UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

NAVEGACIÓN AUTÓNOMA DE UN ROBOT MÓVIL EN CONFIGURACIÓN DIFERENCIAL USANDO UN SENSOR LIDAR.

TESIS PARA OBTENER EL GRADO DE
MAESTRO EN ROBÓTICA

PRESENTA:
SALADO CHÁVEZ KARLA ITZEL

DIRECTOR DE TESIS:
DR. OSCAR DAVID RAMÍREZ CÁRDENAS

CO-DIRECTOR DE TESIS:
DR. JOSÉ ANIBAL ARIAS AGUILAR

H. CD. DE HUAJUAPAN DE LEÓN, OAXACA, MÉXICO, FEBRERO 2025

A mi mamá, a mi hermano
y a mis abuelos Inés y Aurelio,
por el amor, el apoyo y la fuerza
que me han dado en cada paso de mi vida.

Agradecimientos

A mi mamá, por ser la fuerza que necesito en cada paso que doy, por su amor y por ser mi guía, incluso cuando me equivoco. Ella siempre me orienta hacia el camino correcto. A mi hermano, por ser mi compañero de vida, por estar siempre a mi lado en cada adversidad, brindándome su apoyo incondicional. A mis abuelos Inés y Aurelio, por su amor y cuidado, por siempre apoyarme y ser un ejemplo de perseverancia, mostrándome que nunca debo rendirme.

Quiero agradecer de manera especial a mi director, el Dr. Oscar, por su guía a lo largo de este trabajo. Gracias por confiar en mí y permitirme enfrentar los retos que surgieron durante el desarrollo de esta tesis. Aprecio profundamente sus consejos, no solo sobre el trabajo, sino también sobre cómo cuidar mi salud, y por la amistad que me brindó durante estos dos años.

A mis tíos Alberto y Jorge, y especialmente a mi tío Marco, por su ejemplo de bondad, gratitud, altruismo y me ha enseñado que ser ingeniero es una actitud frente a la vida, una manera de contribuir al bienestar de los demás. Gracias por su apoyo, por su cariño y por ser un modelo a seguir, no solo en lo profesional, sino también como ser humano. Gracias por el apoyo y cariño que siempre me han dado. A toda la familia Chávez y a toda la familia Soto, por su apoyo incondicional a lo largo de estos años y por recibirme siempre con mucho

amor en sus hogares, en especial a mis tíos Mireya, Miguel, Marcos y Alicia. A mi familia de Huatulco, a mis tíos, la Dra. Ruth y el Dr. Julio, a Paty y Manuel, y a mis primos Julio y Lupita, por su constante apoyo. Espero que la vida nos permita acompañarnos en cada etapa. Y a Saraí y Alejandro, por cuidar de mí siempre en Huajuapan.

A Claudia, por la increíble amistad que mantuvimos estos dos años, por ser mi apoyo emocional y por siempre ponerle brillo a mi vida. Gracias especialmente por tu ayuda en la revisión de estilo y la corrección de las imágenes de esta tesis. A Filiberto, por su amistad, y a Omar, el encargado, gracias a quienes sus recomendaciones y conocimientos fueron fundamentales para la realización de este trabajo.

A Karina, Julieta, Shamna, Mafer por todas las risas, el amor y llenar mi vida de color y por hacerme sonreír cuando más lo necesitaba. A Benito, Diego Morán, Benny, Jovani, Diego Fernández, por levantarme cada vez que caí. No encuentro palabras suficientes para agradecer su apoyo. A Eduardo, Daniel, Lucía y Cristal por su amistad inquebrantable. A Jorge Pizarro, Yesi, Jaqui, Adaya, Kristof, Lalo, Sunnie y Nuria por su compañía durante la realización de esta tesis. Y a todos los amigos que hice en el camino, que de alguna manera u otra me acompañaron durante la elaboración de esta tesis.

A CONACYT, por su apoyo económico, que fue esencial para la elaboración de esta tesis. A mi co-director, el Dr. Aníbal, por compartir sus conocimientos sobre Aprendizaje por Refuerzo, y a mis sinodales, por sus consejos y correcciones, que contribuyeron a la mejora de esta tesis. Finalmente, quiero expresar mi agradecimiento a la Universidad Tecnológica de la Mixteca, por brindarme los recursos y el entorno académico para llevar a cabo este trabajo. Agradezco también a los encargados de los talleres, en especial a Don Rober, Don Roberto y Don Enrique, por su apoyo. Al personal de intendencia y mantenimiento, quienes se encargaron de mantener las instalaciones en un ambiente agradable para trabajar.

Resumen

Este proyecto se enfoca en el desarrollo de un sistema de navegación autónoma para robots móviles en entornos desconocidos, con el objetivo de lograr desplazamientos eficientes y seguros sin intervención humana. Para ello, se utilizó SLAM para la generación del mapa del entorno. Se evaluaron dos enfoques principales para la navegación: la navegación *offline* y la navegación *online*.

En la navegación *offline*, primero se obtiene el mapa del entorno y luego se emplea el algoritmo PRM para generar una trayectoria. Posteriormente, se utiliza control cinemático para seguir dicha trayectoria. También se empleó Aprendizaje por Refuerzo (RL), lo que permite al robot evadir obstáculos y adaptar su navegación a partir de la información obtenida del mapa generado.

Para la navegación *online*, SLAM se utiliza para la obtención incremental del mapa mientras el robot avanza. En este caso, se aplicaron dos variantes: una en la que el PRM genera trayectorias parciales basadas en la información obtenida hasta el momento, que el robot sigue mediante control cinemático; y otra en la que el robot emplea RL para generar submetas según la información del mapa obtenido, adaptándose continuamente a las características del entorno.

Índice general

Índice de figuras

Índice de tablas

Glosario

A* A-estrella.

ABC *Artificial Bee Colony* (Colonia de Abejas Artificiales).

AGV *Automated guided vehicle* (Vehículo de guiado automático).

AUV *Autonomous Underwater Vehicle* (Vehículo submarinos autónomos).

BFS *Breadth-First Search*(Búsqueda en anchura).

CAD *Computer-Aided Design,*(Diseño Asistido por Computadora).

DDPG *Deep Deterministic Policy Gradient*(Gradiente de Política Determinista Profundo).

EKF *Extended Kalman Filter* (Filtro de Kalman Extendido).

GPS *Global Positioning System*(Sistema de Posicionamiento Global).

IA Inteligencia Artificial.

LiDAR *Light Detection And Ranging* (Detección y medición de la luz).

PCM Punto más cercano a la meta.

PRM *Probabilistic roadmap* (Hoja de Ruta Probabilística).

RL *Reinforcement Learning* (Aprendizaje por refuerzo).

RMR Robots Móviles con Ruedas.

RN Redes neuronales.

ROS *Robot Operating System* (sistema operativo robótico)).

rqt *ROS Qt-based GUI Framework*(Marco GUI basado en ROS Qt).

RRT *Rapidly Exploring Random Tree* (Árbol Aleatorio de Exploración Rápida).

SLAM *Simultaneous Localization and Mapping* (Localización y Mapeo Simultáneos).

TD3 *Twin Delayed DDPG*(Gradiente de Política Determinista con Retraso Gemelo, en su tercera versión).

UGV *Unmanned ground vehicle* (Vehículo terrestre no tripulado).

UTM Universidad Tecnológica de la Mixteca .

Capítulo 1

Introducción

En los años sesenta se introdujeron los robots manipuladores en la industria con el fin de mejorar el proceso productivo, su desarrollo se basó en que fueran más rápidos, tuvieran mayor precisión y en disminuir la complejidad en su programación. Estos robots, tenían tareas repetitivas y estaban diseñados para ser ubicados en un área específica, considerando su espacio de trabajo fijo como la máxima extensión de sus articulaciones, lo cual era complejo debido a que las células industriales sufrían progresivas ampliaciones [1].

Para ampliar el área de trabajo de los robots industriales, en los años ochenta, se desarrollaron los primeros robots móviles con el propósito de que pudieran desplazarse sobre rieles para proporcionar un transporte eficaz, los primeros robots móviles fueron Vehículos Guiados Automáticamente (AGV's, por sus siglas en inglés), estos robots eran programados con una secuencia de acciones. Sin embargo ante cualquier cambio inesperado en el entorno, el robot no tenía la capacidad para ejecutar acciones alternativas que le permitan reanudar su labor [1]. Ante las potenciales aplicaciones fuera del ámbito industrial, en los años noventa, se desarrollaron vehículos con la capacidad de desenvolverse en cualquier tipo de ambiente, con un mayor grado de inteligencia y percepción surgiendo así, los robots móviles inteligentes, los cuales tienen la capacidad de desplazarse en entornos no estructurados de los que se posee un conocimiento incierto [1].

