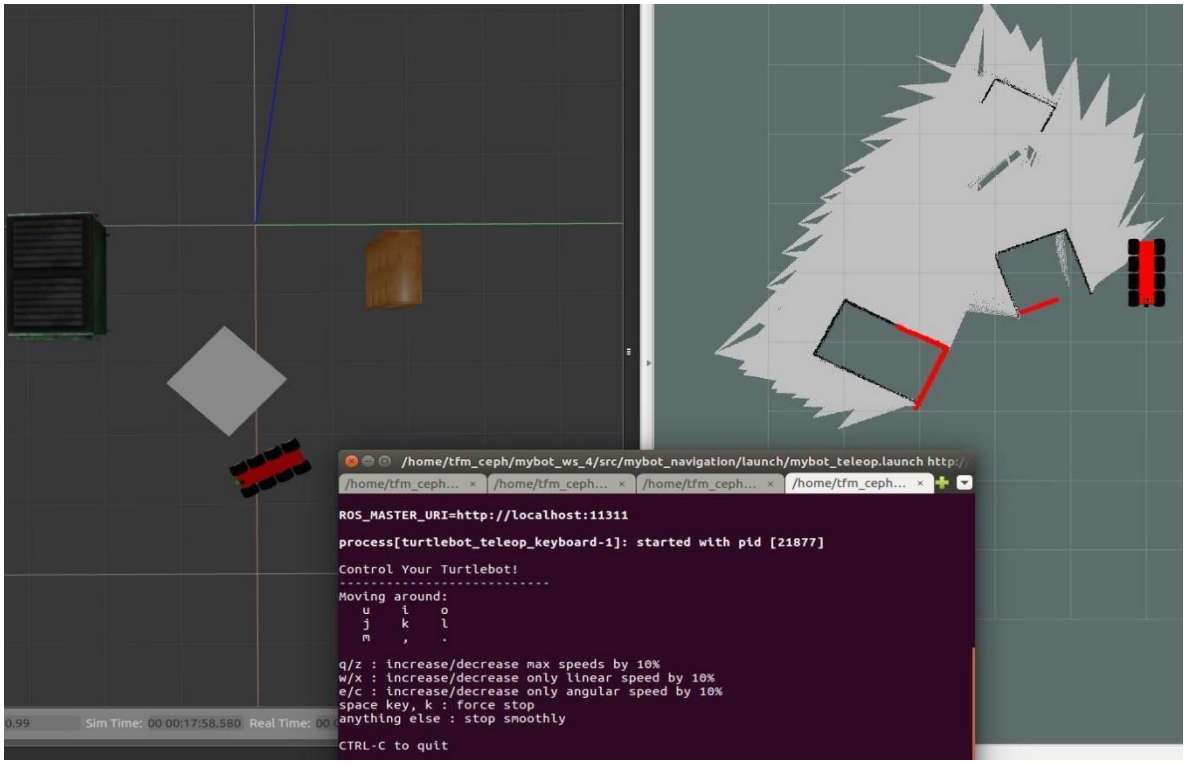


Figura 19. Vehículo teleoperado realizando mapa 2D con Hector SLAM

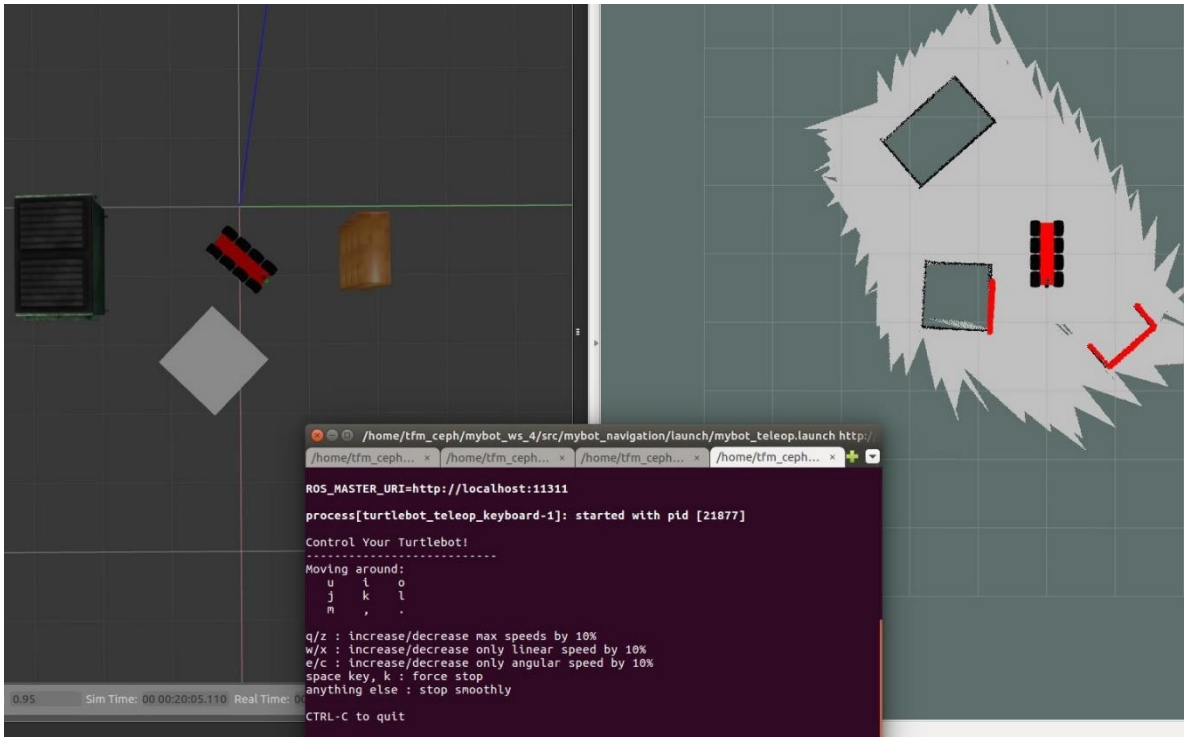


Figura 20. Mapa en 2D completo con Hector SLAM

Se hicieron 2 programas de control en Python los cuales funcionan enviándole comandos al vehículo y así verificar que este es capaz de moverse de acuerdo al entorno de coordenadas que le rodea. En el primer programa se analiza el control del giro del vehículo (*yaw*), se le envía un comando de movimiento angular en Z para que gire primero hacia la derecha y luego hacia la izquierda en intervalos de 10 segundos. Estos giros los hará mientras ejecuta el algoritmo Hector-SLAM (figura 21).

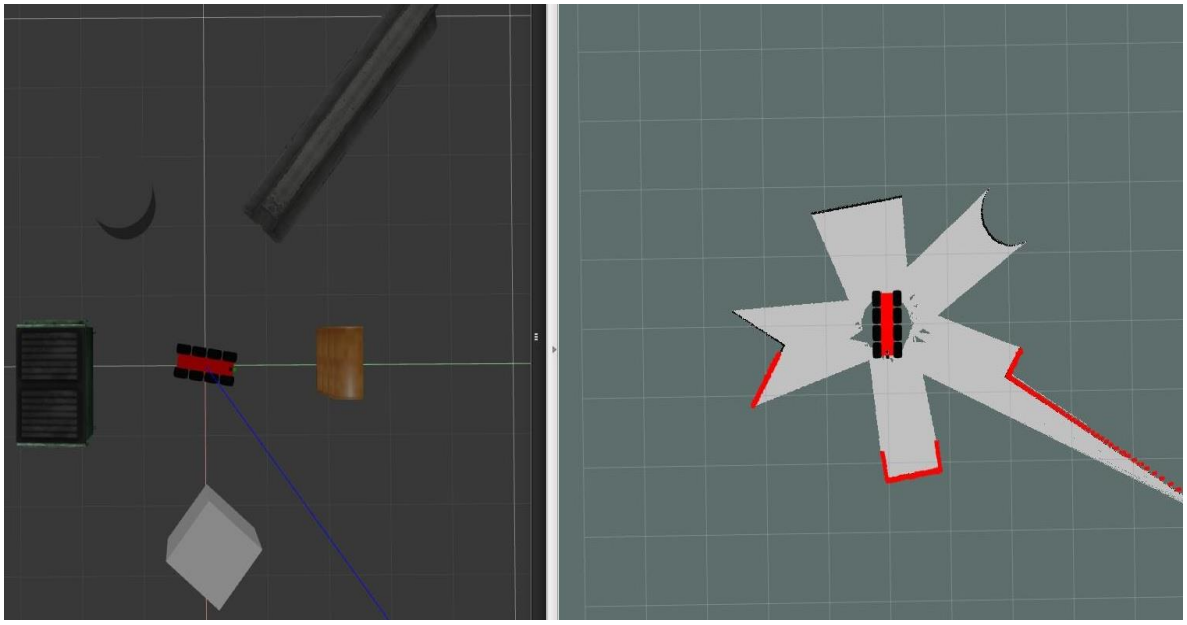


Figura 21. Comprobación del *yaw* en el vehículo 8X8.

Al comprobar que el *yaw* lo hace de manera correcta, se ejecuta el segundo programa. Este segundo programa envía un comando de *go to goal*, dicho comando está compuesto por una coordenada (x,y) la cual el robot debe ejecutar hasta alcanzarla (*goal*). El robot estando en posición inicial, haciendo uso de su dirección deslizante (*skid steer*) girará sobre su propio eje y realizará un escaneo del mundo hasta encontrar la dirección en la que está la coordenada que se le solicita. Una vez que encuentra esta coordenada se dirigirá hacia ella haciendo los giros angulares en z (*yaw*) necesarios hasta llegar a la meta indicada. De igual manera, este programa se ejecutará en conjunto con el algoritmo Hector-SLAM.

El comando que se le envía al robot son las coordenadas (2.5,2.5), cuando alcance esa meta se quedará en reposo esperando recibir otro comando con nuevas coordenadas. En la figura 22 se muestra que el robot al recibir el comando inicial de coordenadas (2.5,2.5) escanea el ambiente que lo rodea para encontrarla, en la figura 23 se observa como el robot se ha movido hacia el objetivo hasta alcanzarlo, y en el trayecto ha ido generando un mapa de su entorno detectando los objetos a su paso.

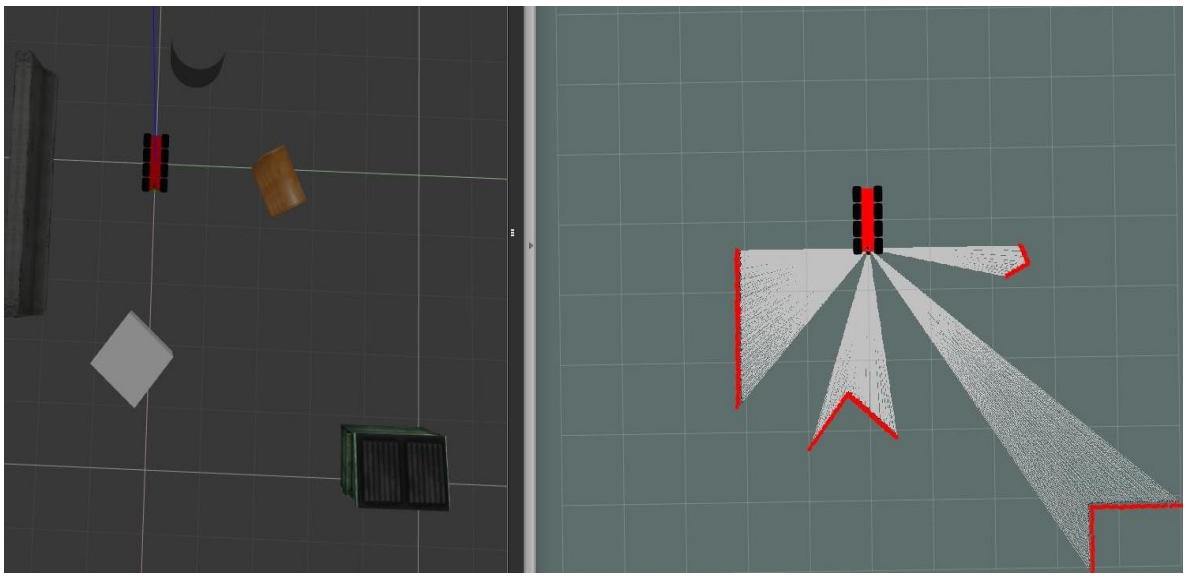


Figura 22. Vehículo recibiendo comando (2.5,2.5)

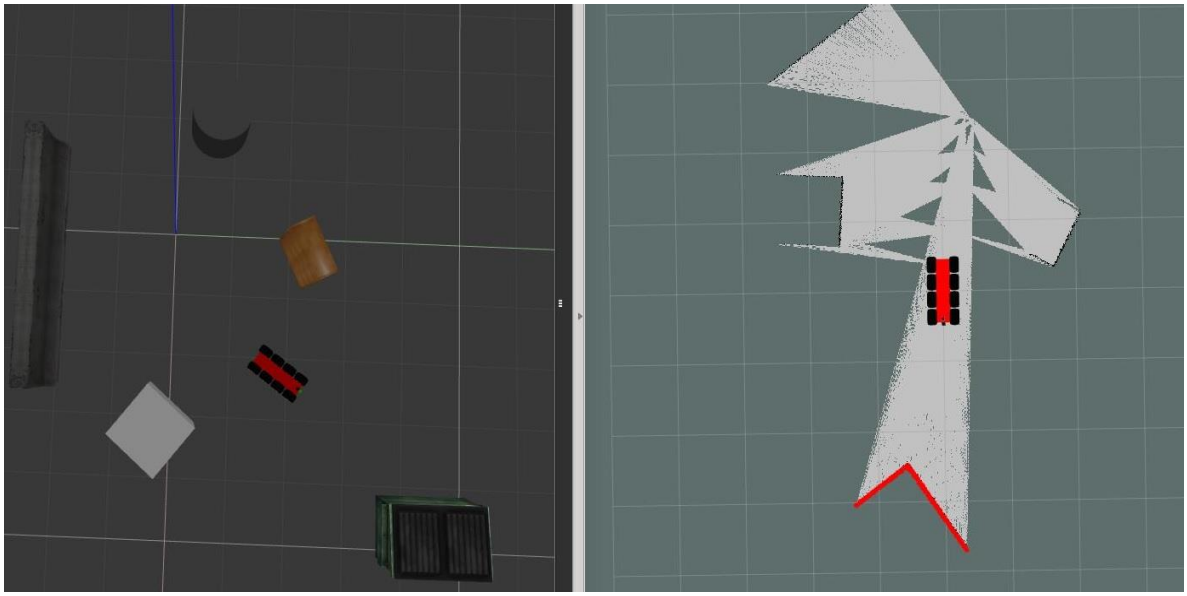


Figura 23. Vehículo llegando al comando (2.5,2.5)

Con estas pruebas queda verificado el funcionamiento correcto del modelo del vehículo en Gazebo y Rviz (odometría, láser, etc), así como también se ha podido profundizar en la implementación del algoritmo Hector-SLAM (mapeo y localización del robot móvil). Cabe destacar que en estas simulaciones no se tomaron en cuenta las dimensiones del mundo en el que opera el robot, así como tampoco las distancias entre los objetos que le rodean. Este tipo de pruebas más específicas se llevarán a cabo a continuación en un ambiente real controlado utilizando un vehículo de pruebas.

4.3.2 Pruebas en ambiente real.

Una vez que en simulación se ha comprobado el uso, los alcances y las limitaciones del algoritmo Hector SLAM, se proceden a realizar ensayos físicos con un modelo de pruebas.

Para las pruebas físicas se ocupará un vehículo de 4 ruedas (figura 24) con direccionamiento Ackerman y mando a distancia por control remoto, las características de hardware y software de este vehículo no son exactamente las mismas que las del vehículo 8X8, pero para poder realizar las pruebas de desempeño del algoritmo Hector-SLAM en un ambiente real este vehículo será más que suficiente.

Figura 24. Vehículo de pruebas

Al vehículo de pruebas se le acondiciona en la parte frontal el escáner láser RPLIDAR-A3 y en la parte central el procesador que ejecutará el programa de SLAM y una batería para alimentarlo.

Serán 5 los ensayos que se realizarán:

1. Test 1: Vehículo en movimiento en línea recta a baja velocidad sin obstáculos.
2. Test 2: Vehículo en movimiento en línea recta a baja velocidad con obstáculos.
3. Test 3: Vehículo en movimiento en línea recta a alta velocidad con obstáculos.
4. Test 4: Vehículo en movimiento en zigzag a velocidad moderada con obstáculos.
5. Test 5: Vehículo en movimiento generando un mapa en 2D de todo un entorno real.

En todas las pruebas mencionadas se ejecutará el algoritmo Hector-SLAM para comparar su comportamiento en diferentes escenarios y situaciones. Todos los test se realizarán en ambientes controlados y dimensionados, cuyos resultados serán descritos en los apartados siguientes.

4.3.2.1 Test 1.

El test se realizará en un pasillo afuera del edificio L1 dentro las instalaciones de los laboratorios de la Universidad de Sevilla. Se ha escogido este sitio debido a que muy pocas personas transitan por ahí, tiene muy buena iluminación y ofrece un terreno estable. El espacio a usar tiene las dimensiones descritas en la figura 25.

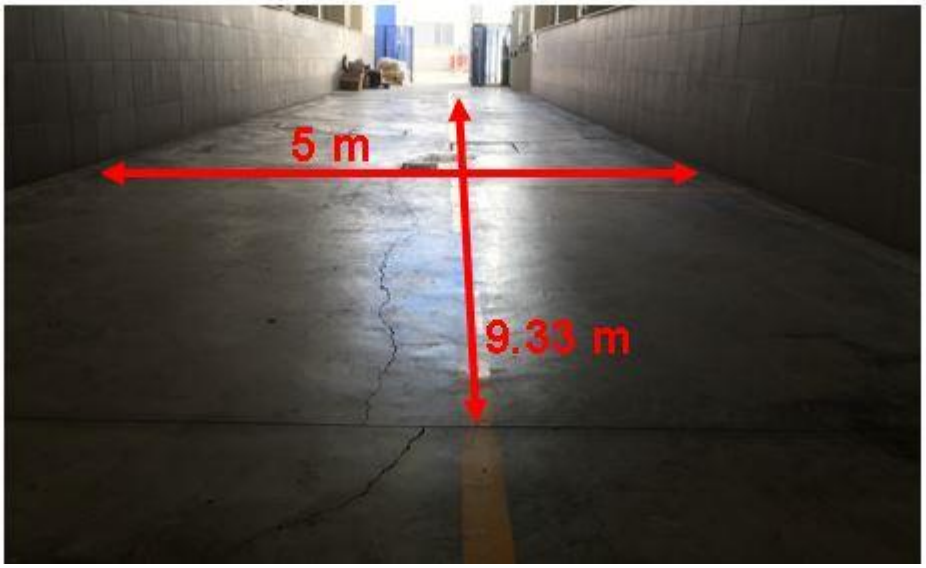


Figura 25. Dimensionado del área de pruebas

La prueba se realizará de la siguiente manera:

- Se ubicará el vehículo en una posición dentro del entorno designado para hacer la prueba (figura 26). Dicho espacio se tomará como el punto inicial (punto 1).
- El vehículo recorrerá una distancia de 9.33 metros en línea recta a baja velocidad hasta alcanzar el punto final (punto 2).
- Durante todo el recorrido del vehículo se tratará de mantener la velocidad constante y no habrá ningún obstáculo u objeto a su alrededor.
- Desde el punto 1 hasta el punto 2 se ejecutará el algoritmo Hector-SLAM para simultáneamente mapear y localizar el vehículo durante todo el recorrido.

Resultados obtenidos:

- Los primeros escaneos que efectúa antes de comenzar a moverse los realiza de manera correcta ya que muestra en el mapa gran parte del entorno que le rodea. Se detallan con bastante precisión las paredes a los costados y el suelo (figura 27 y 28).
- En el punto inicial (punto 1), el vehículo logra ubicarse correctamente en el mapa dentro del espacio donde se encuentra (figura 28).
- Cuando el vehículo comienza a desplazarse hacia el punto final (punto 2), se observa que el mapa no cambia mucho a pesar de que el vehículo se encuentra en movimiento (figura 29).
- El láser en todo momento va detectando las 2 paredes que tiene a los costados, sin embargo, al ser idénticas durante todo el recorrido estas en no tienen ningún cambio aparente en el mapa.
- Con la paquetería *rf2o_laser_odometry* [39] se revisa la odometría del láser y se observa que este va detectando un movimiento en el plano 2D, aunque el mapa muestre todo lo contrario.
- Tal y como se mencionó en apartados anteriores, el algoritmo Hector-SLAM no utiliza odometría para hacer el mapa, sino que utiliza la información que recibe del láser. Debido a que durante el recorrido del punto 1 al punto 2 la información que recibe del láser es siempre la misma (mismas superficies de paredes, suelo, sin ningún objeto alrededor o cambio aparente del entorno), es decir, no existen puntos de referencia que el láser pueda extraer del entorno para que el algoritmo pueda realizar el mapa, por ello el mapa es siempre el mismo.

- Debido a que el mapa es el mismo desde el punto 1 al punto 2, la posición inicial y final del vehículo dentro del mapa también será la misma, como si nunca se hubiera movido (figura 30).

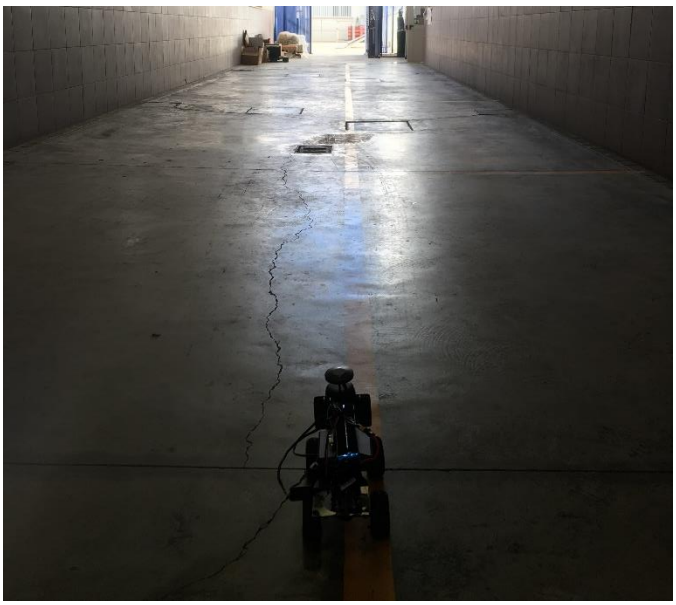


Figura 26. Posición inicial del Test 1

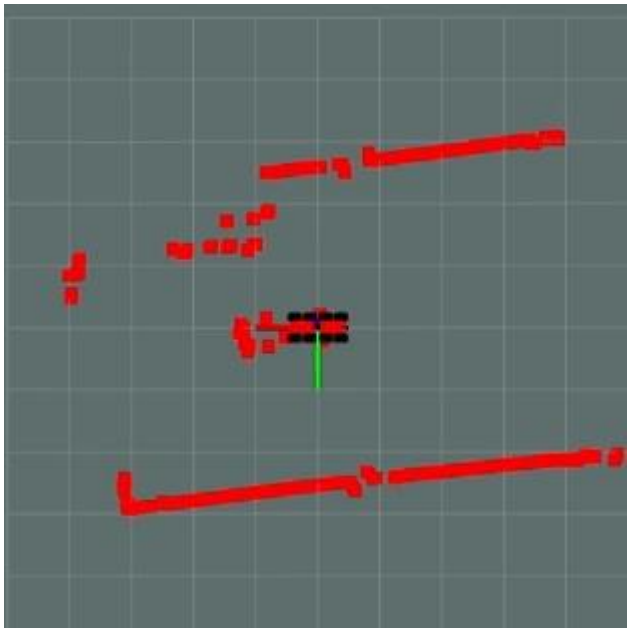


Figura 27. Ejecución del láser en punto inicial Test 1

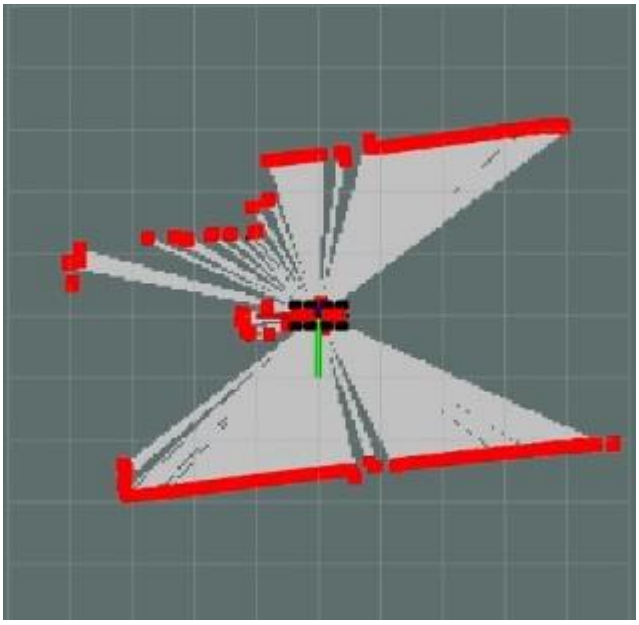


Figura 28. Ejecución del algoritmo Hector-SLAM en punto inicial (reposo) en Test

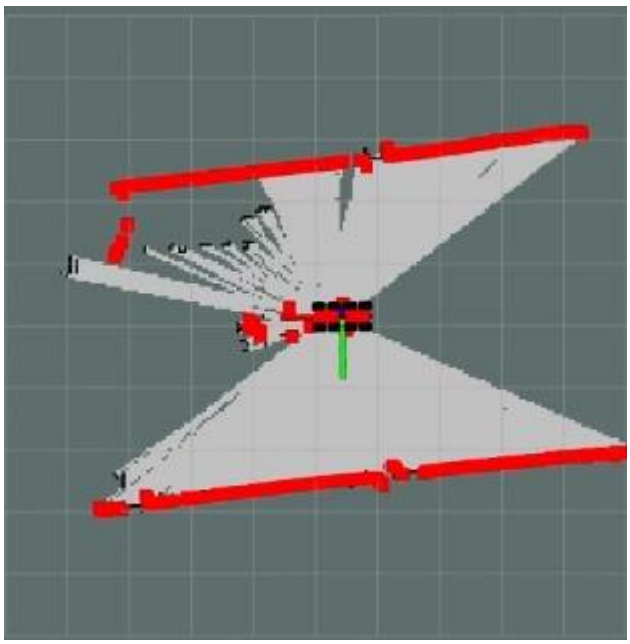


Figura 29. Mitad de trayecto recorrido en Test 1

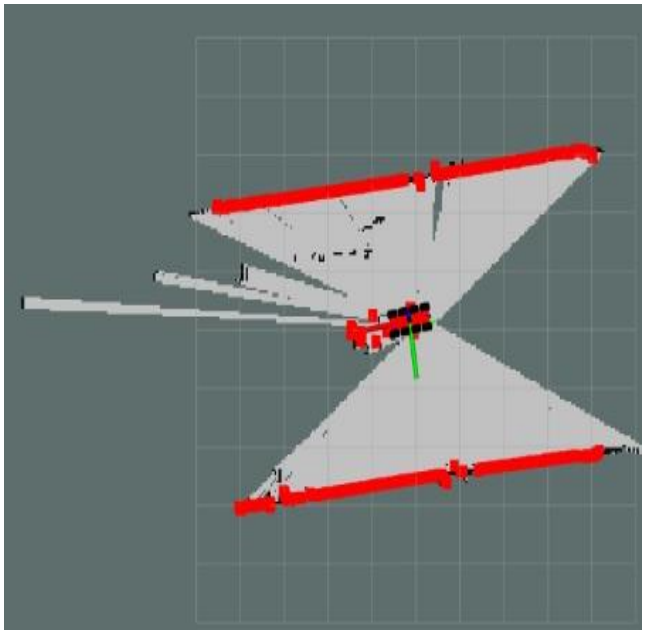


Figura 30. Llegada al punto final y mapa generado en Test 1

4.3.2.2 Test 2.

El test 2 se realizará en el mismo entorno del test 1, respetando las mismas dimensiones descritas en la figura 25.

La prueba se realizará de la siguiente manera:

- Se ubicará el vehículo en la misma posición inicial del test 1 (punto 1) (figura 26).
- El vehículo recorrerá una distancia de 9.35 metros en línea recta a baja velocidad hasta alcanzar el punto final (punto 2).
- Durante todo el recorrido del vehículo se tratará de mantener la velocidad constante.
- Entre el punto inicial y final del recorrido se pondrán 4 obstáculos, 2 de lado izquierdo y 2 de lado derecho.
- Cada obstáculo estará separado 1 metro de la pared, 4 metros uno del otro y 1.5 metros del camino que deberá recorrer el vehículo (figura 31).
- Desde el punto 1 hasta el punto 2 se ejecutará el algoritmo Hector-SLAM para simultáneamente mapear y localizar el vehículo durante todo el recorrido.

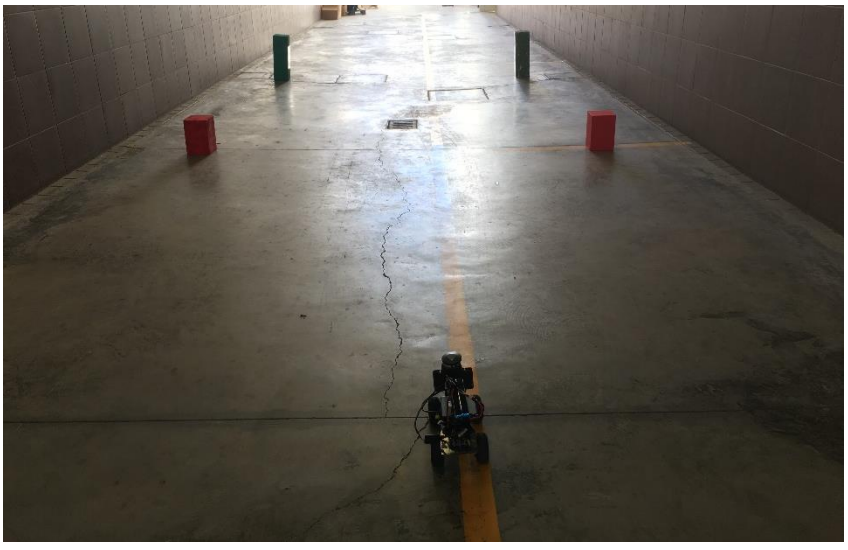


Figura 31. Layout del test 2 y 3

Resultados obtenidos:

- Al igual que en el test 1, al iniciar el test 2 los primeros escaneos que realiza el vehículo sobre el entorno que le rodea son correctos (figura 32 y 33).
- La posición real inicial del vehículo (punto 1) corresponde con la posición que envía el mapa 2D generado por el algoritmo Hector-SLAM (figura 33).
- En el mapa generado durante la posición inicial (aun estando en reposo), se alcanza a observar que el láser detecta una fracción de 3 de los 4 obstáculos que tiene a sus costados (figura 32).
- Durante el recorrido del punto 1 al punto 2, el mapa comienza a tomar forma, no se generan retrasos a la hora de imprimir sobre el mapa la información recibida del láser y no hay superposición de imágenes de los escaneos recibidos (figura 34).
- En el mapa final se observan algunas imperfecciones o detalles en cuanto a los objetos de los costados y las paredes, pero no es nada que afecte la lectura del mapa (figura 35).
- Durante el test 2, el mapa se ha generado de mejor manera que en el test 1, y esto es debido a que el entorno que rodea al vehículo tiene diversas variables (puntos de referencia) que se involucran a la hora de escanear el entorno.
- Debido a los objetos a los costados, los escaneos de información que envía el láser al algoritmo Hector-SLAM no son los mismos siempre, y por lo tanto el algoritmo intuye que el vehículo se debe estar moviendo, ya que el número de celdas libres y ocupadas durante el mapeo por cuadrícula cuadrada (*Occupancy Grid Mapping*) va cambiando constantemente, cosa que no sucedió en el test 1.