Un robot móvil se puede definir como un sistema electromecánico con cierto nivel de autonomía, es decir que tiene la capacidad de desplazarse mediante dispositivos de locomoción sin estar sujeto físicamente a un solo punto, se integran de componentes electrónicos, mecánicos, eléctricos, sistemas de comunicación con un sistema informático con el cual controla el mecanismo en tiempo real y le permite percibir su entorno [2]. Para esto último se emplean sensores con los que monitorea la posición relativa de su punto de origen y de destino, generalmente operan en ambientes no estructurados por lo que se debe de enfrentar a incertidumbres al detectar la posición e identificar objetos, así mismo para este tipo de robots, se emplea control en lazo cerrado [3].

Los robots móviles se clasifican por el tipo de locomoción: por ruedas, por patas y por orugas, sin embargo ha habido mayor desarrollo en los Robots Móviles con Ruedas (RMR), debido a su eficiencia en superficies lisas y firmes, esta configuración no causa desgaste en la superficie donde se desplaza el robot, y sus componentes son menos complejos en comparación con los robots de patas y de orugas, esto permite sencillez en su construcción [1].

Como se mencionó anteriormente, los robots móviles cuentan con cierto grado de autonomía, por lo que deben de tener la capacidad para percibir, modelar, planificar y actuar en un entorno, con la finalidad de alcanzar determinados objetivos sin la intervención del hombre, como por ejemplo los Vehículos Terrestres no Tripulados (UGV, por su siglas en inglés) los cuales tienen la capacidad de percibir su entorno y realizar acciones con la finalidad de cumplir los objetivos programados en ambientes con cierta incertidumbre [4]. Existen diferentes dispositivos con diferente grado de autonomía, el cual depende de su facultad para obtener información de su entorno y procesar las señales en acciones. Las dos características principales son:

- Percepción: Es la capacidad de determinar la relación con su entorno mediante un sistema sensorial. Se puede definir como la síntesis de la información adquirida por los sensores, con la finalidad de generar mapas de su entorno.
- Razonamiento: Es la capacidad de decidir qué acciones son requeridas, según el estado del robot y del entorno, con la finalidad de cumplir su objetivo, se basa en la planificación de trayectorias y en la habilidad de modificarla ante obstáculos inesperados.

Los robots autónomos suelen ser empleados en medios con condiciones extremas Figura 1.1, extraído de [5], a las que el humano no puede acceder y por ende no tiene conocimiento del área en la que el robot va a operar [5], por ejemplo los Vehículos Submarinos Autónomos (AUVs, por sus siglas en inglés) [6], los cuales son empleados para tareas de exploración científica, oceanografía y arqueología submarina. Idealmente estos robots no necesitan de la intervención del hombre, por lo que deben de tener la capacidad de desplazarse en un medio desconocido y de almacenar datos del entorno para luego ser analizados.

(a) Bajo el mar (b) Bajo tierra (c) Otros planetas

Figura 1.1: Aplicaciones de la robótica móvil

En este proyecto se plantea la implementación del algoritmo de Mapeo y Localización Simultánea (SLAM, por su siglas en inglés) para la generación de un mapa del entorno en el que se desea que el robot opere y posteriormente emplear Aprendizaje por Refuerzo (RL, por sus siglas en inglés) como técnica de navegación autónoma que permita que un robot móvil pueda navegar en un entorno.

1.1. Estado del Arte

Para lograr autonomía, un robot debe tener conocimiento de su punto de inicio, su destino y su ubicación en un momento específico, además de contar con una técnica de navegación. Existen dos categorías principales para la planificación de rutas en robots móviles: los algoritmos basados en mapas y los algoritmos reactivos los cuales se fundamentan en la percepción en tiempo real.

Los algoritmos de planificación de rutas se basan en el conocimiento completo del mapa para desplazarse de manera segura en el entorno. Para ello, se considera una trayectoria basada en un ambiente previamente conocido. Estos métodos producen rutas optimizadas, pero no son útiles en escenarios con obstáculos desconocidos o dinámicos [7]. Entre los algoritmos más comunes se encuentran la descomposición trapezoidal, Voronoi, el Método de Hoja de Ruta Probabilística (PRM, por sus siglas en inglés) y el Árbol Aleatorio de Exploración Rápida (RRT, por sus siglas en inglés). Asimismo, existen algoritmos que permiten optimizar las rutas, como Dijkstra y A* (A-estrella).

En el año 2009 se desarrolló un sistema de navegación autónoma por Bonilla [5], con el objetivo de encontrar la ruta más cercana y libre de colisiones en un mapa de localización tipo rejilla, en el cual se aplica la localización y mapeo simultáneo, empleando herramientas de lógica difusa y procesamiento de imágenes. Se implementó un algoritmo de Dijkstra debido a su rápida ejecución y pocos recursos computacionales. En la Figura 1.2 se muestran los resultados obtenidos por [5].

En 2012, Carvajal et al. [8] compararon el desempeño de dos técnicas de planificación: la descomposición exacta trapezoidal y la descomposición adaptativa de celdas a través de mallas, implementadas en un ambiente de desarrollo *Java-LeJOS*. Se probaron en tres escenarios, a partir de los cuales se dedujo que el recorrido es más corto cuando se considera la geometría en comparación con la representación del entorno como un punto en el espacio. Se obtuvieron mejores resultados al emplear la descomposición exacta de celdas en relación con la distancia total recorrida. Ambas técnicas mostraron un desempeño similar en cuanto al error generado en la localización real en comparación con la deseada, una vez alcanzado el punto final de la trayectoria obtenida.

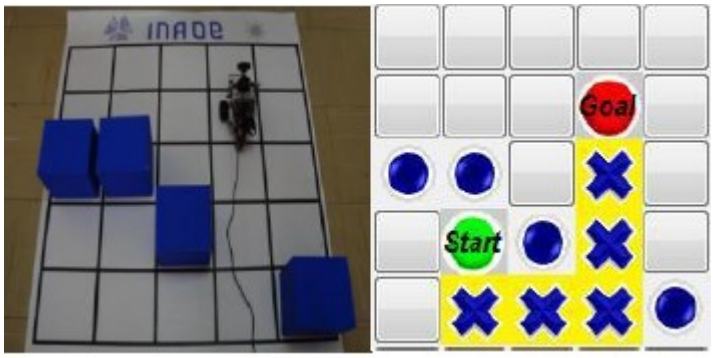


Figura 1.2: Robot con técnicas de localización y planificación de rutas.

En el año 2014 Santa, et al. [9] proponen el uso de un algoritmo de planificación de rutas basado en diagramas de Voronoi, para robots móviles que operan en ambientes interiores y desconocidos, implementando una navegación híbrida compuesta por los diagramas de Voronoi y la triangulación de Delaunay en escenarios estáticos, evitando así posibles colisiones que permite implementar un algoritmo utilizando la métrica euclidiana, con un costo computacional bajo.

Se diseñó un sistema automatizado de almacenamiento y recuperación (AS/RS) el cual puede almacenar materiales sin intervención humana en el 2017 por Qing [10]. Empleando un AGV para transportar la carga desde un nodo de inicio hasta uno de destino en un entorno determinado. Basado en una planificación de ruta efectiva, el AGV puede completar la tarea en el menor tiempo posible, ahorrar energía y mejorar la eficiencia también. Las simulaciones demuestran que el algoritmo Dijkstra mejorado puede resolver el problema de planificación de ruta para encontrar los caminos óptimos con distancia y tiempo para el AGV.

Uno de los algoritmos utilizados para la planificación de trayectoria segura ha sido PRM, por ejemplo, en el año 2018 Alpkiray et al. [11], con la finalidad de conducir un robot hasta un destino, realizaron la fusión con el método de Colonia de Abejas Artificiales (ABC, por sus siglas en inglés), el cual tiene la capacidad de encontrar la ruta óptima, sin embargo tiene un costo alto en tiempo computacional. El algoritmo PRM-ABC combina las ventajas de PRM y el ABC, por lo que optimiza la distancia del camino, la suavidad y la seguridad. Este algoritmo es más rápido que ABC porque tiene capacidad de optimización.

Un año más tarde, Chen et al. [12], proponen una mejora para PRM, introduciendo la teoría de Campos Potenciales, denominando este nuevo método como P-PRM, el cual utiliza el campo potencial para seleccionar los nodos importantes para evitar colisiones, con lo que se demuestra que el algoritmo P-PRM es más óptimo que el algoritmo PRM debido a la eficiencia de planificación es mayor y el tiempo de cálculo y la distancia a recorrer es menor,

esto se verificó por simulación en el espacio tridimensional.