Figura 32. Ejecución del láser en punto inicial Test 2

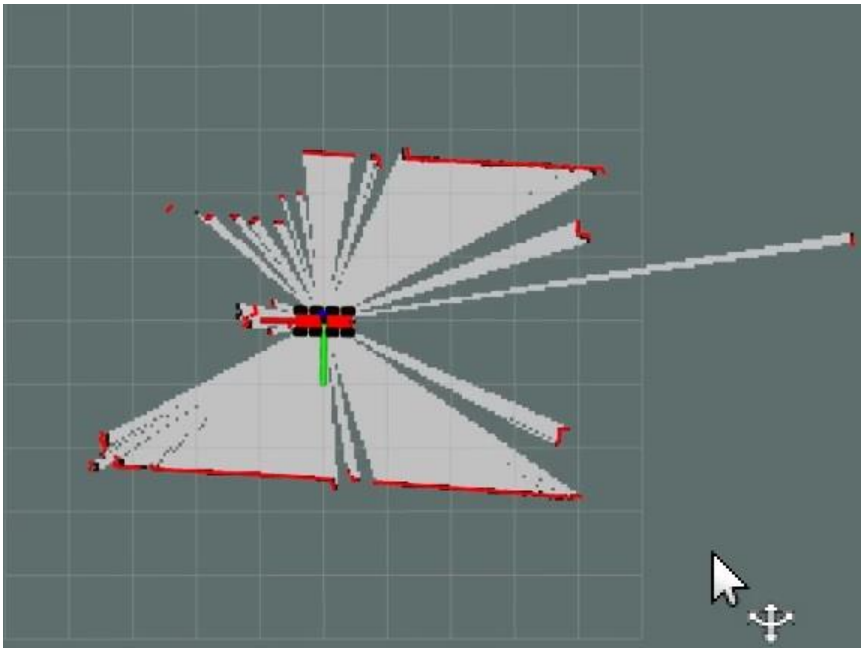


Figura 33. Ejecución del algoritmo Hector-SLAM en punto inicial (reposo) en Test

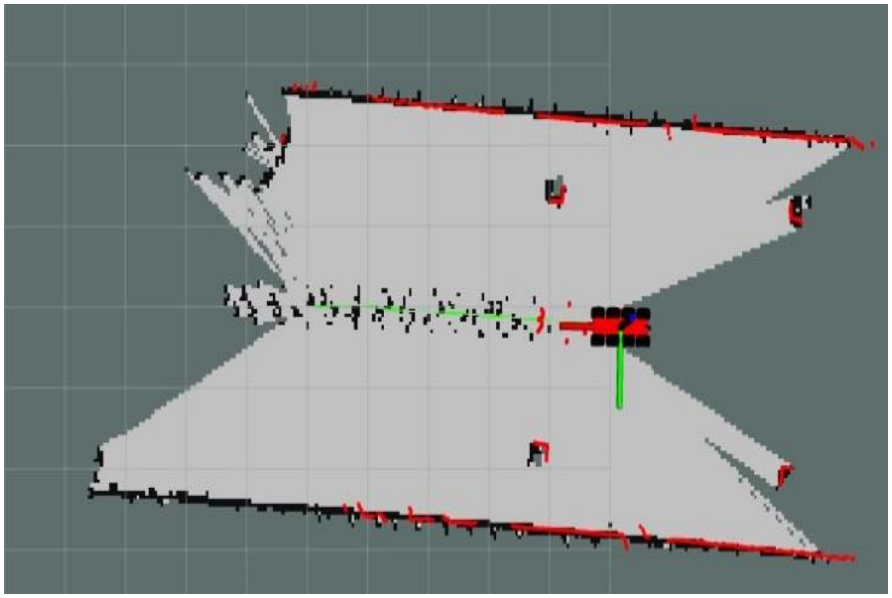


Figura 34. Mitad de trayecto recorrido en Test 2

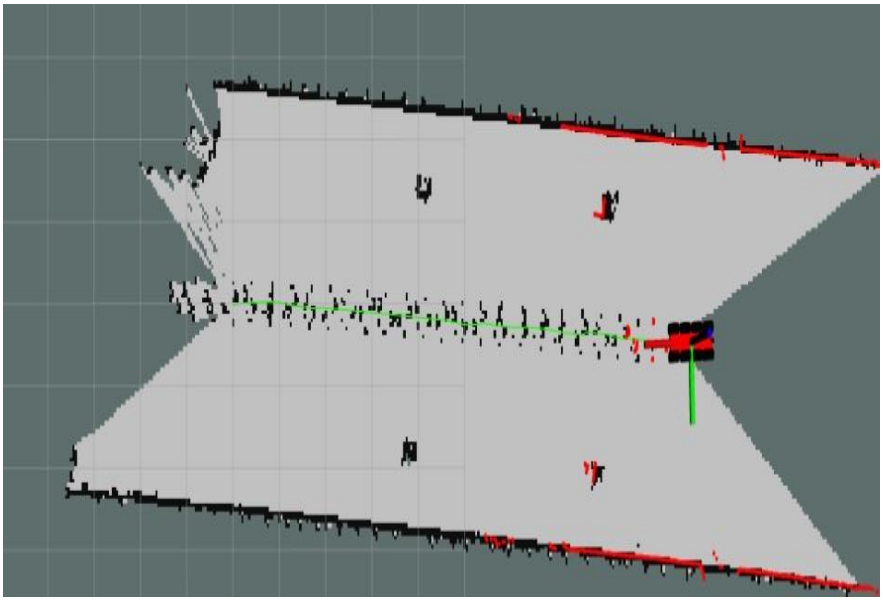


Figura 35. Llegada al punto final y mapa generado en Test 2

4.3.2.3 Test 3.

De igual manera, el test 3 se realizará en la misma ubicación del test 1, respetando las mismas dimensiones descritas en la figura 25.

La prueba se realizará de la siguiente manera:

- El test 3 se realizará bajo las mismas características y parámetros que el test 2.
- La única diferencia será que el vehículo se moverá a alta velocidad, esto con el fin de poder comparar los resultados del test 2 con el test 3. Comprobar si el mapa que se obtiene cambia mucho si la velocidad se aumenta durante todo el recorrido.

Resultados obtenidos:

- Durante la posición inicial de la prueba (punto 1) los resultados mostrados son los mismos que se obtuvieron en el test 2 (figura 36 y 37).
- Debido a que el vehículo avanzó del punto 1 al punto 2 a una velocidad mucho mayor que la del test 2, el mapa obtenido cambió significativamente con respecto al del test 2.
- El mapa del test 3 carece de toda la información con la que cuenta el mapa del test 2.
- El mapa generado (figura 39) tiene muchos espacios vacíos en comparación al mapa del test 2, estos espacios son medidas que no se pudieron determinar si eran celdas vacías o llenas, lo que originó que se volvieran "celdas inexploradas" por el algoritmo (figura 8, cap. 4.2).
- En el mapa de la figura 38 se observa cómo debido a problemas de lectura del láser, el algoritmo crea 6 objetos a los costados del vehículo y no 4.
- Los resultados del mapa a la hora de cambiar entre velocidad baja y alta no variarían tanto si se tuviera un algoritmo de SLAM más robusto y/o se tuviera un LIDAR con mayores prestaciones.

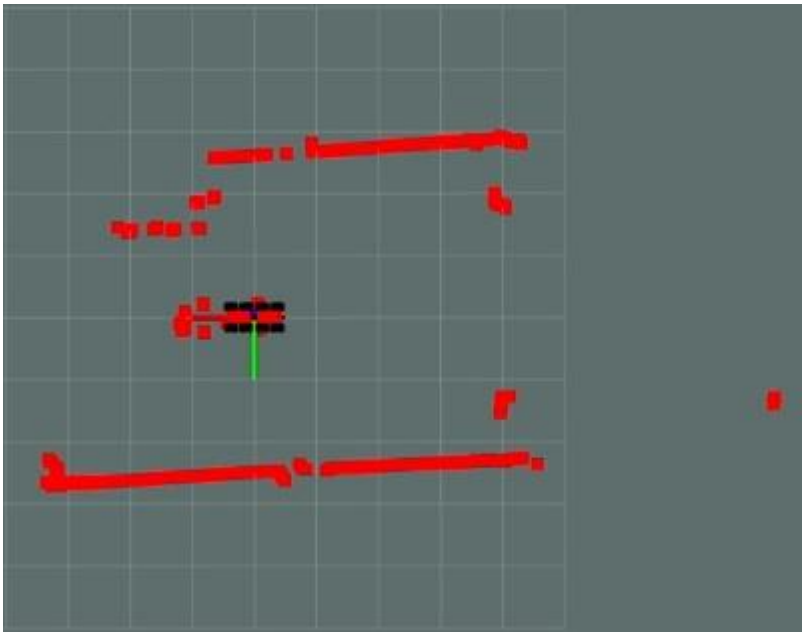


Figura 36. Ejecución del láser en punto inicial Test 3

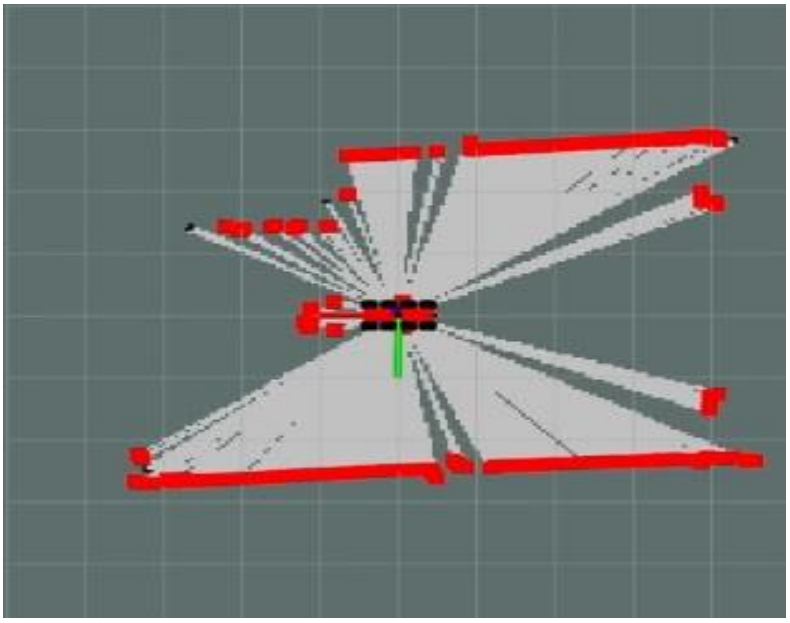


Figura 37. Ejecución del algoritmo Hector-SLAM en punto inicial (reposo) en Test

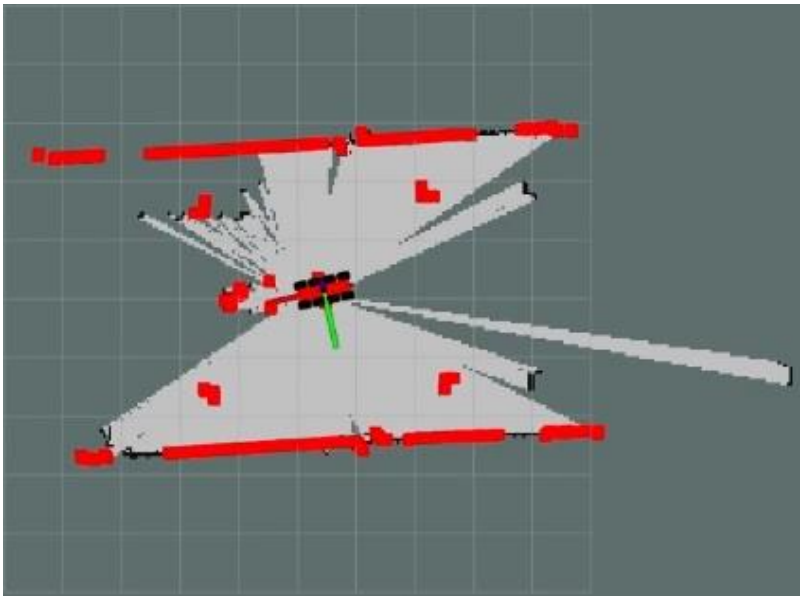


Figura 38. Mitad de trayecto recorrido en Test 3

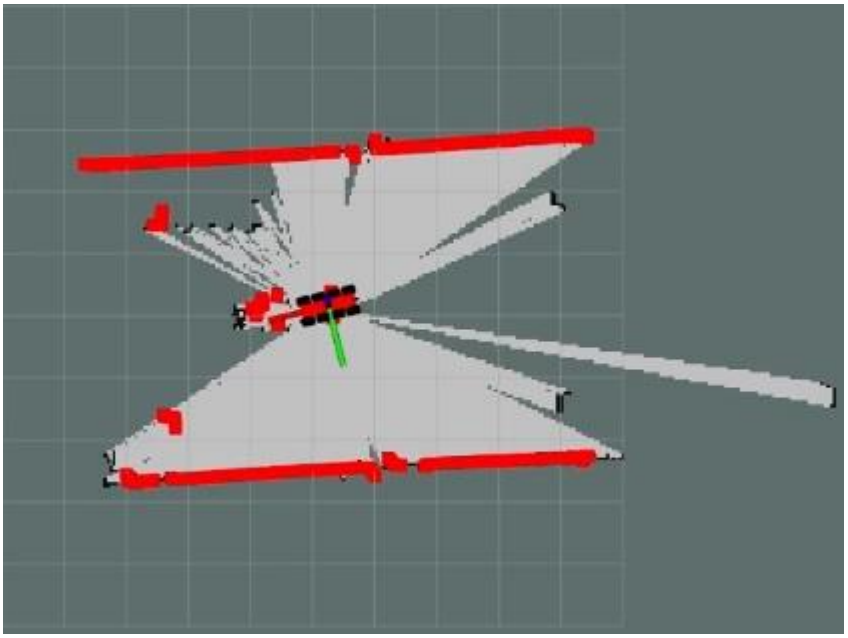


Figura 39. Llegada al punto final y mapa generado en Test 3

4.3.2.4 Test 4.

El test 4 se realizará en la misma ubicación de las pruebas anteriores, respetando las mismas dimensiones descritas en la figura 25. Habrá cambios en cuanto al recorrido que hará el vehículo y a la ubicación de los obstáculos.

La prueba se realizará de la siguiente manera:

- El punto inicial del vehículo (punto 1) en el test 4 será el mismo que en los test anteriores.
- La distancia que recorrerá el vehículo será mayor a la de los otros ensayos. La distancia total que recorrerá será de 12 metros, esto debido a que la realizará de manera diferente y el vehículo requiere un poco más de espacio para maniobrar.

- Se ocuparán 3 obstáculos y ya no estarán a los costados del vehículo, sino que estarán dentro del trayecto que hará el vehículo (figura 40). Los obstáculos tendrán una separación de 4 metros entre cada uno.
- El vehículo tendrá que seguir una trayectoria en zigzag para ir esquivando cada uno de los obstáculos. Debido a que el vehículo estará controlado por un mando a distancia y no por comandos, los giros en zigzag que dará el vehículo irán difiriendo unos con otros a lo largo de la prueba.
- La velocidad del vehículo durante la prueba tendrá que ser moderada y constante, una velocidad intermedia a la que se aplicó en el test 2 y 3.
- Desde el punto 1 hasta el punto 2 se ejecutará el algoritmo Hector-SLAM para simultáneamente mapear y localizar el vehículo durante todo el recorrido y posteriormente evaluar los resultados obtenidos.

Resultados obtenidos:

- Al ejecutar el algoritmo Hector-SLAM durante la posición inicial, los primeros escaneos que realiza el láser permiten ver en el mapa la ubicación correcta del objeto que tiene adelante el vehículo (figura 41 y 42).
- Mientras el vehículo comienza a moverse hacia adelante, el mapa se va generando con mayor detalle. Una vez que tiene de frente al primer obstáculo, se gira la dirección de las ruedas para rodearlo y pasar por un costado. Posteriormente se vuelve a girar la dirección en sentido contrario al anterior para que el vehículo se reincorpore al camino. Continuará con la trayectoria recta hasta encontrar de frente los siguientes objetos y repetirá las acciones anteriores para generar el recorrido en zigzag.
- Durante el recorrido del punto inicial al punto final, se observó que la generación del mapa tuvo ciertos detalles. Hubo momentos en los que el vehículo "saltó" un poco dentro del mapa de una posición a otra, esto para poder ubicarse correctamente según las lecturas que recibía del láser.
- Los "saltos" generados se dieron cuando el vehículo rodeó el objeto 2 y el objeto 3. En la figura 43 se puede observar cómo en el mapa no se muestra el recorrido del vehículo durante el trayecto del objeto 1 al objeto 2, solo se observa el giro que hizo el vehículo rodeando al objeto 2.
- En el mapa de la figura 44 se aprecia que los obstáculos se encuentran ligeramente movidos de su posición real, este detalle afecta considerablemente la lectura del mapa.
- Debido a los cambios de dirección ocasionados por los giros, los bordes de los objetos se ven de manera tenue y distorsionada, y el recorrido que se muestra en el mapa final no es el recorrido real que hizo el vehículo.
- Analizando los resultados obtenido de esta prueba se considera que el algoritmo Hector-SLAM no ha podido desempeñarse de manera suficiente, efectiva y precisa.