Véras et al. en el 2019 desarrollaron el algoritmo basado en RRT, [13] donde se propone un nuevo proceso de muestreo basado en cuadrículas de Sukharev y vértices convexos de los cascos de seguridad de los obstáculos. Se realizan pruebas computacionales para comprobar que la nueva estrategia de muestreo mejora el tiempo de planificación, que se puede aplicar a la navegación en tiempo real de vehículos aéreos no tripulados (AUV, por su siglas en inglés). Los resultados presentados indican que el uso de vértices convexos y la cuadrícula de Sukharev aceleran el tiempo de planificación del RRT y muestra un mejor rendimiento en varios entornos de navegación con diferentes cantidades y distribuciones espaciales de obstáculos poligonales.

En [14] se buscó la la generación de trayectorias cortas y seguras en entornos con obstáculos, por lo que Al-Dahhan y Schmidt en el 2019 proponen un nuevo método híbrido basado en el cálculo de un diagrama de Voronoi y PRM. Su evaluación muestra que el algoritmo propuesto calcula caminos cortos con mayor seguridad en comparación con el algoritmo PRM clásico, además, solo se necesita una pequeña cantidad de nodos debido al uso del diagrama de Voronoi para la generación de rutas.

Erke et al. [15] durante el 2020 desarrollaron un algoritmo de planificación de rutas para vehículos terrestres autónomos en el cual, se introduce una evaluación para el rendimiento de diferentes algoritmos, para ello se emplea una directriz generada por la planificación para desarrollar la función heurística para superar la deficiencia de los algoritmos A* tradicionales. Para mejorar el rendimiento de evasión de obstáculos, se emplean puntos clave alrededor del obstáculo, lo que guiaría la ruta de planificación para esquivar el obstáculo mucho antes que el tradicional, también se incluye un nuevo algoritmo A* basado en pasos variables para reducir el tiempo de cálculo del algoritmo propuesto. En comparación con las técnicas más modernas, los resultados experimentales muestran que el rendimiento del algoritmo propuesto es robusto y estable.

En 2021 [16] Bravo et al. desarrollaron un sistema de planificación de rutas para recoger desechos sólidos ubicados en diferentes puntos, el método que se empleo consiste en obtener puntos georreferenciados, la información es recopilada por un servidor de mapas, para luego convertir la información en un grafo dirigido y ponderado, posteriormente se aplicó el algoritmo de Dijkstra tomando los datos del grafo dirigido, el punto de partida y los puntos de acción, los cuales son los focos infecciosos, el algoritmo debe realizar el cálculo del vértice adyacente varias veces hasta hallar la mejor ruta es decir la ruta más corta.

En el 2022, Palacios et al. [17] presentan el análisis de tres técnicas de planificación de trayectorias para resolver un laberinto, el objetivo del sistema es que la bola recorra desde un punto inicial hasta un punto de destino, para lo cual se implementaron los algoritmos

RRT, PRM y los diagramas de Voronoi junto con el algoritmo de búsqueda A*. Los principales resultados muestran que el algoritmo RRT la trayectoria con mayor longitud y menor tiempo de ejecución, el algoritmo PRM genera la trayectoria con menor longitud, sin embargo, tiene el peor rendimiento. Finalmente, la técnica con diagramas de Voronoi presenta la trayectoria más suave y equidistante entre las paredes del laberinto.

Sin embargo, no siempre es posible contar con el mapa del entorno o en caso de que el robot se encuentre en un entorno dinámico, es necesario utilizar algoritmos reactivos basados en percepción en tiempo real, estos métodos de planificación de ruta local no necesitan información del entorno, sino que va generando y obteniendo la información de los sensores integrados, al mismo tiempo decide cómo navegar para llegar a su destino. Estos algoritmos son muy útiles en entornos desconocidos o dinámicos, donde no se dispone de un mapa previamente generado [7]. Entre los cuales se encuentran los algoritmos de navegación autónoma con campos potenciales, algoritmos de Mapeo y Localización Simultánea (SLAM, por su siglas en inglés), y de navegación basados en aprendizaje profundo, con los que se dota al robot de la capacidad de moverse en entornos donde no existe un mapa previo.

En el 2012, Espitia y Sofrony [18] propusieron un algoritmo que logró evadir mínimos locales, cuando se emplean campos potenciales, la técnica consiste en utilizar un modelo de partículas activas brownianas, con la finalidad de emplear un modelo compacto que permita la planificación de trayectorias. Siguiendo con esta idea en el año 2016 Osorio et al. [19] desarrollaron un algoritmo de campos potenciales asociado al uso de la evasión de mínimos locales y lograr que la navegación del robot sea óptima. Esta propuesta redujo el tiempo de máquina y la velocidad de operación del robot móvil, debido a que centra su trabajo en encontrar el mínimo local en un movimiento circular, con lo que se obtuvo un resultado 20 % más eficiente que al implementar el algoritmo A* en conjunto con campos potenciales.

La definición del problema de SLAM surgió en la Conferencia de Automatización y Robótica IEEE de 1986 [20] con la finalidad de combinar la localización y el mapeo ya que se requería de un estado conjunto, compuesto por la posición del vehículo y la posición del punto de referencia. A partir de esta conferencia se desarrollaron una serie de documentos. Tal es el caso de Durrant Whyte [21] quien demostró un alto grado de correlación entre la estimación de la ubicación de diferentes puntos de referencia en mapas. Las investigaciones de Crowley y Chatila dieron como resultado un artículo de Smith et al. [22] en el cual se muestra que un robot móvil en un entorno desconocido toma observaciones relativas de puntos de referencia que están correlacionadas debido al error que se presenta en la estimación de la ubicación del vehículo [23].

Se implementó la navegación autónoma en sillas de ruedas, con la finalidad de disminuir el esfuerzo requerido por el usuario, al maniobrar en ambientes restringidos. La aplicación del algoritmo SLAM genera un mapa de probabilidad asociado al mapa geométrico construi-

do, con el cual se obtiene un camino seguro que le permite al vehículo alcanzar la orientación deseada por Cheenin et al. [24] en el 2011. En la Figura 1.3, extraída de [24], se muestra el mapa construido por el algoritmo SLAM. Los segmentos negros son las paredes y las puertas del entorno; los círculos verdes a esquinas y magenta son los datos del sensor; la línea punteada azul es el camino por el que se ha desplazado la silla de ruedas. El SLAM es empleado principalmente para entornos estáticos, para entornos dinámicos se ha recurrido al uso de redes neuronales, con ello se elaboran estrategias de evasión de obstáculos dinámicos sin el conocimiento del entorno. Basándose en esta idea, Martínez [25] diseñó un controlador empleando navegación con red neuronal electrónica.

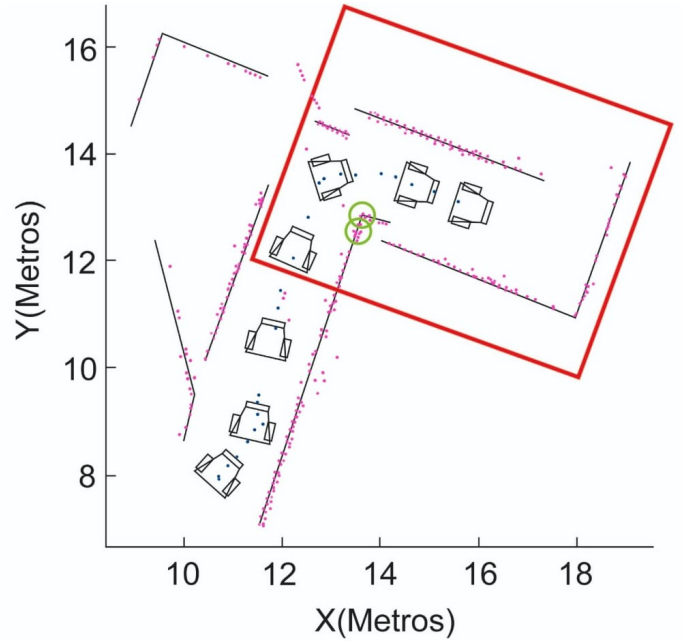


Figura 1.3: Mapa generado por el algoritmo SLAM

En el 2021, Toan and Gon-Woo [26] implementaron en un robot en movimiento (*Turtlebot 2*) una cámara, un sensor LiDAR 2D (Hokuyo), basándose en la simulación de Gazebo para moverse en un entorno de simulación con paredes, obstáculos y peatones (Figura 1.4, tomada de [26]). Con la finalidad de comprobar que la exploración ambiental es mejor aplicando la política de Boltzmann ya que esta conduce a un mejor equilibrio entre la explotación y la exploración, esto permite que el robot pueda elegir un camino más corto para llegar a una meta. La principal contribución es la capacidad de explorar el entorno para cumplir los objetivos en diferentes entornos de prueba.

Figura 1.4: El entorno de entrenamiento.