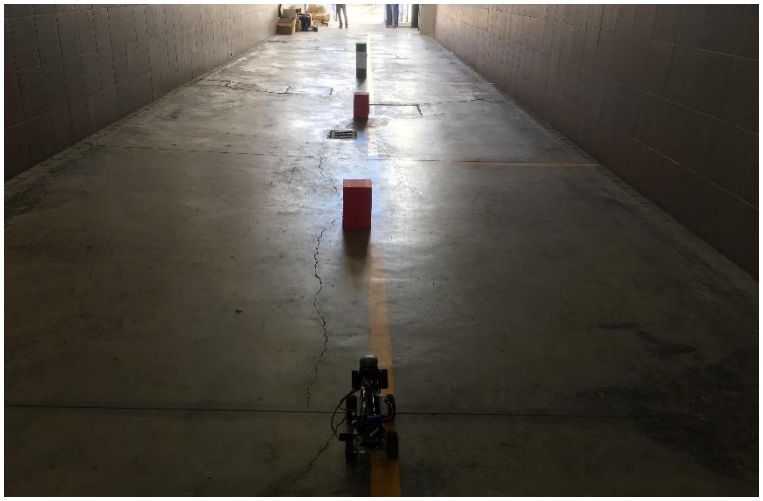


Figura 40. Layout del entorno en test 4

Figura 41. Ejecución del láser en punto inicial Test 4

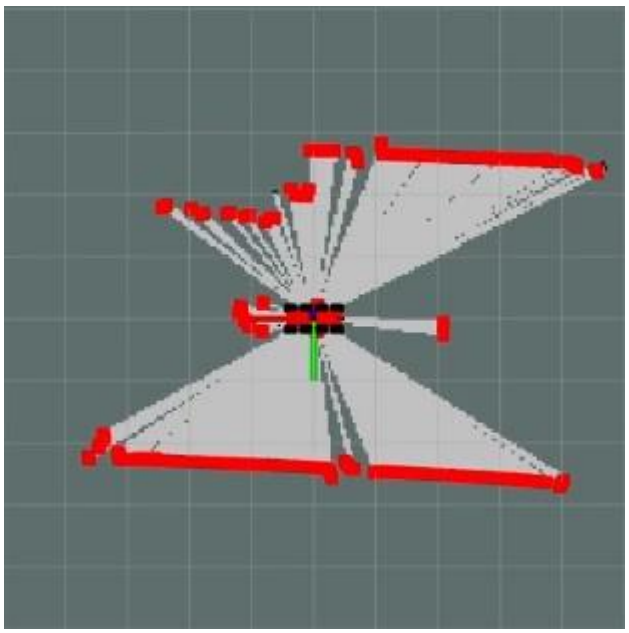


Figura 42. Ejecución del algoritmo Hector-SLAM en punto inicial (reposo) en Test

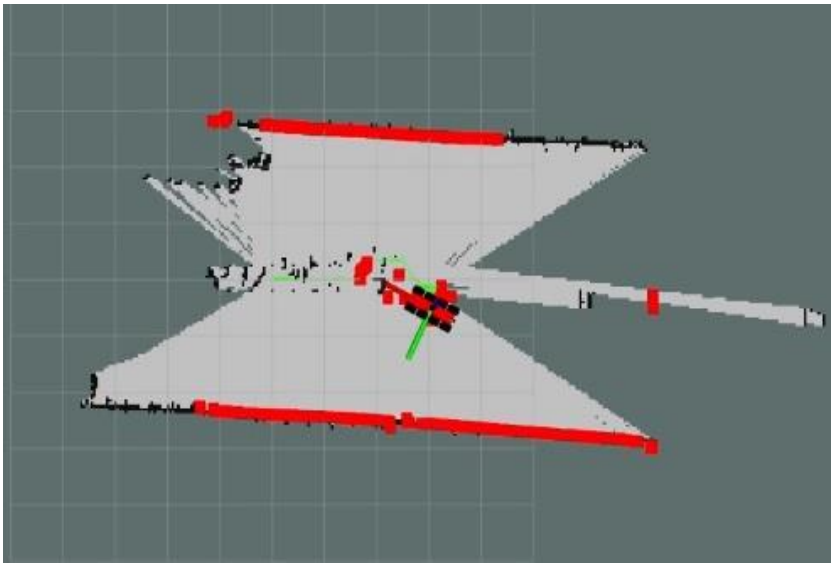


Figura 43. Mitad de trayecto recorrido en Test 4

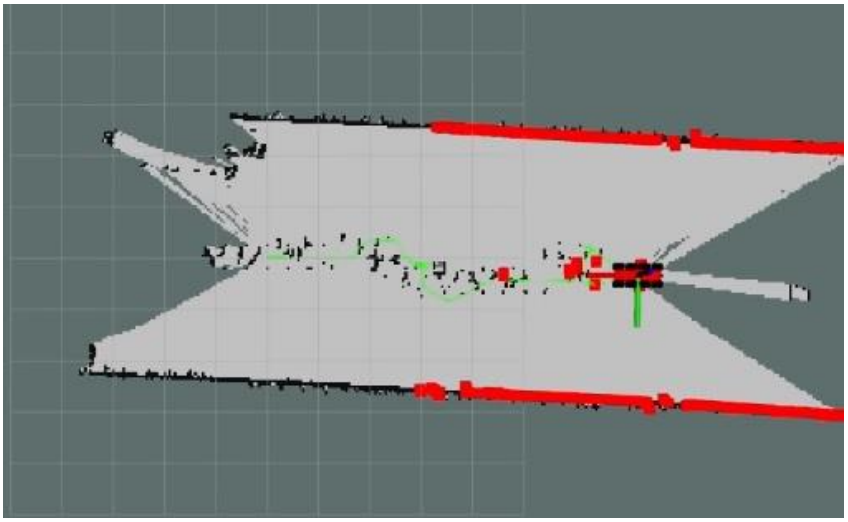


Figura 44. Llegada al punto final y mapa generado en Test 4

4.3.2.5 Test 5.

Este test se realizó en el interior del laboratorio de Robótica y Automática, esto con la finalidad de ver cómo se comporta el algoritmo Hector-SLAM en un ambiente más real en el que existen diversas variables a considerar durante la creación del mapa en 2D. Se definen como variables a todos los objetos, obstáculos, cambios de tamaños del entorno, distintas superficies de cada objeto u obstáculo, cambios en la iluminación del entorno, entre otras.

La prueba se realizará de la siguiente manera:

- El punto inicial y final del recorrido serán el mismo, es decir, el vehículo realizará la misma trayectoria tanto de ida como de vuelta por un camino determinado. Ya no será como en los ensayos anteriores donde solo se recorría un trayecto de ida.
- El recorrido lo realizará tratando de llevar una velocidad moderada de crucero.
- El recorrido tendrá una distancia total de 23.2 m (ida y vuelta).
- Durante la implementación de este ensayo, no se tomará en cuenta la distancia que existe entre cada objeto/obstáculo, ni la distancia que existe desde los objetos/obstáculos hacia

el vehículo. Se trata de analizar el comportamiento del algoritmo en una situación más real y no tan controlada como en los ensayos anteriores.
- Durante todo el recorrido se ejecutará el algoritmo Hector-SLAM para simultáneamente mapear y localizar el vehículo.

Resultados obtenidos:

- Al ejecutar el funcionamiento del RPLIDAR-A3 y posteriormente el algoritmo Hector-SLAM en la posición inicial, los primeros escaneos que realiza el láser ubican de manera correcta al vehículo dentro del mapa (figura 45 y 46).
- Al haber una mayor concentración de objetos/obstáculos alrededor del vehículo, los haces del láser detectan e imprimen en el mapa 2D más espacios ocupados, que libres.
- A pesar de las diversas superficies, formas y materiales que tienen los objetos/obstáculos del entorno, el RPLIDAR-A3 puede detectarlos con bastante precisión.
- Durante gran parte del recorrido el mapa se creó de manera continua (figura 47, 48 y 49), el vehículo no mostró "saltos de posición" dentro del mapa como en el caso del test 4.
- Cuando se estaba por finalizar el recorrido se presentó en varias ocasiones el problema de "cierre de bucles" (figura 50), ya que apareció un ligero *lag* que desvió un poco la ubicación del vehículo.
- A pesar de que el algoritmo Hector-SLAM no toma demasiada importancia para contrarrestar y corregir el problema de cierre de bucles, este lo ha sabido manejar muy bien y no se ha visto gran afectación al crear el mapa (figura 51).
- Con este último ensayo se ha podido demostrar la gran capacidad y presentaciones que tiene en interiores el algoritmo Hector-SLAM.
- Al observar con detalle el mapa 2D creado, es posible afirmar que la capacidad sensorial del vehículo puede detomar con bastante precisión medidas de la ubicación relativa entre los puntos de referencia y el propio vehículo.

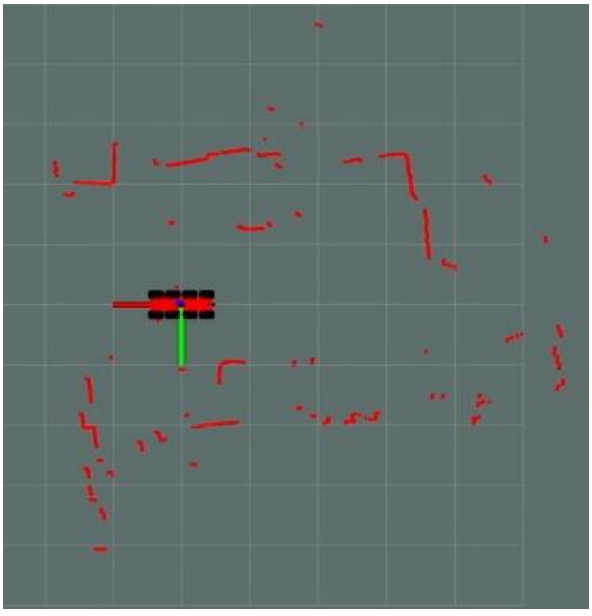


Figura 45. Ejecución del láser en punto inicial Test 5

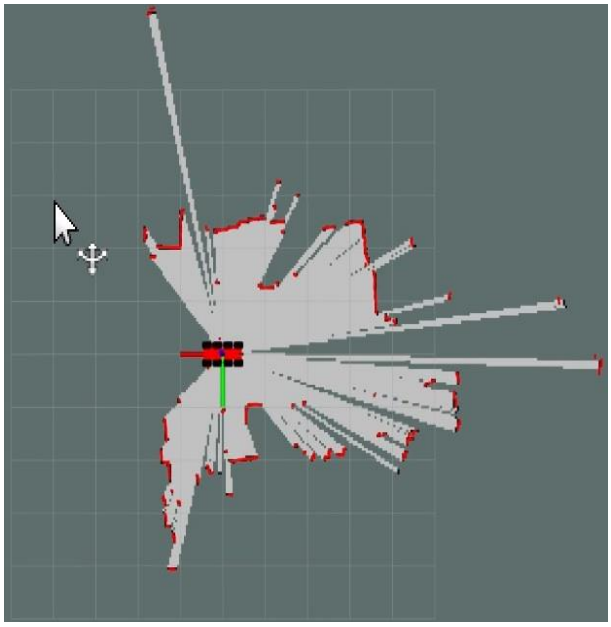


Figura 46. Ejecución del algoritmo Hector-SLAM en punto inicial (reposo) en Test

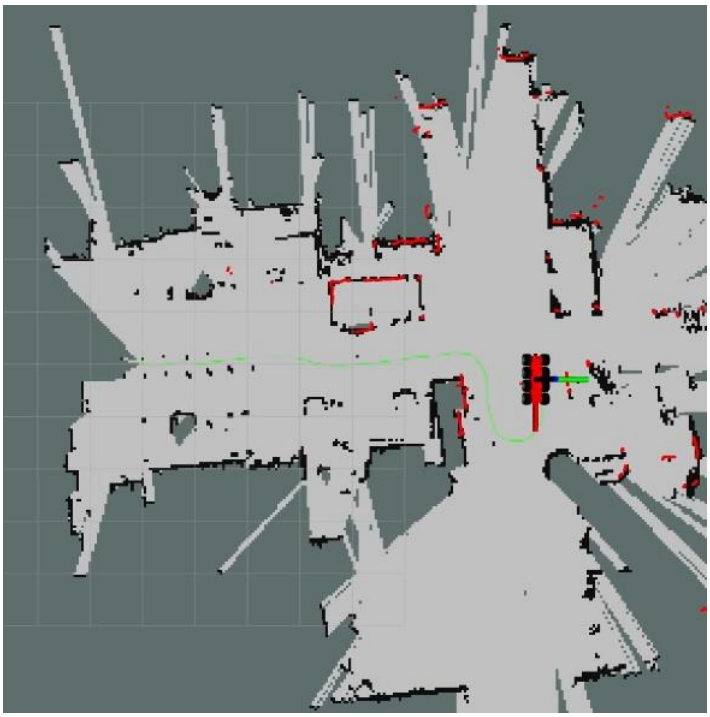


Figura 47. Primera vuelta durante el recorrido en Test 5

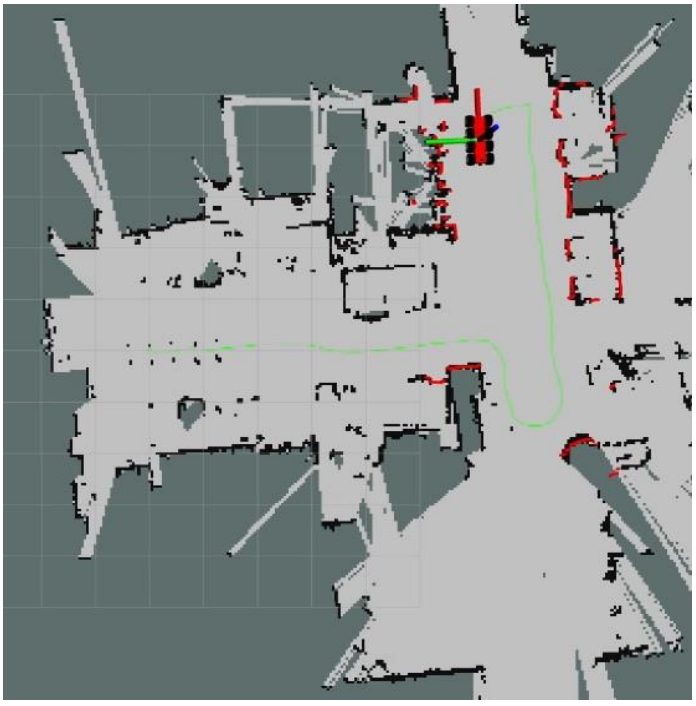


Figura 48. Segunda vuelta durante el recorrido en Test 5

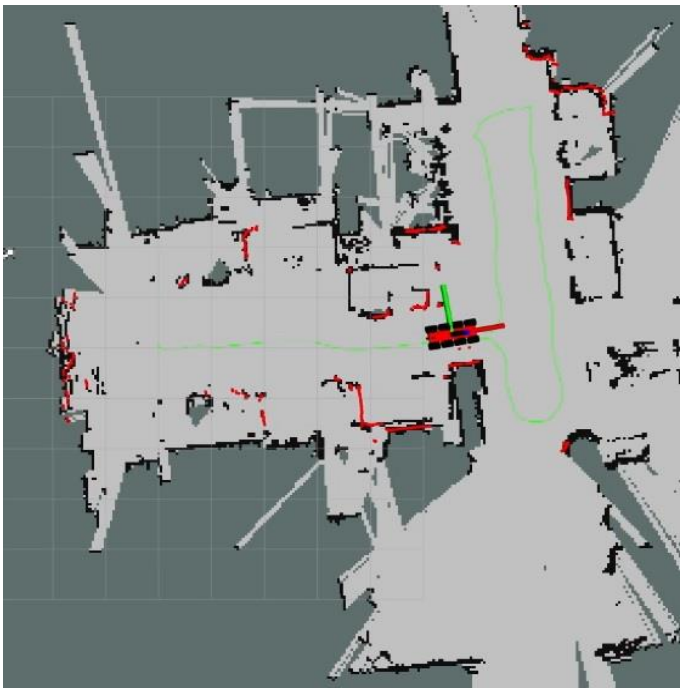


Figura 49. Tercera vuelta durante el recorrido en Test 5

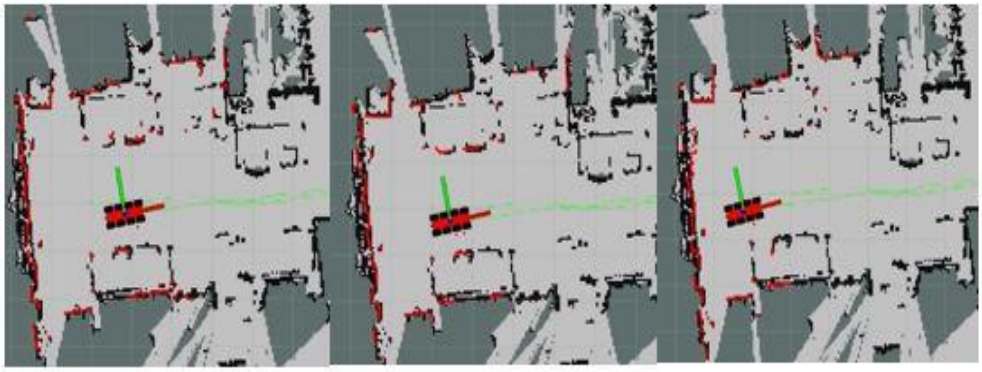


Figura 50. Saltos del vehículo debido al *loop closure* en Test 5

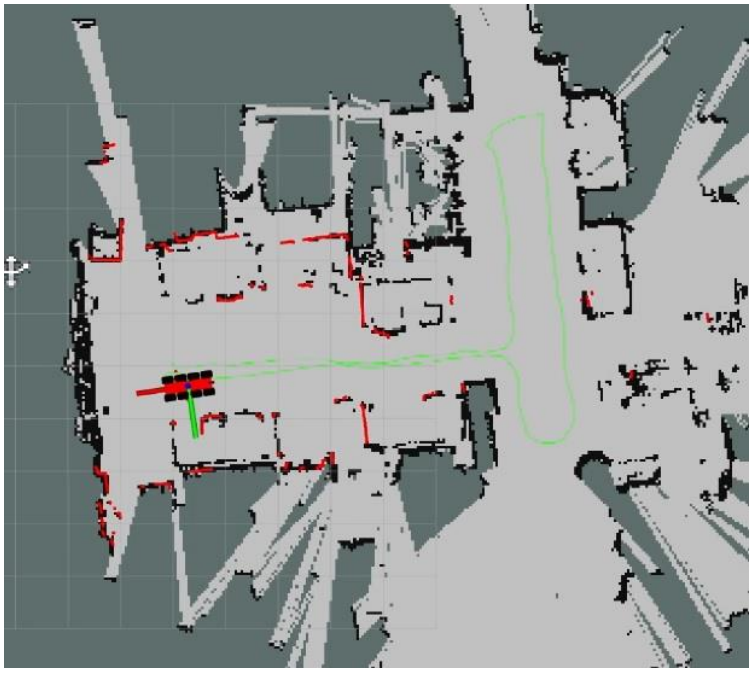


Figura 51. Llegada al punto final (posición y orientación correctas) y mapa generado en Test 5

5 Navegación

En este capítulo se explicarán algunos de los métodos de navegación autónoma más usados o conocidos para robots móviles. Algunos de estos métodos se implementarán en nuestro vehículo 8X8 para realizar pruebas en simulación, y posteriormente comparar los resultados obtenidos.