En ese mismo año Zou et al. [27] realizaron el análisis y compararon diferentes métodos de navegación en interiores basados en LiDAR SLAM, el cual permite proporcionar la localización mediante el uso de información espacial capturada por nubes de puntos LiDAR, en las últimas décadas se han propuesto varios métodos entorno al LiDAR SLAM, sin embargo, no son claras las ventajas y desventajas, por lo que se realizan extensos experimentos para evaluar su rendimiento en entornos reales, los resultados de los análisis comparativos pueden ayudar a los investigadores a construir un sistema LiDAR SLAM adecuado para la navegación.

En los últimos años, se han desarrollado investigaciones enfocadas al RL, especialmente en robots móviles autónomos, debido a su aplicación en problemas de navegación sin mapas, ya que permite completar los objetivos establecidos en diferentes entornos, sin mapas ni rutas preestablecidas. Sin embargo, para la navegación sin mapas basada en el RL, el equilibrio entre la explotación y la exploración son cuestiones que deben considerarse cuidadosamente. La capacidad que tiene un robot de descubrir y ejecutar acciones, en un entorno de trabajo es importante en la mejora del rendimiento del aprendizaje por refuerzo, al crear ruido durante el entrenamiento de la red neuronal convolucional, el problema de rendimiento puede resolverse mediante algunos enfoques populares en la actualidad [26].

En el 2017 Xin, et al. [28] implementaron la tecnología RL profunda para la planificación de la ruta con la finalidad de determinar la acción óptima para que el robot móvil llegue a un objetivo, evitando obstáculos. Los resultados experimentales muestran que el método de planificación de ruta basado en RL profundo es un método efectivo. En el 2022 Rufo [29]

diseñó una red y una función de recompensa capaz de entrenar agentes que logren alcanzar el destino en escenarios sencillos, por medio de aprendizaje por refuerzo para controlar un vehículo con la finalidad de desplazarse en un escenario con obstáculos hasta alcanzar una meta, se empleó dicha técnica debido a que no se necesita de un set de entrenamiento, para ello solo se necesita definir un entorno con el que el vehículo pueda interactuar para realizar pruebas, otra ventaja de esta técnica es que no se tiene un límite de aprendizaje, pues mientras más interacción con entorno, habrá un mejor desempeño.

Uno de los primeros en implementar una red neuronal en el algoritmo Filtro de Kalman Extendido (EKF, por sus siglas en inglés) en un problema de SLAM fueron Choi et al. [30] cuyo objetivo fue presentar un EKF basado en una red neuronal para tratar el error sistemático, diferenciando entre la planta real y el mejor modelo, ya que para el procesamiento de datos imprecisos o ruidosos por las redes neuronales es más eficiente que las técnicas clásicas porque la red neuronal es altamente tolerante a los ruidos [31].

En el 2021 Rodas [32] presenta un sistema autónomo de un robot móvil tipo *Skid Steer-con* con la finalidad de navegar en un ambiente desconocido de forma autónoma, el cual tiene la capacidad de generar mapas en 2D. Para ello se implementó el algoritmo SLAM para la creación de mapas, y se propusieron Dijkstra y A*, y seguimiento de trayectoria como segundo método para la navegación. Con finalidad de comparar el desempeño se implementa RL, utilizando el algoritmo *Deep Q-Learning* para la navegación autónoma del robot en un entorno simulado.

1.2. Planteamiento del Problema

En cuanto a la navegación autónoma, Tesla ha presentado avances para ayudarle al conductor con las partes más agotadoras de la conducción implementando una función de Piloto automático la cual permite que el automóvil gire, acelere y frene de forma automática dentro de un carril, sugiere cambios para optimizar su trayecto, y hace ajustes para evitar que se quede atascado detrás de automóviles lentos, sin embargo necesitan una supervisión activa del conductor y no permiten que el coche sea autónomo [33].

El *Perseverance rover* de la NASA, tiene la capacidad de desplazarse en Marte utilizando un sistema de navegación automática recientemente mejorado, con la finalidad de buscar signos de vida antigua. A futuro el *Perseverance rover* se hará cargo de la unidad por sí mismo, utilizando un poderoso sistema de navegación automática, llamado *AutoNav*, este sistema mejorado hace mapas 3D del terreno por delante, identifica peligros y planea una ruta alrededor de cualquier obstáculo sin una dirección adicional de los controladores en la Tierra [34].

En cuanto a la robótica, la navegación es la base para un sistema móvil, por lo que se deben de planificar la misión, la ruta y la evasión de obstáculos. Por ejemplo para un robot que se desplaza en interiores, la misión es determinar a qué habitación se debe desplazar. La ruta se determina desde el inicio del recorrido hasta llegar al destino, considerando que esta no contempla la presencia de objetos inesperados [5].

En México se han dotado robots con navegación autónoma en diversas universidades, empleando algoritmos como Dijkstra y SLAM [5]. En la Universidad Nacional Autónoma de México (UNAM) se diseño un sistema de navegación para un vehículo autónomo usando técnicas de visión robótica, enviando imágenes de forma inalámbrica a una computadora, empleando un sistema de control difuso, la computadora envía en forma inalámbrica las señales al vehículo [35].

La robótica móvil busca ejecutar tareas de transporte, búsqueda, rescate y limpieza [5, 36] para las cuales los robots necesitan de un mapa del entorno, al contar con un mapa preciso le permite al robot móvil operar en entornos complejos en función de sensores sin depender de un sistema de referencia externo como el Sistema de Posicionamiento Global (GPS, por sus siglas en inglés). Por lo que se necesita un sistema de navegación que obtenga información del entorno y posteriormente dote al robot de la capacidad de navegar en el entorno.

Debido a que en ambientes interiores no se cuenta con estos sistemas, se han desarrollado técnicas para generar mapas de aprendizaje bajo incertidumbre como el SLAM [37], el cual se basa en combinar la localización y el mapeo, requiere de un estado compuesto por la posición del vehículo y la posición del punto de referencia [21]. Para la navegación se ha recurrido al uso de redes neuronales con la finalidad de generar estrategias de evasión de obstáculos [32], debido a que el comportamiento es reactivo, no se puede asegurar que el robot móvil llegue a su destino, sin embargo las redes neuronales son eficientes para la navegación debido a su alta tolerancia al ruido y la imprecisión [28]. Sin embargo al modificar el algoritmo de aprendizaje, este ajusta los pesos de la red y los coeficientes de las neuronas para que la red implemente la función esperada y con ello se tenga la respuesta deseada del sistema [38].

Por lo anterior, en este trabajo se plantea diseñar y simular un algoritmo de navegación basado en aprendizaje por refuerzo utilizando un mapa generado por el algoritmo SLAM, que rescate las características de ambas topologías de navegación, es decir que con un algoritmo ya conocido (SLAM) se genere un mapa, en el que se implemente la técnica de RL con la finalidad de dotar al robot móvil con la capacidad de desplazarse en un entorno estático sin colisionar.

1.3. Justificación

En el campo de la robótica, se han diseñado robots con la finalidad de mejorar la calidad de vida del hombre, por lo que la industria ha sido uno de los sectores en los que ha habido mayor desarrollo, por lo que en los años sesenta se diseñaron robots manipuladores los cuales ejecutaban tareas repetitivas; sin embargo, eran ubicados en un área específica, ya que tenían un espacio de trabajo fijo limitado por la máxima extensión de sus articulaciones [1]. Ante este problema se buscó desarrollar robots móviles, los cuales tuvieran la capacidad de desplazarse sobre rieles para proporcionar un transporte eficaz. Aunque con los años ha surgido la necesitad de dotar de autonomía a estos robots. Con la finalidad de que transporten materia prima y herramientas, o bien sean empleados en sistemas de inspección y monitoreo [35]. Estos se programan para que puedan navegar en un ambiente desconocido.

La implementación de un sistemas de navegación autónoma en un robot móvil experimental permite dar autonomía a agentes para reconstruir y explorar el entono, sin necesidad de contar con condiciones iniciales para su operación. Principalmente este tipo de sistemas son implementados en el transporte automatizado de piezas, robots destinados a evaluar zonas de desastres cuyos entornos son desconocidos para el hombre, o bien son empleados para realizar tareas de vigilancia. Dichos robots tienen la capacidad de desplazarse en entornos que cambian en el tiempo [39]. Así mismo se emplean en áreas de exploración terrestres y acuáticas no tripuladas y para la localización de objetos. Por lo que el desarrollo de sistemas de navegación autónoma en áreas reducidas en las que no se cuenta con un mapa es un tema de interés creciente. Para el desarrollo de sistemas de navegación autónoma, se ha empleado la técnica SLAM para la generación de mapas, lo que permite obtener información de ambientes con incertidumbre [37]. Por otro lado, al emplear RL en un robot móvil se puede navegar en entornos complejos [29].