5.1 Descripción general.

Los robots móviles autónomos son requeridos en muchas aplicaciones como en el espacio, el transporte, la industria, la medicina, entre otros. La principal funcionalidad que debe tener un robot autónomo es poder navegar de manera libre en un ambiente estático o dinámico sin perjudicar lo que le rodea (obstáculos, personas, animales, etc). Para ello se debe lograr que la navegación sea fluida desde el punto de inicio hasta el punto final (meta), siguiendo un trayecto seguro y óptimo.

Para realizar una tarea, un robot debe comprender el entorno a partir de las mediciones de sus sensores, es decir, lograr capturar la información con un nivel correcto de extracción.

Un robot con navegación autónoma deberá tener conocimiento de:

- Cómo es el ambiente que le rodea (mapa) y cómo interpretarlo.
- Dónde está localizado (posición y orientación) dentro de ese ambiente.
- Qué tipo de control ejecutar para realiza una determinada ruta.

Un mapa permitirá mostrarle al robot una representación del ambiente donde está operando, dicho mapa deberá contener información suficiente para que el robot pueda completar las tareas que le involucren. Como se mencionó en el capítulo 4, el mapa que se utilizará en este proyecto está basado en la técnica de representación métrica de cuadrícula ocupada (*occupancy grid mapping*).

Un mapa métrico servirá como marco de referencia para que el robot pueda saber su ubicación en el ambiente que le rodea. Una vez que el robot tenga conocimiento del entorno que le rodea y la ubicación que tiene dentro de ese entorno, deberá determinar por medio de una planeación de ruta si le es posible realizar y alcanzar la ubicación objetivo que se le ha determinado. Tanto la posición actual del robot, como la ubicación objetivo, deberán estar dentro del mismo marco de referencia o coordenadas.

La planificación de ruta requiere definir un camino o trayectoria del robot en el entorno, de manera que el robot evite colisiones con obstáculos y los demás agentes involucrados para poder cumplir la tarea asignada. De manera general, la planificación de ruta se puede dividir en dos tipos, global y local, donde el primero encuentra el camino óptimo con un conocimiento previo del entorno y obstáculos estáticos, y el segundo recalcula el camino para evitar obstáculos dinámicos [40].

5.2 Planificación de ruta.

El término planificador de ruta (*motion planning*) indica el algoritmo que calcula el movimiento desde una configuración inicial ***Cs*** (*start configuration*) hasta una configuración objetivo o final ***Cg*** (*goal configuration*) [40].

Se conocen muchos ejemplos de algoritmos de planificación del movimiento. Algunos de estos algoritmos son deterministas (es decir, suponen que toda la información se conoce o se puede

calcular con precisión) y algunos son estocásticos (es decir, conocidos hasta cierto punto). Aun así, es posible clasificarlos en tres grupos principales: algoritmos de campo potencial, algoritmos basados en cuadrículas y algoritmos basados en muestreo [41].

Los algoritmos de campo potencial se inspiraron en el concepto de cargas eléctricas. Un caso simple es considerar un espacio bidimensional con configuraciones de inicio ***Cs*** y meta ***Cg*** con un obstáculo redondo situado en el interior del espacio. Para resolver la planificación del movimiento, uno puede pensar intuitivamente en una partícula con carga eléctrica negativa fijada en la configuración objetivo ***Cg***, mientras que en ***Cs*** uno puede imaginar una partícula con carga eléctrica positiva. Una partícula con carga positiva que representa al robot se encuentra inicialmente en ***Cs***. El robot será repelido por ***Cs*** y atraído por ***Cg***. Siguiendo la misma idea, podemos colocar otra partícula con carga positiva en el centro del obstáculo, por lo que ***Cs*** se mueve en la dirección del gradiente de ambos campos potenciales, evitando consecuentemente el obstáculo y siendo atraído por la configuración objetivo.

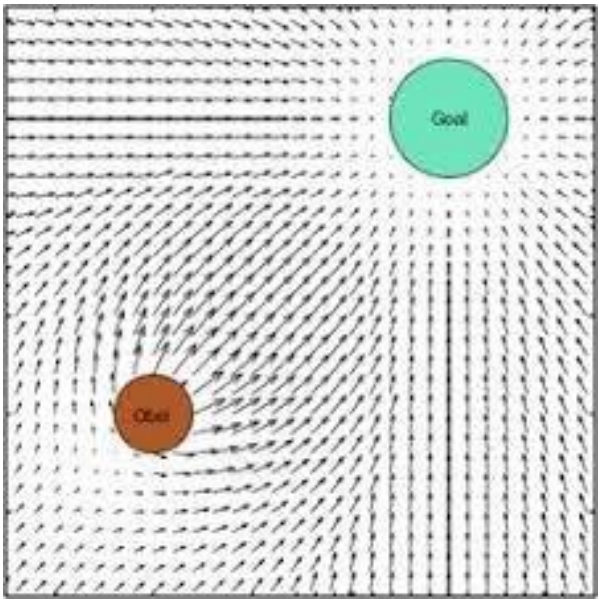


Figura 52. Representación en 2D de un campo potencial.

En la figura 52 se pueden observar de manera representativa los campos potenciales artificiales que permitirán regular el movimiento de un robot en un espacio determinado. El círculo de color café representa el obstáculo, el cual tiene un campo potencial de repulsión, mientras que el circulo de color vede representa la meta o destino final, la cual tiene un campo potencial de atracción.

Aunque los algoritmos de campo potencial son sencillos, ya que no se requiere cálculo previo (aparte de fijar la partícula cargada), su principal vulnerabilidad es la aparición de configuraciones indeseables donde la fuerza resultante se desvanece. En tales configuraciones, el planificador de movimiento puede no ser definitivo, mientras que en otros casos, el robot puede detenerse por completo (mínimos locales). Esto puede ocurrir, por ejemplo, cuando existen múltiples obstáculos que rodean al robot o también debido a la forma que tienen estos (figura 53). Para evitar este problema, se debe elegir correctamente la función de navegación, también llamada función potencial [42].

Los algoritmos basados en cuadrículas cuantifican el espacio del objeto en un número finito de celdas y luego realizan las operaciones necesarias en el espacio cuantificado, es decir, primero toman muestras del espacio para posteriormente crear un espacio ***C*** discreto. La principal ventaja de este algoritmo es su rápido tiempo de procesamiento el cual depende del número de celdas en cada dimensión en el espacio cuantificado.

Un algoritmo basado en cuadrículas muy conocido y eficaz es el A*, el cual, a partir de un nodo inicial específico de un gráfico, su objetivo es encontrar una ruta al nodo objetivo dado que tenga el menor costo (menor distancia recorrida, menor tiempo, etc.). Esta ruta la hace manteniendo un árbol de caminos que se originan en el nodo de inicio y estos se extienden un borde a la vez hasta que se satisface su criterio de terminación.

En cada iteración A * necesita determinar cuál de sus caminos debe extender basándose en el costo de la ruta y una estimación del costo requerido para extender la ruta hasta la meta. La última etapa que implementa A* es construir la ruta desde ***Cg*** hasta ***Cs*** únicamente rastreando las celdas principales. Algo a tomar en cuenta es que si se desea que la trayectoria encontrada resulte en un movimiento suave para el vehículo, se deberá aumentar la densidad de la cuadrícula, al hacer esto el volumen de datos aumentará y por lo tanto el costo computacional también [40].

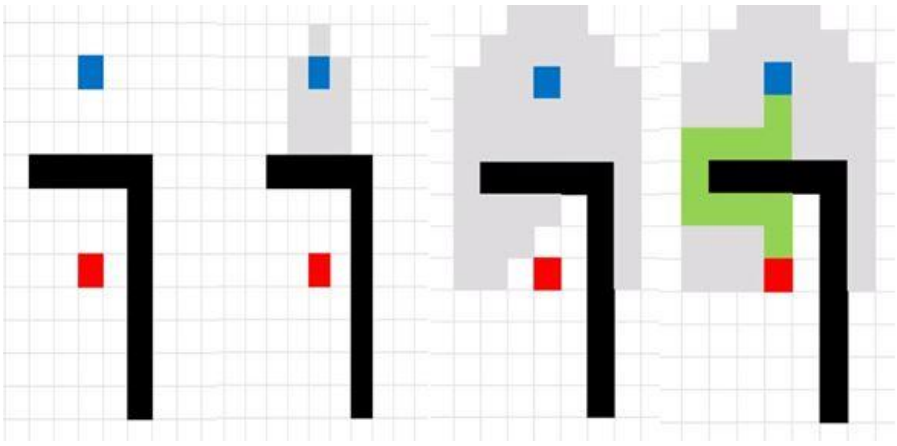


Figura 53. Implementación de algoritmo A*.

En la figura 53 la cuadrícula en rojo representa el punto final, y la cuadrícula en azul el punto inicial. Los caminos comienzan a extenderse, rodeando parte del obstáculo, hasta encontrar el punto final. Posteriormente trazará la ruta que haya tenido el menor costo requerido.

Los algoritmos basados en muestreo tratan de construir una ruta entre ***Cs*** y ***Cg*** pasando por varias configuraciones intermedias, generalmente muestreadas al azar [43]. En lugar de utilizar una representación explícita del entorno, se basan en un módulo de verificación de colisiones, que proporciona información sobre la viabilidad de trayectorias candidatas para posteriormente conectar un conjunto de puntos muestreados dentro del espacio libre de obstáculos y construir un gráfico (llamado hoja de ruta) de trayectorias factibles. Estos algoritmos son muy eficaces para la planificación de movimientos en espacios de gran dimensión [44] [45].

El algoritmo Dijkstra [46], es un buen ejemplo de algoritmos basados en muestreo. Todo el espacio lo transforma en una red en la que los nodos de la red representan puntos en el espacio de trabajo y los bordes (los arcos) representan el movimiento de un nodo al siguiente. Por lo general, cada borde está asociado con un valor no negativo que representa el costo de pasar de un nodo a otro. La ruta resultará en una colección de movimientos desde el nodo inicial (nodo A, figura 54) hasta el objetivo (nodo G, figura 54), y el costo de la ruta será la suma de todos los costos a lo largo de la ruta.

A pesar de su gran desempeño, se considera más eficiente el algoritmo A* que el algoritmo Dijkstra, ya que el A* requiere menos tiempo computacional debido a algunas heurísticas que se implementa.

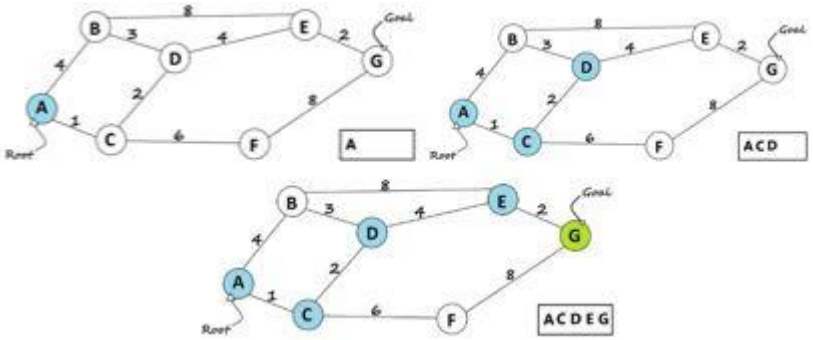


Figura 54. Implementación de algoritmo Dijkstra.

Como se mencionó al final del capítulo 5.1, la planificación de ruta puede ser global o local. El planificador de ruta global utiliza información a priori del entorno para crear el mejor camino posible, es decir, requiere un mapa del entorno para poder calcular la mejor ruta [47]. Dentro de los algoritmos que destacan como planificadores de ruta global está Dijkstra, A*, descomposición de celdas, campos potenciales, entre otros. Debido a como divide el mapa el planificador global, el resultado obtenido de ruta planificada puede no ser uniforme y en algunos puntos no cumplir con la geometría y la cinemática del vehículo, por ello es necesario implementar también un planificador de ruta local.

El planificador de ruta local crea nuevos puntos de ruta teniendo en cuenta los obstáculos dinámicos y las limitaciones del vehículo [47]. El mapa se irá generando a un ritmo específico según el vehículo se vaya moviendo, lo que hará que el mapa que vaya creando se reduzca a los alrededores del vehículo y se vaya actualizando conforme el vehículo se mueva.

Para el desarrollo de la planificación local no es posible hacer uso del mapa completo debido a que los sensores no pueden actualizar el mapa en todas las regiones existentes y además porque se aumentaría el costo computacional. En conclusión, la planificación local genera estrategias de evitación de obstáculos dinámicos e intenta hacer coincidir la trayectoria tanto como le sea posible de acuerdo a los puntos proporcionados por el planificador global. Algunos métodos que destacan para su implementación son las líneas de Clothoids, las líneas de Bezier, arcos y segmentos, entre otros [47].

Para el desarrollo e implementación de este proyecto se utilizaron y compararon dos algoritmos muy conocidos, el algoritmo Bug2 y el algoritmo de localización adaptativa de Monte Carlo (AMCL). Ambos ofrecen diversas prestaciones, así como también tiene diversas deficiencias.

5.3 Algoritmo Bug2.

Este algoritmo pertenece al esquema de algoritmos de planificación de movimiento donde el espacio de configuración debe ser parcialmente conocido. La planificación de ruta en un espacio totalmente conocido seguirá un sistema de coordenadas globales, los movimientos podrán ser ejecutados sin que el robot utilice ninguna información proveniente de los sensores, esto es debido a que se intuye conocer al 100% el entorno y este no tendrá ningún cambio [40].

Por lo general, la información disponible del robot son los datos que recopila de sus sensores durante el movimiento. Para la mayoría de los problemas de planificación del movimiento de la vida real, no hay suficiente tiempo o información para los cálculos previos. El problema surge cuando se desconoce la ubicación y la forma de los obstáculos.

Los algoritmos tipo Bug resuelven este problema, ya que utilizan la información que reciben los sensores para percibir obstáculos y/o la distancia hacia el punto final (meta). Los 3 tipos de algoritmos Bug más conocidos son: Bug0, Bug1 y Bug2.

Para el algoritmo Bug0 [48], el movimiento se establece como una línea recta hacia ***Cg***. Si el robot encuentra un obstáculo, el movimiento continúa siguiendo los límites del obstáculo (rodeándolo), mientras que la dirección en la que gira (izquierda / derecha) es aleatoria o predefinida. El robot avanza hasta que el segmento de línea entre la configuración actual y la configuración objetivo deja de cruzarse con el obstáculo. Luego, el movimiento continúa hacia ***Cg*** en línea recta. El problema principal con este algoritmo es que la convergencia no es segura, ya que la elección de la dirección del movimiento cuando el robot se encuentra con el obstáculo no se elige sabiamente.

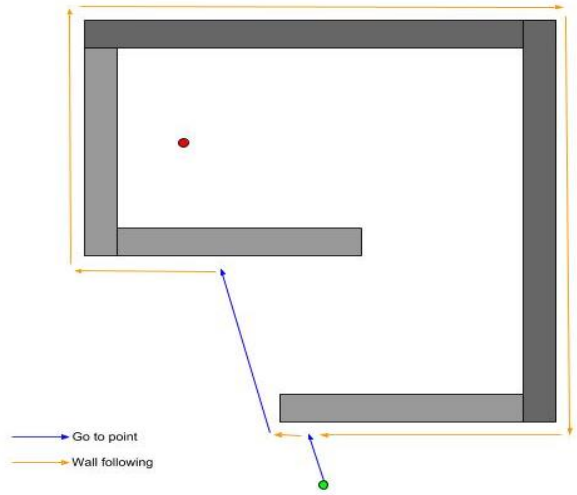


Figura 55. Implementación de algoritmo Bug0.

El algoritmo Bug1 [40] es similar al antes mencionado. El algoritmo rodea el obstáculo buscando la posición de distancia mínima a ***Cg***. Posteriormente, después de haber rodeado todo el obstáculo, el robot vuelve a la configuración más cercana a ***Cg*** y avanza hacia ella en línea recta.

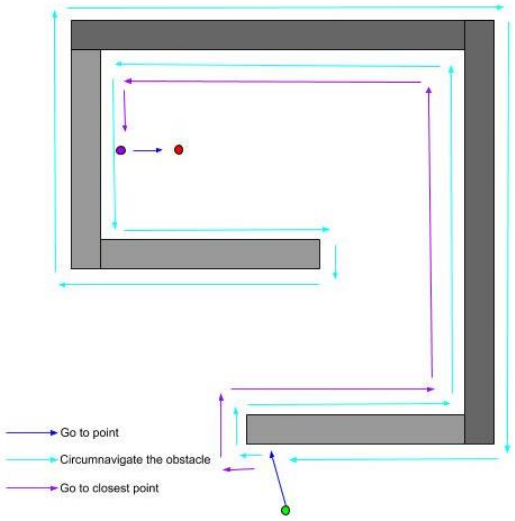


Figura 56. Implementación de algoritmo Bug1.

En cambio, el algoritmo Bug2 [40], difiere a BUG1 debido a las decisiones que toma cuando encuentra un obstáculo. Cuando el robot ejecuta el algoritmo, marca una línea ***L*** recta hacia ***Cg***. El algoritmo rodea el obstáculo hasta que vuelve a encontrar ***L*** y así proceder hacia ***Cg*** en línea recta. Se ha optado por implementar en el vehículo 8X8 el algoritmo Bug2 debido a que de los 3 algoritmos Bug existentes, el Bug2 es el que presenta mayor eficacia para llegar al punto objetivo ***Cg*** y menores deficiencias mientras se implementa.

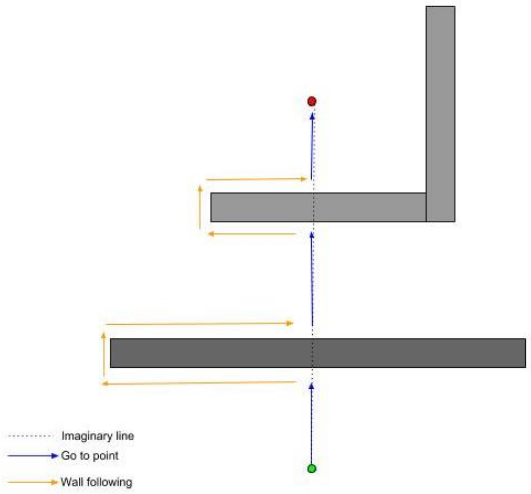


Figura 57. Implementación de algoritmo Bug2.

El desarrollo del algoritmo Bug2 que se implementó en las simulaciones del vehículo 8X8 fue programado en lenguaje Python. Este algoritmo consta de 3 partes: primero un programa que le permita al robot ir desde un punto inicial a un punto final utilizando la información de la odometría para la ubicación del vehículo. Segundo, un programa que le permita detectar los objetos que tiene a su alrededor, acercarse a ellos y posteriormente rodearlos. Y tercero, un programa que le permita trazar una línea recta imaginaria desde el punto inicial hasta el punto final, en conjunto con los 2 programas anteriores.

El programa para la planificación de la ruta (con nombre *go to goal*) consiste en los siguientes pasos:

1. El robot se ubica en un punto inicial dentro del espacio de coordenadas planas (x,y).

2. Se envía un comando (x,y) el cual se definirá como su punto final o meta (*goal)*.
3. Al recibir el comando, el vehículo comenzará a girar sobre su eje angular z (*yaw*) buscando la dirección en la que se encuentra el objetivo (*goal*).
4. Una vez que haya localizado el objetivo, deberá moverse en línea recta hasta alcanzar el objetivo asignado.
5. Como es de esperarse, el vehículo irá perdiendo la dirección conforme comience a moverse hacia el objetivo. Por ello se implementó un cálculo que corrige el error angular z (*yaw error*) y no permitirá que el vehículo pierda la dirección mientras se dirige hacia el objetivo.
6. Una vez que el vehículo llega al objetivo se mantendrá en estado de reposo hasta no recibir otro comando.

El diagrama 1 muestra el proceso general del algoritmo *go to goal*.

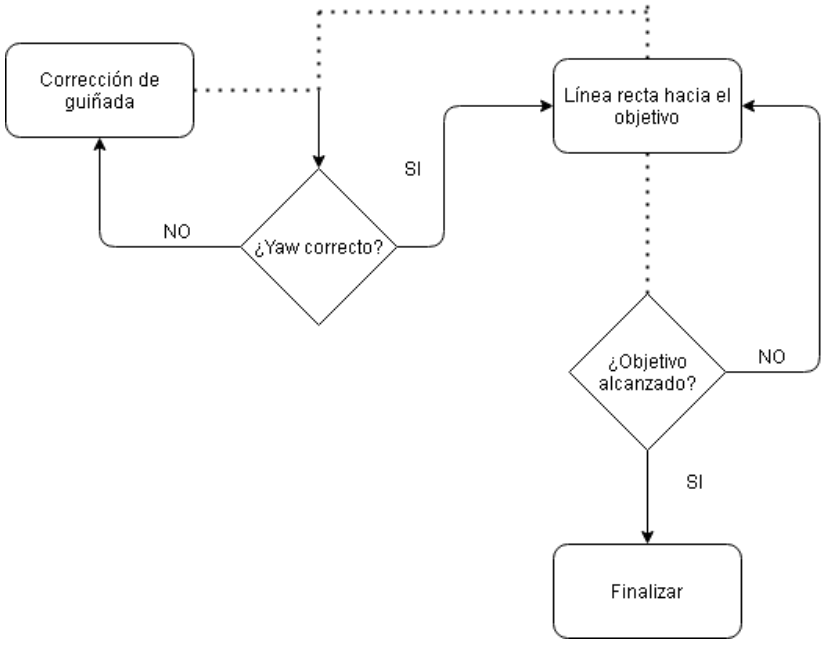


Diagrama 1. Desarrollo del algoritmo *go to goal*.

El segundo programa que se implementó fue el de detección y seguimiento (rodear) de objetos (con nombre *boundary following*), el cual consiste en las siguientes instrucciones:

1. El vehículo se moverá sobre su eje angular z (*yaw*) observando el entorno buscando la pared u objeto más cercano.
2. Una vez que localice el objeto irá en línea recta hacia él.
3. Para evitar que colisione con el objeto detectado, se le ha dado una distancia de seguridad entre el objeto y el vehículo que deberá respetar.
4. Una vez que alcance esa distancia de seguridad, el vehículo girará hacia la izquierda y comenzará a rodear dicho objeto o pared en un bucle.

A continuación, se presenta el diagrama de flujo con el proceso de ejecución del algoritmo *boundary following*:

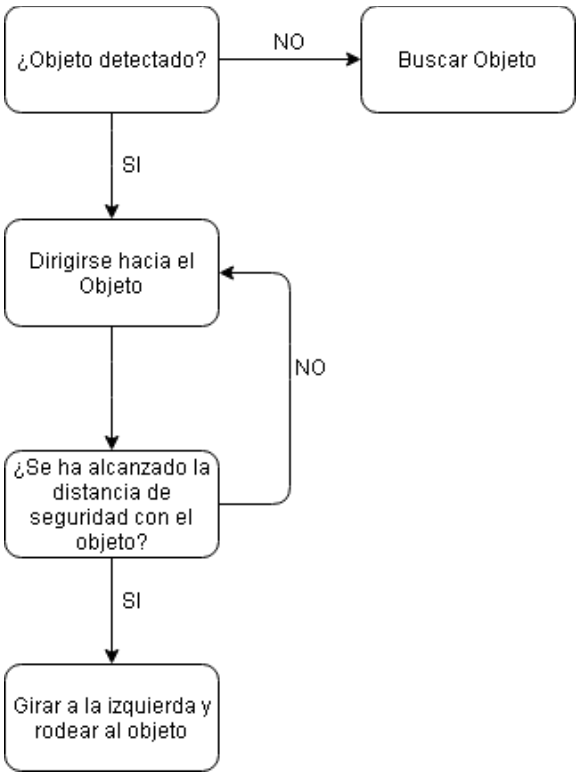


Diagrama 2. Desarrollo del algoritmo *boundary following*.

Por último, el algoritmo Bug2 calculará una línea recta desde el punto inicial hasta el final e implementará los 2 algoritmos anteriores para poder llegar a ese punto.

La ejecución del algoritmo es la siguiente:

1. El programa inicia ejecutando el algoritmo *go to goal* para encontrar el punto objetivo (*goal*).
2. Una vez encontrado el objetivo, el algoritmo Bug2 calculará y trazará una línea recta imaginaria que conecta el punto inicial con el punto final. Esto ayudará a que el robot tenga conocimiento de la ubicación final en todo momento.
3. Si el vehículo detecta que no tiene ningún objeto que obstaculice su movimiento hacia el punto final, entonces entrará otra vez el programa *go to goal*, lo cual hará que el vehículo se mueva en dirección recta hacia el objetivo final.
4. Cuando el vehículo detecte un obstáculo que le impida continuar moviéndose en línea recta hacia el objetivo entonces entrará en ejecución el algoritmo *boundary following*, el cual hará que el vehículo se acerque a ese obstáculo hasta la distancia de seguridad permitida.
5. Una vez que llegue a ese obstáculo, girará hacia la izquierda y comenzará a rodearlo siempre respetando que no se reduzca esa distancia de seguridad permitida entre el vehículo y el objeto.
6. El vehículo siempre tiene presente la ubicación final a la que debe dirigirse aún mientras va rodeando el objeto. Por ello, una vez que alcanza la línea recta trazada al inicio entrará en ejecución el algoritmo *go to goal* para dirigir al vehículo hacia el destino final.
7. Cuando el vehículo llegue al destino final quedará en estado de reposo hasta no recibir otro nuevo comando.

Para el cálculo de la línea recta entre el punto inicial y final se ha utilizado la ecuación definida por 2 puntos [49]. Dicha ecuación especifica que si la línea pasa entre dos puntos *P1(x1,y1)* y *P2(x2,y2)*, en la posición *(x0,y0)*, la distancia de la línea será:

$$distancia\big(P1, P2, (x0, y0)\big) = \frac{|(y2 - y1)x0 - (x2 - x1)y0 + x2y1 - y2x1|}{\sqrt{(y2 - y1)^2 + (x2 - x1)^2}}$$

Donde *P1* es la posición inicial, *P2* es la posición final (*goal*) y *(x0,y0)* es la posición actual en cada movimiento. El denominador de esta ecuación representará la distancia entre el punto inicial y el final.

A continuación, se muestra el desarrollo y los resultados obtenidos durante la simulación hecha en Gazebo y Rviz del algoritmo Bug2 con el vehículo 8X8.

5.3.1 Desarrollo e implementación.

Al vehículo se le dio como punto inicial las coordenadas x=0 y=10 (figura 58), como punto final las coordenadas x=0 y=0 (figura 58), un margen de error en la posición final de 0.2 m y que la parte frontal del vehículo apuntara en sentido opuesto al destino final. Como obstáculos y/o objetos se han puesto 4 a los costados del vehículo y 1 en la parte central del entorno (figura 59). Al ejecutar el algoritmo el vehículo comienza a girar en el eje angular z (yaw) en busca del objetivo asignado. Una vez que encuentra el objetivo, traza una línea recta imaginaria hacia el punto final y comienza a avanzar en línea recta hacia dicho punto final. Durante su recorrido detecta un objeto frente a él, se acerca a dicho objeto hasta la distancia de seguridad permitida para posteriormente rodearlo hasta volver a encontrarse con la línea recta imaginaria trazada en

un inicio (figuras 60, 61 y 62). Seguirá esta línea recta hasta alcanzar el objetivo final (*goal*) y se quedará en reposo (figura 63).

Durante todo el recorrido se utilizó el algoritmo Hector-SLAM para mapear y localizar el vehículo, así como también se utilizó el *plugin* del láser RPLIDAR-A3 para la detección de objetos y odometría láser.

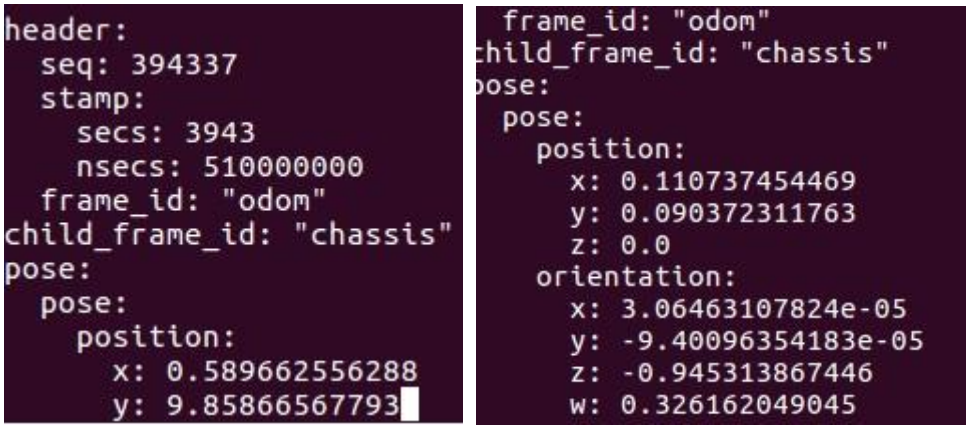


Figura 58. Posición inicial (izquierda), posición final (derecha)

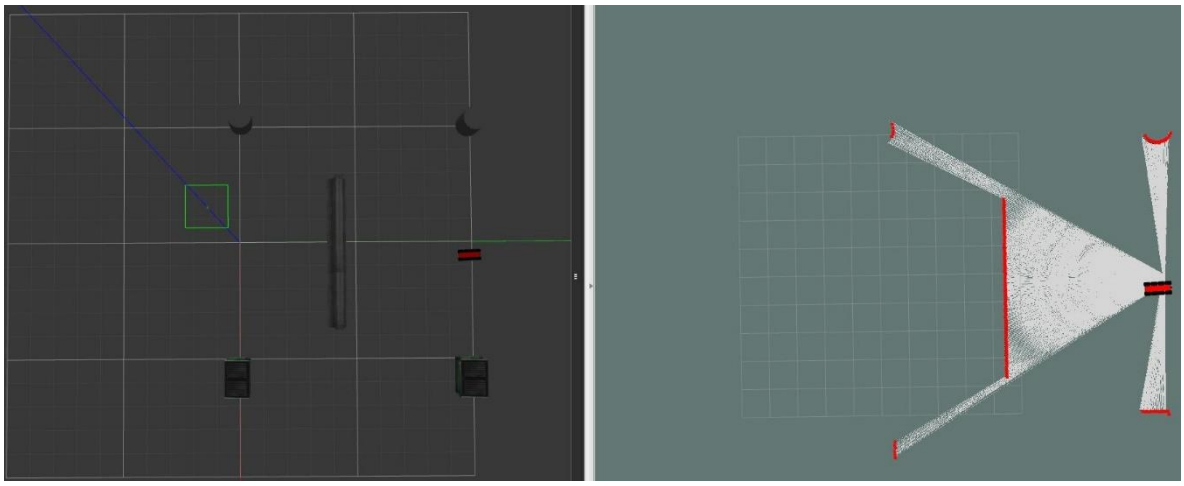


Figura 59. Entorno simulado y mapeo en posición inicial.