Por tanto, se busca implementar un sistema de navegación autónoma en un robot móvil, que navegue en el edificio de la División de Estudios de Posgrado, en la Universidad Tecnológica de la Mixteca (UTM), para lo que se necesita obtener el mapa empleando la técnica SLAM y posteriormente implementar RL con la finalidad de que el robot tenga la capacidad de navegar en dicho entorno. Al desarrollar un sistema de navegación, ampliará la línea de investigación que se desarrolla en la UTM. Actualmente existen pocos proyectos en desarrollo sobre técnicas para la navegación autónoma de robots móviles. Ya que, anteriormente los robots móviles autónomos que se han desarrollado principalmente son de arquitectura reactiva. Por este motivo la implementación de algoritmos basados en RL para la navegación en robots móviles proverían a la universidad de estas tecnologías, obteniendo el mapa del entorno con SLAM e implementación de un sistema para la evasión de obstáculos en un robot móvil. Esto permite la exploración y extrapolación de diferentes sistemas robóticos. Por otro lado, el desarrollo de estas técnicas se podrían implementar en investigación y el desarrollo de distintos robots inteligentes tales como robots de servicio, de rescate o exploración.

1.4. Hipótesis

Utilizando un algoritmo de navegación basado en aprendizaje por refuerzo, un robot móvil en configuración diferencial podrá navegar de manera segura en un entorno estático.

1.5. Objetivo general

Diseñar y simular un algoritmo basado en aprendizaje por refuerzo y un mapa generado por SLAM para la navegación autónoma de un robot móvil en configuración diferencial.

1.5.1. Objetivos específicos

- Simular un control cinemático para el correcto movimiento de un robot en el entorno virtual.
- Realizar el entorno virtual 3D.
- Generar el mapa del entorno con el algoritmo Gmapping-SLAM.
- Diseñar un algoritmo de navegación basado en RL empleando el mapa obtenido como referencia.
- Simular el algoritmo de navegación basado en RL .

1.6. Limitantes

- Se utilizará un robot en configuración diferencial.
- Se hará uso del modelo cinemático en un punto de operación fuera del eje de acción de las ruedas para el control.
- Se empleará un sensor LiDAR 2D con un alcance de detección de 5m.
- Se utilizará el algoritmo de acceso libre Gmapping-SLAM en un entorno virtual para la generación del mapa.
- Los obstáculos que se presenten en el entorno serán estáticos.
- El sistema de navegación será probado en entornos virtuales.
- El área de trabajo del robot será la planta baja del edificio de “División de estudios de posgrado” de la Universidad Tecnológica de la Mixteca.

1.7. Metodología

En el 2021 [40] Pérez presenta una metodología para el diseño de algoritmos y en trabajos relacionados con este proyecto, Gongora en el 2015 [41] presenta una metodología en su trabajo sobre algoritmos SLAM, sin embargo, estas metodologías no abarcan por completo los alcances de este proyecto, por lo que en la Figura 1.5 se presenta la metodología propuesta para la elaboración de este proyecto, basándose principalmente en el control del robot y la implementación del algoritmo SLAM para obtener el mapa, y posteriormente implementar un algoritmo de navegación.

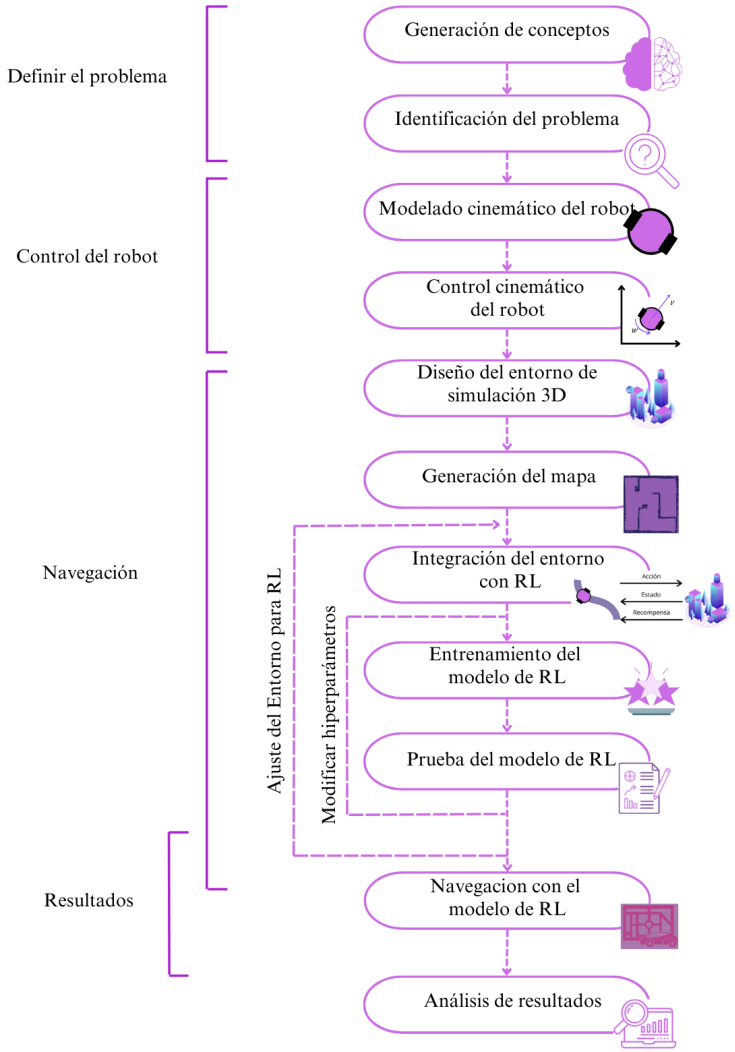


Figura 1.5: Metodología propuesta.

La etapa inicial implica la recopilación de información sobre trabajos relacionados y la adquisición de conocimientos básicos sobre el tema de interés.

- *Generación de conceptos:* Investigar los conceptos y antecedentes para generar un marco teórico que permita identificar cuáles serán los aspectos necesarios para realizar este proyecto y las limitantes.
- *Identificación del problema:* Al analizar los antecedentes y establecer las limitaciones del proyecto se establece una hipótesis así como el enfoque del proyecto, con la finalidad de establecer los requerimientos que debe de cumplir el algoritmo a diseñar.

En la fase de control, se desarrollan las siguientes etapas con el propósito de trasladar el robot desde un punto inicial hasta un punto final, controlando la posición y velocidad.

- *Modelado del robot:* Se realizará el modelo cinemático para determinar la posición y orientación del robot con respecto a una referencia.
- *Control del robot:* Una vez obtenido el modelo cinemático se realizará el control cinemático para llegar a un punto deseado mediante las velocidades de las ruedas.

La Navegación se realizará con el algoritmo de RL, para ello se necesita diseñar el entorno, obtener el mapa, e integrar el agente (en este caso el *Turtlebot 3*) con el entorno de simulación.

- *Diseño del entorno de simulación 3D:* En el entorno 3D se deben integrar los obstáculos que se presentan en la planta baja del edificio de la División de Estudios de Posgrado.
- *Generación del mapa (Algoritmo SLAM):* Implementar a nivel simulación el algoritmo Gmapping-SLAM en el robot móvil, con la finalidad de obtener el mapa.
- *Integración del entorno con RL:* Para entrenar el modelo de RL se necesita comunicar el agente y así obtener información del entorno.
- *Entrenamiento del modelo de RL:* Una vez que se vinculo el entorno con el agente, se integra el modelo de RL y se modifican los hiperparámetros. Hasta obtener el resultado deseado.
- *Prueba del modelo de RL:* AL obtener el modelo, se realizan pruebas de su funcionamiento.
- *Navegación con el modelo de RL*: Se integran el mapa obtenido con SLAM y el modelo de RL, para que robot navegue en el entorno.

Finalmente se realizará la etapa de evaluación de resultados.

- *Análisis de resultados:* Realizar un análisis de resultados de acuerdo al desempeño del robot.

Capítulo 2

Marco Teórico

En este capítulo se presenta información sobre tipos de mapas y sistemas de planificación de rutas, los cuales se emplean para la navegación. Posteriormente se presenta información sobre el Sistema Operativo Robótico (ROS, por sus siglas en inglés) el cual es empleado para el desarrollo de software en robótica. Para esta tesis, se utilizará SLAM que es un algoritmo para estimar la posición del robot y reconstruir la estructura en un entorno desconocido, así mismo se aborda información sobre el RL, el cual dota al robot de la capacidad de adquirir de manera autónoma comportamientos óptimos mediante interacciones con su entorno.

2.1. Tipos de mapas

Un algoritmo de SLAM tiene la capacidad de representar el entorno o mapa de su ambiente de diversas formas, cada tipo de mapa tiene sus ventajas y desventajas. Al momento de elegir un tipo de mapa a implementar, se debe tener en cuenta si es necesario tener nociones métricas, el uso, si el entorno es dinámico, los sensores a emplear y el poder de cómputo. Existen diferentes tipos de mapas:

2.1.1. Mapas topológicos

Los mapas topológicos son los que se crean a partir de los puntos en los que ha estado el robot y la relación existente entre los mismos de forma compacta con la finalidad de representar modelos gráficos del mismo [42]. Sin embargo, este tipo de mapas tienen complicaciones en el "*aliasing*" que ocurre cuando hay elementos en el entorno que parecen ser iguales para el robot móvil, esto ocurre cuando se presenta ruido sensorial, campo de visión limitado y estructuras repetidas [43].

2.1.2. Mapa de rejilla probabilística

El mapa de rejilla probabilística se basa en discretizar el espacio, dividiéndolo en unidades de tamaño predefinido, que se clasifican como ocupadas o vacías con un determinado nivel de confianza o probabilidad, por lo que la probabilidad de ocupación se incrementa si se detecta un obstáculo en cierta celda de la rejilla. El modelo de rejilla de ocupación se basa en la probabilidad de ocupación de las celdas, lo que representa un obstáculo, así mismo los sensores tienen un modelo probabilístico, facilitando la fusión sensorial. Debido a la discretización del entorno en celdas sólo se recomienda para ambientes de interiores y estructurados. [44].

2.1.3. Mapa métrico probabilístico

En el mapa métrico probabilístico dividido en celdas, la probabilidad de ocupación, es cuando se obtiene información indirecta sobre la ubicación de los objetos detectados, a partir de los cuales se obtienen funciones de densidad de probabilidad, de acuerdo a las características del sensor, si un objeto se encuentra en la celda, la probabilidad de ocupación es cercana a uno; si la celda está vacía, la probabilidad de ocupación es cercana a cero. [45, 46].

2.2. Sistemas de planificación de rutas.

Los sistemas de planificación de rutas (o caminos) tienen como propósito calcular un control de movimiento que evite colisiones, mientras el robot se desplaza hacia la ubicación de destino, obteniendo como resultado una secuencia de movimientos que impulsan al robot a un camino libre de colisiones al objetivo [47]. Existen algoritmos basados en mapas y algoritmos no basados en mapas.

- Basados en mapas
  - Descomposición trapezoidal
  - Exploración Rápida de Árbol Aleatoria (RRT, por sus siglas en inglés)
  - Hoja de Ruta Probabilística (PRM, por sus siglas en inglés)
  - Diagramas de Voronoi
- No basados en mapas
  - Campos potenciales
  - Tipo Bicho

2.2.1. Basados en mapas

Descomposición trapezoidal

La descomposición trapezoidal está relacionada con la representación poligonal del espacio, este se representa en celdas bidimensionales que tienen forma de trapezoides, algunas celdas pueden tener forma de triángulos, que pueden verse como trapecios deformados donde uno de los lados paralelos tiene un borde de longitud cero. Para el espacio de configuración planar, el espacio libre y los obstáculos están delimitados por polígonos. Tiene la característica de requerir mayor complejidad computacional que otras técnicas, sin embargo presenta mayor exactitud y completitud. En la Figura 2.1, tomada de [8], se detalla un ejemplo de descomposición trapezoidal con dos obstáculos y las posiciones de inicio y de objetivo final [8, 48].

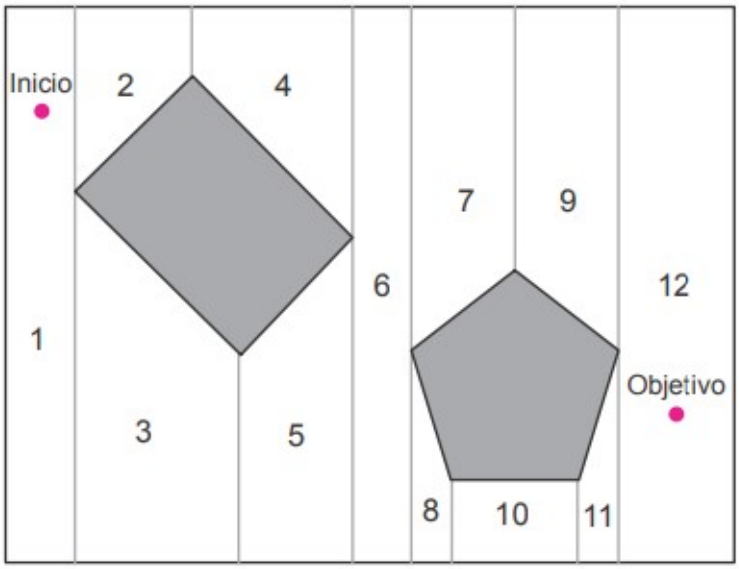


Figura 2.1: Espacio de configuración descompuesto con trapezoides.

Exploración Rápida de Árbol Aleatoria (RRT, por sus siglas en inglés)

El algoritmo de RRT se emplea para resolver el problema de planificación de movimiento, basándose en el muestreo aleatorio de un espacio de configuraciones que están conectadas a una estructura de árbol en el que se puede encontrar la ruta óptima. Se puede dividir en tres partes: selección de un vértice, expansión y condición de terminación. El rendimiento del algoritmo RRT puede ser deficiente en entornos que contienen pasajes estrechos ya que atrofian el proceso de crecimiento del árbol, lo que ralentiza la búsqueda de resultados [49].

Hoja de Ruta Probabilística (PRM, por sus siglas en inglés)

El algoritmo de PRM se basa en el muestreo propuesto por Kavraki, en 1996. Este algoritmo consiste en una primera fase de aprendizaje que se basa en un gráfico de red que consiste en nodos y conexiones, en un área libre de colisiones, (Figura 2.2 a, Modificada de [11]). La segunda fase, es de consulta en la cual se buscan los mejores nodos conectados que van del inicio y a la posición del objetivo, basado en el gráfico de red de la Figura 2.2 a), la ruta final generada por PRM se muestra en la Figura 2.2 b).
Existen factores que afectan el rendimiento de PRM, como la cantidad de nodos, la distancia de conexión, etc. El algoritmo solo calcula la ruta más corta, sin embargo, no tiene la capacidad de cambiar los nodos según la seguridad o la suavidad del camino [11].

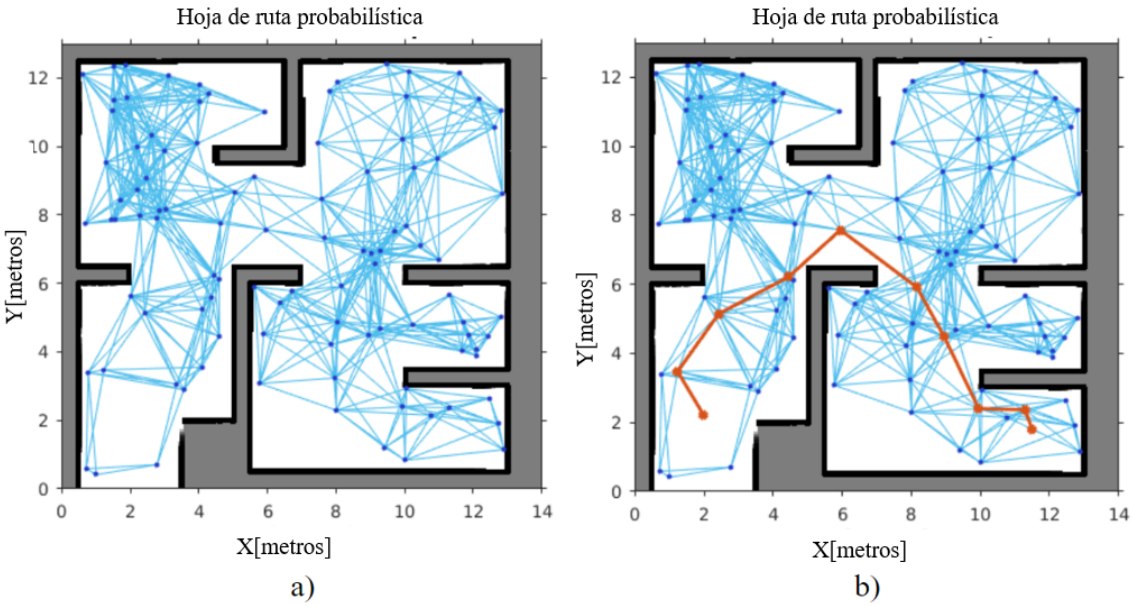


Figura 2.2: a) Ejemplo del grafo b) Ruta final generada.