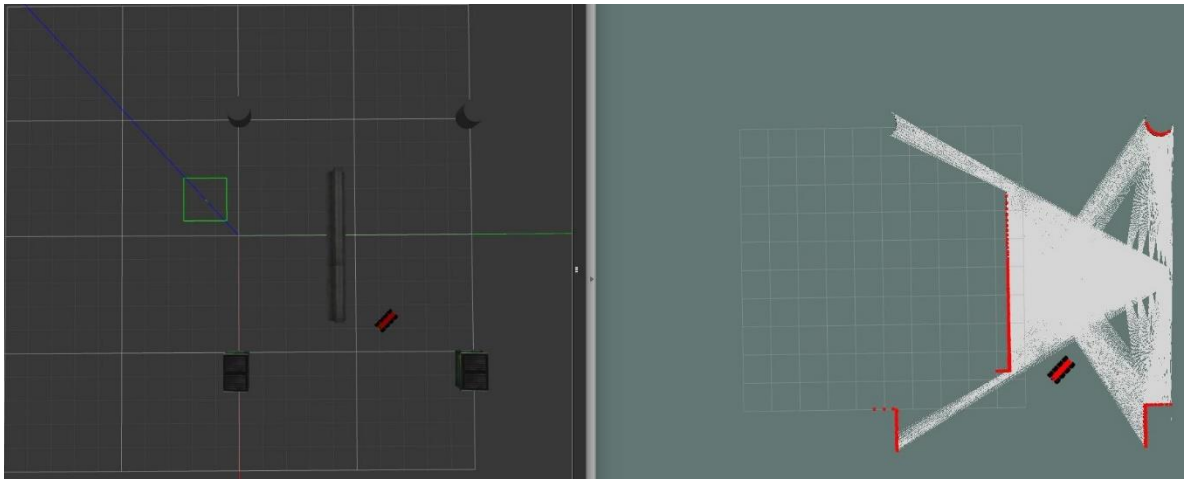


Figura 60. Detección y seguimiento de la parte frontal del objeto.

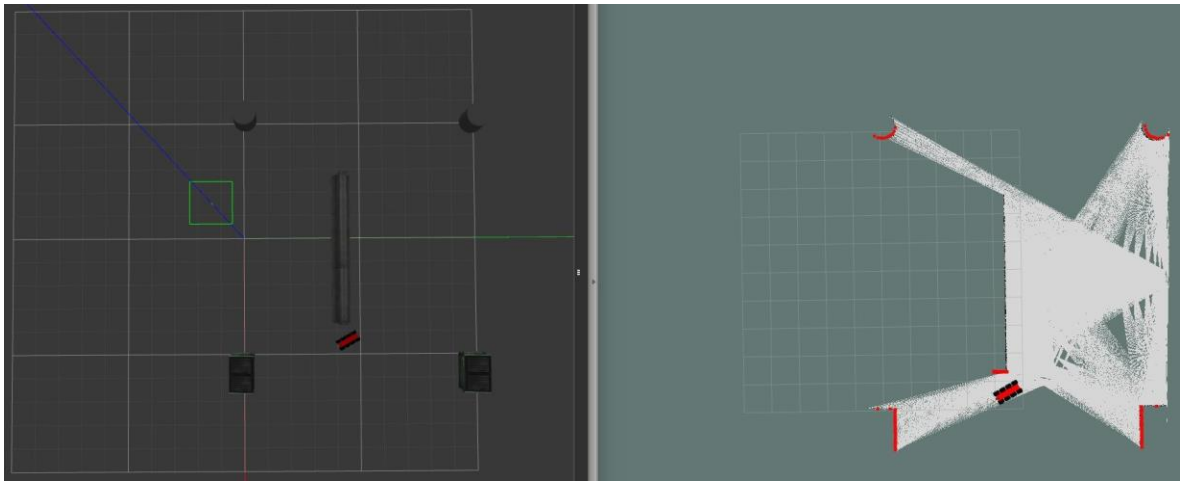


Figura 61. Detección y seguimiento de la parte lateral del objeto.

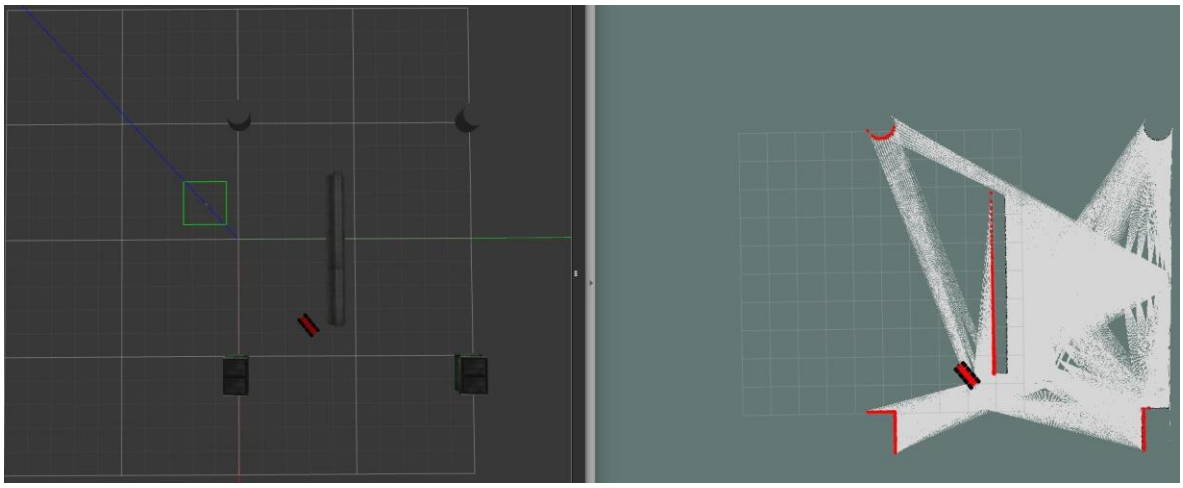


Figura 62. Seguimiento del objeto hasta alcanzar la línea recta imaginaria.

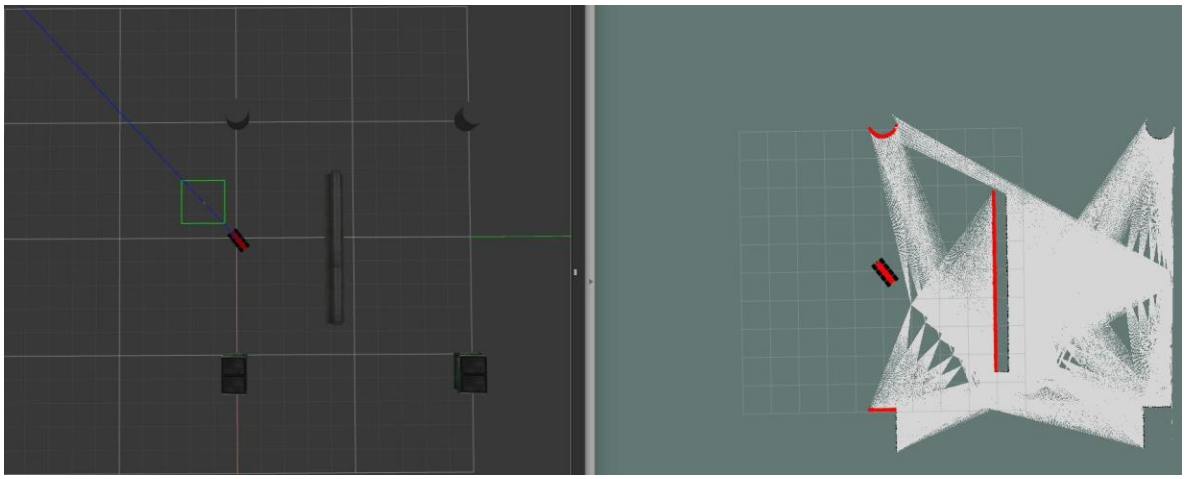


Figura 63. Ubicación del vehículo en la posición final (*goal*).

Analizando el comportamiento del vehículo durante la simulación se ha llegado a diferentes conclusiones. Una gran ventaja de este algoritmo es su simplicidad para desarrollarlo e implementarlo, no requiere un gran costo computacional, es medianamente eficiente para encontrar el punto final deseado y no sufre del problema mínimo local. Sin embargo, el algoritmo Bug2 tiene ciertas deficiencias, una de ellas es que no considera la cinemática del vehículo, otra es que debido a las dimensiones que tiene el vehículo, le cuesta un poco de trabajo poder seguir

en todo momento el objeto detectado sobre todo rodearlo por los bordes laterales. Hay momentos en que el movimiento se vuelve algo lento ya que se fía mucho de las lecturas que recibe del sensor láser y estas se van generando a medida que el robot se va moviendo.

5.4 Algoritmo de Localización Adaptiva de Monte Carlo (AMCL).

El algoritmo de AMCL es un algoritmo de localización probabilística para robots que se mueven en un ambiente en 2D. Este sistema implementa un filtro de partículas que usa para rastrear la posición del robot dentro de un mapa conocido, este algoritmo es altamente efectivo cuando se utilizan mapas generados en base a escaneos láser [50].

El filtro de partículas implementado proporcionará una mejor información sobre la posición real del robot, donde el robot tenga una mayor probabilidad de estar ubicado habrá una mayor concentración de partículas, mientras que en los lugares donde exista una menor probabilidad habrá un menor número de partículas. Cada una de estas partículas estará asociada a un peso que será igual a la probabilidad del estado de creencias en esa ubicación en especial.

El mapa que se genera es con el algoritmo Hector-SLAM, los pixeles en negro representan los bordes de los objetos y los pixeles en gris representan los espacios libres. En un principio se muestra una posición del robot la cual no es muy precisa, y por ello se implementa el filtro de partículas el cual debe minimizar este error y encontrar la localización más precisa dentro del mapa. Primero se genera una serie de hipótesis sobre la posición del robot, las cuales genera las partículas conocidas como "partículas discretas". Estas son extraídas de una distribución gaussiana y permiten establecer una serie de posiciones hipotéticas sobre la ubicación del robot.

Dependiendo de la posición de cada una de estas partículas discretas se orienta un escaneo del láser para posteriormente implementar un coeficiente de correlación de cada uno de ellos haciendo uso de la siguiente fórmula:

$$S = \frac{\Sigma m \Sigma \mathrm{n}(\mathrm{Amn} - \bar{A})(\mathrm{Bmn} - \bar{B})}{\sqrt{(\Sigma m \Sigma \mathrm{n}(\mathrm{Amn} - \bar{A})^2)(\Sigma m \Sigma \mathrm{n}(\mathrm{Bmn} - \bar{B})^2}}$$

Donde A es el mapa y B es el escaneo, $\bar{A}$ y $\bar{B}$ son los valores medios de los pixeles, donde las paredes tiene un valor de 1 y los espacios libres tiene un valor de 0, m y n son las coordenadas x y y de los pixeles. Esta ecuación cuenta el número de veces que los pixeles negros y naranjas se solapan entre sí, este proceso se repite en cada partícula y se almacena cada resultado obtenido. La partícula que más se alinea con el mapa tendrá el coeficiente de correlación mayor en comparación con las otras partículas, por lo tanto, se le dará un mayor peso y se escogerá como la posición en la que se encuentra el robot.

Sin embargo, cuando el peso de las partículas es demasiado grande después de varias iteraciones se puede caer en el error de un estado de creencia falso, por eso mismo las partículas con altos pesos tienen que volver a muestrearse cada cierto tiempo. Este remuestreo se lleva a cabo añadiendo más partículas cerca de las partículas con altos pesos, se resetea el peso de todas las partículas y se inicia el proceso otra vez. Con este proceso se asegura que ninguna partícula predomine de entre las demás, y que cualquiera tenga la misma oportunidad de ser prominente.

Este proceso se realiza en cada actualización del sensor, esto permite tener un registro de los pesos de cada partícula en cada instante, y así generar partículas en los lugares donde se estén generando mayores pesos y menos partículas en donde se estén generando menores pesos.

5.4.1 Desarrollo e implementación.

Para la implementación de este algoritmo con el modelo del vehículo 8X8, se han colocado diversos objetos en el espacio de simulación con el objetivo de darle un ambiente más real a la simulación.

Al igual que en el capítulo del algoritmo anterior, el mapa ha sido trazado utilizando el *plugin* del láser RPLIDAR-A3 y el algoritmo de mapeo y localización Hector-SLAM. Para el movimiento del vehículo se hizo uso del nodo de ROS que permite la navegación teleoperada [37] haciendo uso el teclado. Una vez creado el mapa, se guardó y se cargó en una nueva visualización en Rviz (figura 64).

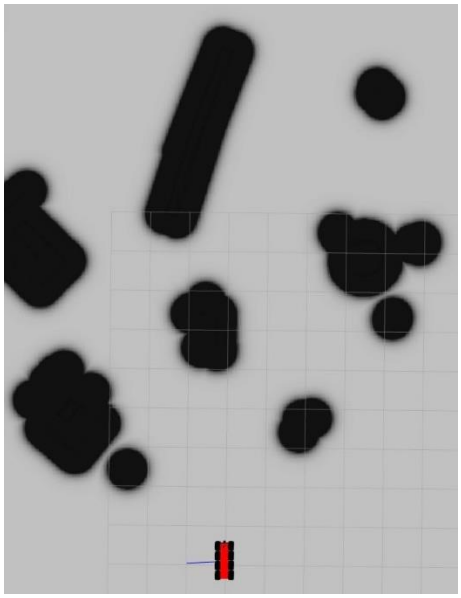
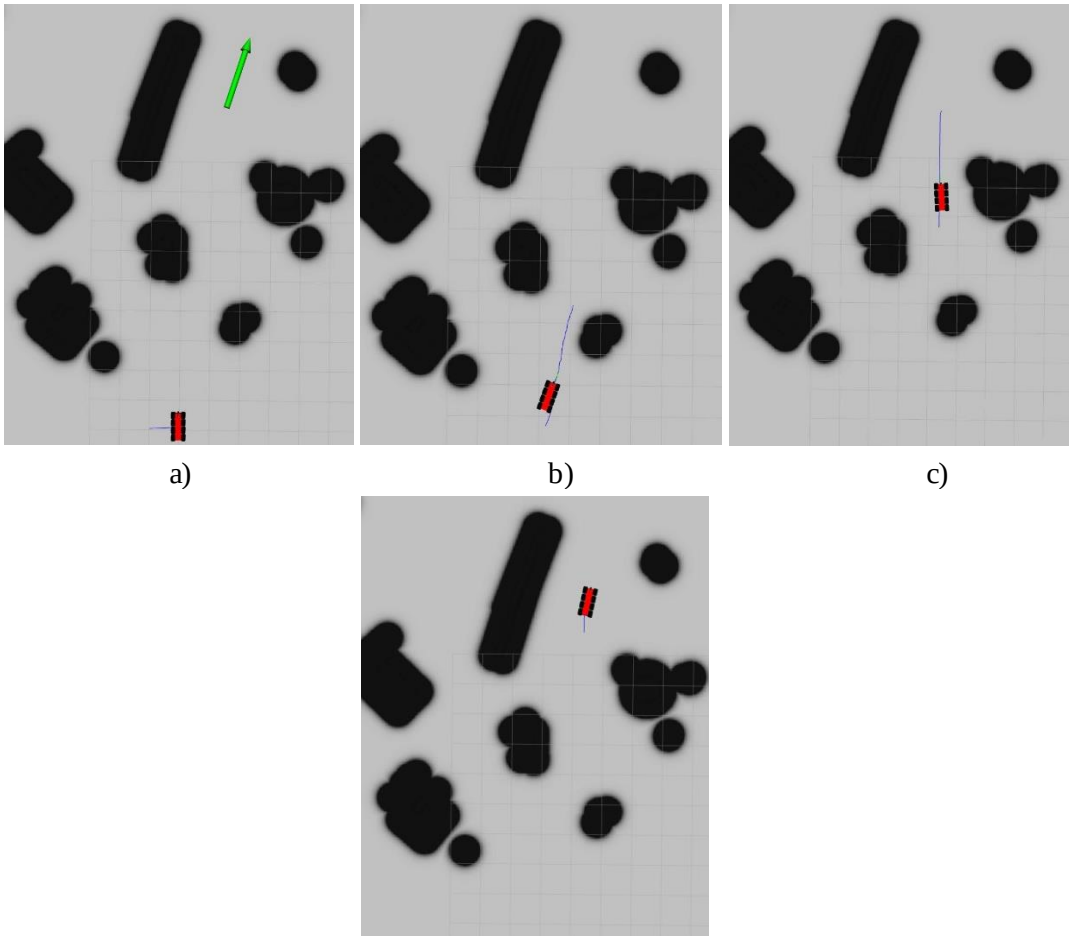


Figura 64. Mapa utilizado para pruebas con algoritmo AMCL

Posteriormente se cargó el algoritmo AMCL que proporciona la paquetería de ROS [50]. Para trazar el movimiento del vehículo y su trayectoria se han fijado unos parámetros locales y globales, así como también se han impuesto valores de velocidad máximos y mínimos para el movimiento rotacional y angular en z del vehículo.

Se hicieron 3 ensayos diferentes con este método. El primero (figuras), consistió en darle un objetivo (*goal*) que estuviera prácticamente en línea recta con el vehículo (imagen a, figura 65). El vehículo trazó su ruta global (de color azul) y conforme se iba moviendo iba trazando su ruta local (línea verde) en base a los obstáculos que iba encontrando en su camino (imagen b, figura 65). En los momentos en que el vehículo detectaba que estaba en un espacio libre de obstáculos aumentaba un poco la velocidad y cuando detectaba el borde de algún obstáculo, la reducía (imagen c, figura 65). Durante esta prueba solo tuvo que rodear un poco los obstáculos que se le presentaron en la parte frontal y seguir una ruta casi en línea recta con el objetivo final. Se consiguió llegar a la meta sin problema alguno (imagen c, figura 65).