Diagramas de Voronoi

Los diagramas de Voronoi son parte de las estructuras de planificación de rutas más importantes en geometría computacional, los cuales parten de la comparación entre elementos cercanos en un plano. Al realizar una partición del espacio en un conjunto de regiones se hace una ponderación de los puntos, considerando regiones asociadas con subconjuntos de puntos en lugar de puntos individuales o con conjuntos de características geométricas distintas de los puntos. Dado un conjunto finito de puntos distintos en el espacio euclidiano, se asocian todas las ubicaciones en ese espacio con el miembro más cercano del conjunto de puntos. [9, 50]

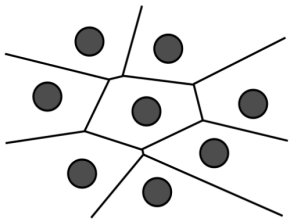


Figura 2.3: Diagrama de Voronoi.

En este método es fundamental considerar el recorrido lo más lejos posible de los obstáculos, en un espacio bidimensional, todos los puntos que son equidistantes de dos barreras se consideran parte del diagrama de Voronoi mostrado en la Figura 2.3, extraida de [9].

2.2.2. Basados en mapas

Campos potenciales

La técnica de Campos Potenciales consiste en calcular campos imaginarios de repulsión que emanan de los obstáculos, estos varían de acuerdo a la distancia del obstáculo, también se imponen límites de influencia de éstos para no calcular los campos de los obstáculos lejanos, este proceso consiste en encontrar un camino que se mantenga tan alejado de los obstáculos como sea posible, en la Figura 2.4 (Modificado de [51]) se ejemplifican los campos generados por los obstáculos presentes. [51]

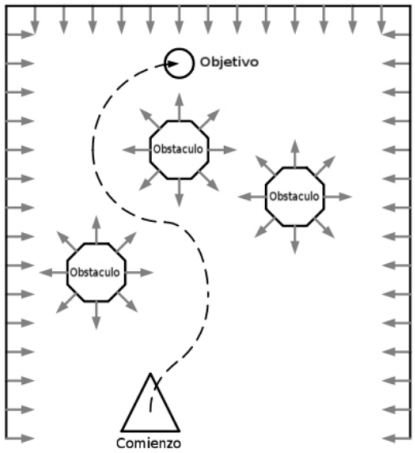


Figura 2.4: Campos potenciales generados en una superficie.

Tipo Bicho

Los algoritmos Tipo Bicho son planificadores de ruta simples con garantías demostrables. Su propósito es producir una ruta libre de colisiones siguiendo límites y moviéndose hacia la meta. Cuando se el robot se encuentra con un obstáculo, el algoritmo puede generar una ruta contorneando el objeto en la superficie 2D si existe una ruta hacia la meta. Se tienen que tomar en cuenta tres suposiciones:

- El robot es considerado un punto
- Se tiene información correcta sobre la localización
- Los sensores no obtienen lecturas erróneas

Si estos criterios no se cumplen el algoritmo falla, por lo que se dificulta la implementación en robots móviles reales, sin embargo, en los programas de simulación se puede implementar ya que se cumple con estas condiciones [52].

Existen 2 tipos de algoritmos Tipo Bicho (Figura 2.5, modificada de [52]). En el Tipo 1, el robot móvil va directamente hacia la meta a lo largo de una línea que conecta el inicio con la meta, a menos que encuentre un obstáculo, como se muestra en la Figura 2.5 (Tipo 1), rodea el obstáculo en sentido horario (por defecto) y determina el punto de salida calculando la distancia entre la posición actual y la meta, durante el recorrido alrededor del objeto. El Tipo 2, sigue una pendiente constante calculada inicialmente entre el inicio y la meta. Cuando aparece un obstáculo, sigue los bordes del obstáculo utilizando sus sensores de sonar en sentido horario hasta que encuentra su pendiente inicial nuevamente (Figura 2.5 (Tipo 2)) [52].

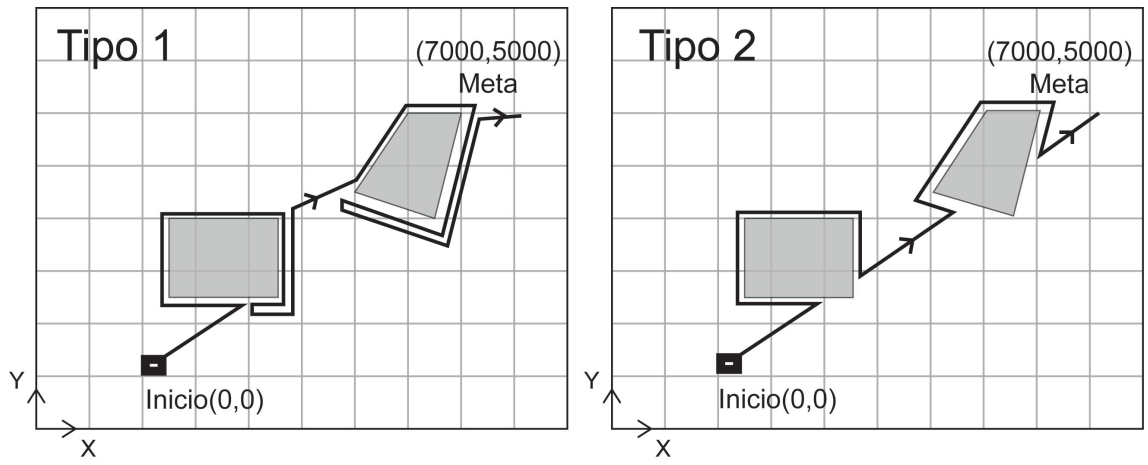


Figura 2.5: Algoritmo Tipo Bicho.

2.3. Algoritmos de optimización de rutas

Los algoritmos para la optimización de rutas tienen como propósito explorar un espacios a partir de poca o ninguna información sobre el entorno con el fin de encontrar una ruta que permita ir de una posición inicial a una posición final.

2.3.1. Algoritmo Dijkstra

El algoritmo Dijkstra es el más simple para encontrar la ruta más corta entre un nodo y los demás que pertenecen a un grafo. Es un grafo dirigido que sirve para encontrar la ruta más corta entre dos nodos, uno inicial o uno final. Todos los nodos están etiquetados con números, al principio todos tienen la etiqueta "infinito" excepto el nodo inicial que tiene etiqueta 0. Los arcos tienen un peso que representa la distancia del enlace, el algoritmo Dijkstra genera una lista con todos los nodos del grafo, el nodo seleccionado se marca y es extraído de la lista inicial y se adjunta a la lista de nodos marcados, posteriormente se actualizan las etiquetas de los sucesores del nodo seleccionado [53].

2.3.2. A* (A-Estrella)

El algoritmo A* (A-Estrella) fue creado en 1968 para combinar el enfoque heurístico del algoritmo de Búsqueda en Anchura (BFS, por sus siglas en ingles) con el método de optimización del algoritmo de Dijkstra. Es el único algoritmo heurístico que determina el camino más corto entre dos nodos específicos y se utiliza ampliamente en navegación, donde la rapidez de respuesta es fundamental. Su funcionamiento se basa en dos conjuntos: "Disponibles", que incluye los nodos candidatos a ser examinados (comenzando solo con el nodo inicial), y "Analizados", que contiene los nodos ya revisados y que forman parte de la ruta elegida (inicialmente vacío).

Aunque es similar al algoritmo de Dijkstra, A* se distingue por la inclusión de una función heurística. Si esta función heurística es cero (no se utiliza), el algoritmo se comporta exactamente como Dijkstra. La heurística modifica el criterio para seleccionar el nodo candidato a marcar. Aunque A* no siempre garantiza el camino óptimo, se mantiene cerca de la solución ideal. Su efectividad depende completamente de la función heurística utilizada. A* rinde mejor en términos de velocidad y optimalidad cuando la heurística indica el costo de llegar desde un nodo dado hasta el nodo destino.

2.4. Localización y Mapeo Simultáneo (SLAM)

En el ámbito de la navegación con Simultaneous Localization and Mapping (SLAM) el movimiento de robots implica dos aspectos clave: el mapeo, que se refiere a la creación de una representación del entorno, y la localización, que implica determinar la posición del robot en ese entorno [54].

Existen diversas técnicas para abordar estos desafíos. Por un lado, en la navegación con conocimiento del mapa pero sin información precisa sobre la localización, se emplean herramientas como el Filtro de Kalman y el Filtro de Partículas. Estos métodos permiten al robot moverse en un entorno conocido, ajustando su posición a medida que se desplaza. Por otro lado, en la navegación con conocimiento de la localización pero sin información sobre el mapa, se utiliza la técnica de Mapa de Rejilla de Ocupación (Occupancy Grid Map). Este enfoque permite al robot moverse con conocimiento de su posición sin tener un mapa detallado del entorno. Finalmente, en situaciones donde no hay conocimiento ni del mapa ni de la localización, se recurre a SLAM. [54, 55].

2.4.1. Filtro de Kalman

El filtro de Kalman estima los estados de un sistema lineal. Este filtro minimiza la varianza del error de estimación. Los filtros de Kalman se implementan a menudo en sistemas de control integrados, ya que para el control se necesita una estimación precisa de las variables. Se desea que el valor promedio de la estimación de estado sea igual al valor promedio del estado real, para evitar que esté sesgada la estimación en una dirección u otra. Así mismo se desea encontrar el estimador con la menor varianza de error posible [56].