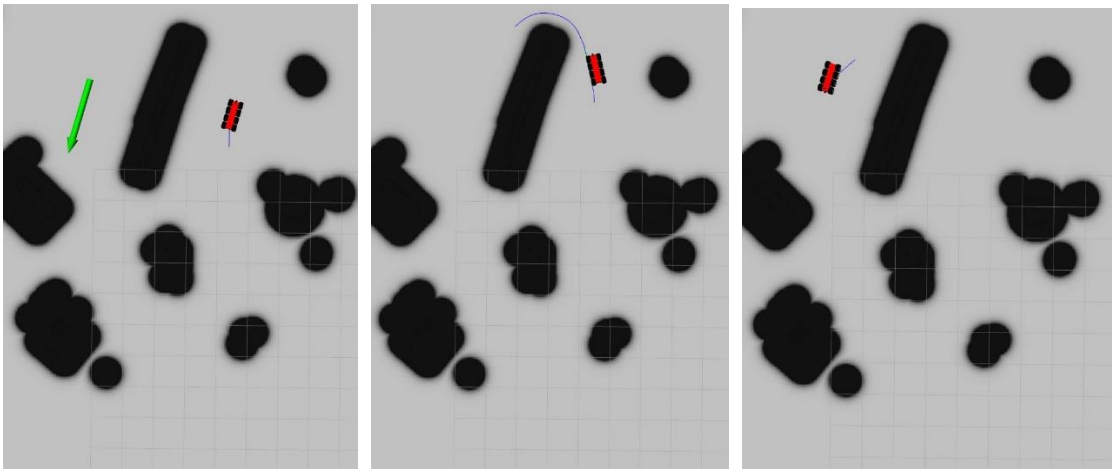


a) b) c)

d)

Figura 65. Trayecto realizado durante la primera prueba con AMCL

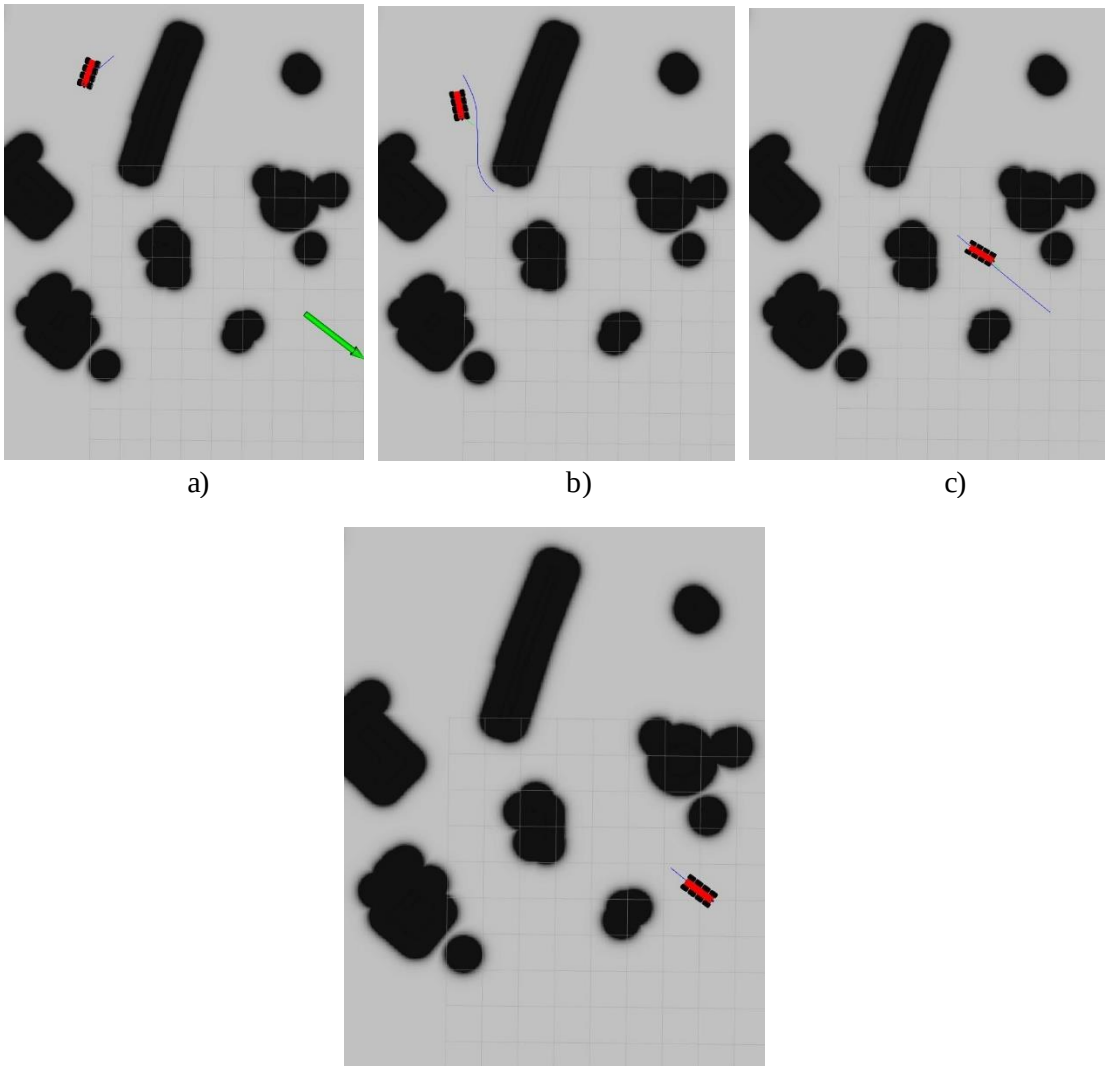
En la segunda prueba se le dio un objetivo cuya orientación era contrariara a la que tenía en ese instante, en este caso para verificar el desempeño del direccionamiento *skid steer* del vehículo. El punto inicial de esta prueba fue el punto final de la prueba anterior (imagen a, figura 66), al comenzar a ejecutarse el algoritmo el vehículo una vez más trazó su ruta global en color azul y mientras se movía hacia el objetivo (flecha verde) iba trazando la ruta local en color verde. El direccionamiento *skid steer* se ejecutó de manera satisfactoria mientras rodeaba los bordes del obstáculo que se encontraba a un costado de él (imagen b, figura 66), el vehículo llegó al objetivo final sin ningún percance (imagen c, figura 66).



a) b) c)

Figura 66. Trayecto realizado durante la segunda prueba con AMCL

La tercera y última prueba consistió en darle un objetivo alejado y que en su camino existieran diversos obstáculos los cuales tendría que esquivar o rodear. El punto final de la prueba 2 fue el punto inicial para esta prueba, el objetivo final (ubicación y orientación del vehículo) está representado con una flecha de color verde (imagen a, figura 67). A pesar de que el trayecto fue mayor que las 2 pruebas anteriores y que existía un mayor número de obstáculos, el vehículo una vez más llegó al punto final sin ningún incidente (imágenes b, c y d, figura 67).



a) b) c)

d)

Figura 67. Trayecto realizado durante la tercera prueba con AMCL

En las tres pruebas el vehículo se desempeñó correctamente mientras se ejecutaba el algoritmo AMCL. Durante el desarrollo de estas pruebas se encontraron las siguientes ventajas y desventajas del algoritmo AMCL.

Ventajas:

- Se logró conocer de manera eficaz el posicionamiento y orientación del robot dentro del mapa.
- Se generó una buena planeación de ruta basándose en los bordes de los obstáculos que hay en el entorno.
- El algoritmo genera un buen control de manejo de la velocidad del vehículo durante todo el trayecto de ruta y también a la hora de evitar o rodear los obstáculos.

Desventajas:

- A medida que aumentan las partículas, el costo computacional aumenta. Por ello es necesario especificar el número de partículas necesarias para poder estimar de manera precisa la posición del robot.

- Otra gran desventaja es que este algoritmo solo puede ser ejecutado con conocimiento previo del mapa.

6 Conclusiones y Desarrollos futuros

En este proyecto de fin de máster se presenta un modelo digital del vehículo 8X8 capaz de hacer mapas en 2D de su entorno con un láser RPLIDAR-A3 y moviéndose de manera autónoma. Para el desarrollo del mapa y la localización se utilizó el algoritmo Hector-SLAM, el cual brindó resultados satisfactorios tanto en simulación con el vehículo 8X8, como en pruebas reales con un vehículo de 4 ruedas (Robot Crawler).

Para la parte de navegación, se utilizaron y compararon 2 diversos algoritmos de planificación de ruta: Bug2 y AMCL. Como resultado de ese análisis comparativo, se determinan las ventajas e invconvenientes de cada uno de ellos.

En futuros proyectos se espera poder implementar todo el sistema de navegación, mapeo y localización realizado en este proyecto en el vehículo real 8X8. De igual manera dotar al vehículo real de un nuevo sistema de carga para las baterías. Una vez que el vehículo pueda navegar de manera autónoma dentro de un entorno real, podría indicársele una estación de carga a la cual podría acudir para recargar las baterías.

Este vehículo podría ser de gran uso como material didáctico o de pruebas para estudiantes de grado y/o máster.

7 Referencias

[1] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1271–1278, May 2016.

[2] T. Stoyanov, J. Saarinen, H. Andreasson, and A. J. Lilienthal. Normal distributions transform occupancy map fusion: Simultaneous mapping and tracking in large scale dynamic environments. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4702–4708, Nov 2013.

[3] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In ICRA Workshop on Open Source Software, 2009.

[4] A. M. Hellmund, S. Wirges, Ö. S. Tas, C. Bandera, and N. O. Salscheider. Robot operating system: A modular software framework for automated driving. 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pages 1564–1570, Nov 2016.

[5] J.C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, Norwell, MA, USA, 1991b. ISBN 079239206X.

[6] Joseph, L. (2017). ROS Robotics Projects. Birmingham: Packt Publishing Ltd.

[7] Thomas, D. (2014, May 22). [Online]. Introduction. Retrieved May 9, 2018, from ROS: http://wiki.ros.org/ROS/Introduction [Accessed: 11-Sept-2020]

[8] Ademovic, A. (2015). An Introduction to Robot Operating System: The Ultimate Robot Application Framework. Retrieved. [Online]. May 10, 2018, from Toptal: https://www.toptal.com/robotics/introduction-to-robot-operating-system [Accessed: 17-Sept-2020]

[9] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., et al. (2009). ROS: an open-source Robot Operating System. ICRA workshop on open source software, vol. 3, no. 3.2,5.

[10] [Online]. http://wiki.ros.org/roscore [Accessed: 1-Sept-2020]

[11] Clearpath Robotics. (2015). [Online] Intro to ROS. Retrieved May 9, 2018, from Clearpath Robotics: http://www.clearpathrobotics.com/assets/guides/ros/ [Accessed: 1-Sept-2020]

[12] Tellez, R. (2017, September 29). [Online] How to Start with Self-Driving Cars Using ROS. Retrieved May 10, 2018, from The Construct: http://www.theconstructsim.com/start-self-driving-cars-using-ros/ [Accessed: 14-Sept-2020]

[13] T. Foote. tf: The transform library. In 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pages 1–6, April 2013.

[14] Slamtec rplidar_ros. [Online]. Available at: https://github.com/Slamtec/rplidar_ros. [Accessed: 15-Sept-2020]

[15] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. IEEE Robotics Automation Magazine, 13(2):99–110, June 2006.

[16] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 32(6):1309–1332, 2016.

[17] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on, pages 155–160. IEEE, 2011.

[18] HY88. Rodriguez Castaño A. Universidad de Sevilla. 16, December, 2010.

[19] R.C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty", International Journal of Robotic Research, vol. 5, no. 4, pp. 56-68, 1986.

[20] M.W.M. Gamini Dissanayake, P. Newman, S. Clark, H.F. Durrant.Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem", IEEE Transactions on Robotics and Automation, vol. 17, pp. 229-241, 2001.

[21] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part I", IEEE Robotics and Automation Magazine, vol. 13, no. 3, pp- 108-117, 2006.

[22] Davison A.J., Reid I.D., Molton N.D. and Stasse O., "MonoSLAM: Real-Time Single Camera SLAM", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, 2007, pp. 1052-1067.

[23] Davison A.J., Cid Y.G. and Kita N., "Real-Time 3D SLAM with Wide-Angle Vision", IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 2004.

[24] Durrant-Whyte H. and Bailey T., "Simultaneous Localization and Mapping: Part I", IEEE Robotics & Automation Magazine, 2006, pp 99-108

[25] Sebastian Thrun and John J. Leonard. "Simultaneous Localization and Mapping." pages 871–889. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[26] Fofi David and Samia Ainouz. "Current state of the art vision based SLAM." Conference Paper - The International Society for Optical Engineering · February 2009. pages 5-9.

[27] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: "A factored solution to the simultaneous localization and mapping problem." In Eighteenth National Conference on Artificial Intelligence, pages 593–598, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[28] Shan Tixiao and Englot Brendan. "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain". Conference Paper - October 2018.

[29] E. Nelson, BLAM (Berkeley Localization and Mapping). [Online]. Available at: https://github.com/erik-nelson/blam [Accessed: 14-Nov-2020].

[30] W. Hess, D. Kohler, H. Rapp, and D. Andor. "Cartographer Documentation". 2018. [Online]. Available at: https://google-cartographer-ros.readthedocs.io/en/latest/ [Accessed: 14-Nov-2020].

[31] M. Kaess., A. Ranganathan., F. Dellaert. "iSAM: Incremental Smoothing and Maping. IEEE Transactions on Robotics, vol. 24, pp. 1365-1378, 2008.

[32] TurbtleBot package for Kinetic ROS. [Online]. Available at: https://wiki.ros.org/Robots/TurtleBot [Accessed: 4-Sept-2020].

[33] Summit-XL Robot package. [Online]. Available at:

https://github.com/RobotnikAutomation/summit_xl_common
[Accessed: 4-Sept-2020].

[34] Hokuyo Laser Node Package. [Online]. Available at: http://wiki.ros.org/hokuyo_node [Accessed: 4-Sept-2020].
[35] RPLIDAR ROS Laser Package. [Online]. Available at: http://wiki.ros.org/rplidar [Accessed: 4-Sept-2020].

[36] Hector-SLAM Package. [Online]. Available at:
https://github.com/NickL77/RPLidar_Hector_SLAM [Accessed: 1-Sept-2020].

[37] Turtlebot Keyboard Teleop Node. [Onlie]. Available at:
http://wiki.ros.org/turtlebot_teleop/Tutorials/Keyboard%20Teleop [Accessed: 1-Sept-2020].

[38] ROS Joint State Publisher Package. [Online]. Available at:
http://wiki.ros.org/joint_state_publisher [Accessed: 18-Nov-2020].

[39] rf2o_laser_odometry package. [Online]. Available at:
http://wiki.ros.org/rf2o_laser_odometry [Accessed: 20-Nov-2020].

[40] KaganE., Nir Shvalb, Ben-Gal I. Autonomous Mobile Robots and Multi-Robot Systems. Wiley. December 2019

[41] Canny, J. (1988). The Complexity of Robot Motion Planning. Cambridge, MA: MIT press.

[42] Kim, J.-O. and Khosla, P. (1992). Real-time obstacle avoidance using harmonic potential functions. IEEE Transactions on Robotics and Automation 8 (3): 338–349.

[43] Kavraki, L., and Latombe, J.-C. (1998). Probabilistic roadmaps for robot path planning.

[44] S. Prentice and N. Roy. The belief roadmap: Efficient planning in blief space by factoring the covariance. International Journal of Robotics Research, 28(11–12):1448–1465, 2009.

[45] R. Tedrake, I. R. Manchester, M. M. Tobekin, and J. W. Roberts. LQR-trees: Feedback motion planning via sums of squares verification. International Journal of Robotics Research (to appear), 2010.

[46] Dijkstra, E. (1959). A note on two problems in connexion with graphs. Numerische Mathematik 1: 269–271.

[47] Marin-Plaza O, Ahmed Hussein, Martin D., and de la Escalera A. Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. Journal of Advanced Transportation. Universidad Carlos III. Madrid, Leganes, Spain. 22 Feb 2018.

[48] Medina, O., Shapiro, A., and Shvalb, N. (2016). Kinematics for an actuated flexible n-manifold. Journal of Mechanisms and Robotics 8 (2): 021009.

[49] Distance from a point to a line. [Online]. Available at:
https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line [Accessed: 10-Nov-2020]

[50] AMCL ROS Package. [Online]. Available at: http://wiki.ros.org/amcl [Accessed: 15-Nov-2020]