2.4.2. Filtro de Partículas

El filtro de partículas tiene como objetivo determinar la ubicación del robot en un entorno. Se trata de un filtro bayesiano que se fundamenta en el método de Monte Carlo, ofreciendo una aproximación precisa de expresiones matemáticas mediante la generación de números pseudoaleatorios. Además, asume que el proceso sigue una cadena de Markov, una herramienta de predicción de comportamientos futuros basada en valores históricos o anteriores del sistema [57, 58].

La inicialización del filtro de partículas implica la generación de un conjunto de partículas con pesos uniformes. Una vez inicializado, se procede a predecir los estados de las partículas, calcular las estimaciones observadas y actualizar los pesos, normalizándolos [58, 59]. Se procede a una etapa de predicción y en la cual el conjunto de muestras V_{t-1} se evalúa en el modelo del sistema, generando $V_t^* = f(v_t(i), \delta_{t-1}(i))$, con el objetivo de obtener muestras a priori en el tiempo t. Posteriormente se pasa a una etapa de actualización en la cual se

realizan mediciones del entorno con los sensores w_t, y se calculan los pesos de cada estado $Q_t = \{q_1, q_2, \ldots, q_M\}$, utilizando la ecuación 2.1:

$$q_i = \frac{p(w_t|v_t^*(i))}{\sum_{j=1}^{M} p(w_t|v_t^*(j))} \tag{2.1}$$

Al llegar a la etapa de actualización, se obtiene una nueva función de probabilidad a priori basada en los pesos de importancia q_i, la cual es remuestreada para obtener un nuevo conjunto de datos $V^K = \{v_k(i) : i = 1, 2, ..., M\}$ que está distribuido aproximadamente como $p(v_k|w_k)$ [59].

Este proceso implica la conservación o el descarte de partículas, manteniendo un número constante y ajustando los pesos. Estos pasos se repiten iterativamente hasta completar el proceso de filtrado. El remuestreo contribuye a reducir la varianza y enfocar las partículas en regiones más relevantes del espacio de estados. A diferencia de los filtros de Kalman, que se basan en distribuciones normales, los filtros de partículas pueden aproximar una amplia gama de distribuciones de probabilidad [58–60].

2.4.3. Mapa de Rejilla de Ocupación (Occupancy Grid Map)

La representación a través de Mapas de Rejilla de Ocupación divide el entorno en celdas ocupadas o desocupadas, brindando al robot una aproximación detallada del entorno. Aunque estos mapas no logran una precisión absoluta, la elección de un tamaño de celda reducido facilita la obtención de datos para la generación del mapa.

Esta técnica utiliza sensores para informar sobre el estado de un conjunto de celdas de la cuadrícula sin requerir referencia al resto del mapa. Sin embargo, su implementación, como la propuesta por Moravec en 1988, no aborda de manera exhaustiva la localización precisa del robot. En este enfoque, el mapa se mantiene en relación con el robot en lugar de en un marco de referencia global, lo que genera inconvenientes cuando el robot se desplaza, provocando que se difumine el mapa. A pesar de posibilitar una representación detallada del entorno, esta técnica restringe la aplicación de ciertos algoritmos debido a la abundancia de características en el mapa, con una por cada celda, lo que afecta las técnicas de localización y cartografía disponibles [55].

2.4.4. SLAM

El método Localización y Mapeo Simultáneo (SLAM, por sus siglas en inglés) se basa en la posibilidad de ubicar a un robot en un ambiente desconocido para este, por lo que debe de ser capaz de moverse en dicho entorno mientras se realiza el mapeo o reconstrucción espacial del lugar y simultáneamente calcula su localización, de acuerdo al marco de referencia global

que representa el mapa en sí mismo [41].
El proceso de SLAM consiste en la extracción de características, asociación de datos, estimación del estado y actualización de las características. Para ello se usa el entorno para actualizar la posición del robot. Dado que la odometría (la cual es la estimación de la posición) del robot no es completamente fiable, no se puede depender directamente de ella, por lo que se emplean sensores de distancia, para corregir el error de posición, por lo que se pueden emplear filtros para reducir los errores de odometría y de los sensores [41].

Los pasos del SLAM son los siguientes:

a. Movimiento del robot, al tener un modelo del robot se puede estimar su posición.

b. Corregir la estimación tras la re-observación de los puntos de referencia para conocer su posición. Usando la estimación de la posición del paso previo es posible estimar dónde deberían estar ubicados, esta diferencia de información es la que utiliza el robot para estimar el estado.

c. Finalmente, se reinicia el ciclo al incorporar nuevos puntos de referencia al mapa del robot. Este proceso se realiza utilizando la información actualizada de la posición e incertidumbre recién obtenidas.

2.5. ROS

El Sistema Operativo Robótico (ROS, por sus siglas en ingles) el cual es un sistema de código abierto que cuenta con un conjunto de bibliotecas de software y herramientas que permiten crear aplicaciones robóticas, controladores, algoritmos de última generación y herramientas de desarrollo [61]. Puede considerarse el marco estándar para la investigación y desarrollo de tecnologías robóticas. ROS aborda la necesidad de comunicación entre los diversos procesos de un sistema robótico [62].

2.5.1. Arquitectura de Comunicación

Nodos

Los nodos de [61] son procesos que utilizan las interfaces de programación de aplicaciones (APIs, por sus siglas en inglés) para comunicarse. Un robot puede tener muchos nodos para realizar sus cálculos. Por ejemplo, un robot móvil autónomo puede tener un nodo para el hardware, para la interfaz, la lectura de sensores, localización y mapeo [61].

Nodo Maestro

El nodo maestro funciona como un intermediario que ayuda a las conexiones entre diferentes nodos [61], tiene los detalles sobre todos los nodos que se ejecutan en el entorno [61], intercambia detalles de un nodo con otro para establecer una conexión y comunicación entre ellos [61] .

Servidor de parámetros

El servidor de parámetros es útil en [61], ya que un nodo puede almacenar una variable en el servidor de parámetros y establecer su privacidad también. Si el parámetro tiene un alcance global, todos los demás nodos pueden acceder a él [61].

Mensajes

Los nodos pueden comunicarse entre sí de muchas maneras, En todos los métodos, los nodos envían y reciben datos en forma de mensajes. El mensaje es una estructura de datos utilizada para intercambiar datos [61].

Tópicos

Es un método para comunicarse e intercambiar mensajes entre dos nodos son canales de comunicación con nombre, en los que los datos se intercambian mediante mensajes. Cada tópico tendrá un nombre específico y un nodo publicará datos y otro puede leer del tópico suscribiéndose a él [61].

Servicios

Los servicios son otro tipo de método de comunicación, como los tópicos, en los servicios, se emplea un método de solicitud o respuesta. Un nodo actuará como proveedor de servicios, que tiene una rutina de servicio en ejecución, y un nodo cliente solicita un servicio. El servidor ejecuta la rutina de servicio y envía el resultado al cliente, el nodo cliente debe esperar hasta que el servidor responda con los resultados [63].

Acciones

Permiten la comunicación entre un cliente y un servidor a través de mensajes ROS, proporcionando una API simple para solicitar y ejecutar metas. Para esta comunicación, se definen mensajes específicos como la Meta, Retroalimentación y Resultado, que permiten al cliente hacer solicitudes y recibir información continua sobre el progreso y el resultado final de la acción [61].

2.5.2. Robot Turtlebot

El robot *Turtlebot* es un robot móvil de cinemática diferencial el cual es empleado para investigación y educación. [64]. Tiene como soporte [61], el cual es un conjunto de bibliotecas de software y herramientas que permiten crear aplicaciones robóticas, controladores, algoritmos de última generación y herramientas de desarrollo, es un sistema de código abierto [61].

2.6. Aprendizaje Automático

Casi todas las piezas de software se han programado explícitamente, eso significa que se establecieron un conjunto de reglas para los dispositivos. La programación explícita es la columna vertebral de la programación, es ideal para administrar datos, calcular un valor o hacer un seguimiento de las relaciones. Sin embargo cuando se requieren hacer cosas complejas, como el reconocimiento de una imagen, para enseñarle a una computadora qué buscar se tendría que escribir un código para cada contingencia [65].

Con la programación explícita se definen las etiquetas y se tiene en cuenta cada contingencia en el código. En contraste, los algoritmos de aprendizaje automático permiten a la computadora descubrir las etiquetas. El algoritmo puede contener algunas características clave, se ingresa una base de datos, con diversos elementos, los cuales no están etiquetados, cuando el algoritmo hace una suposición, se refuerzan las suposiciones correctas y se dan recompensas negativas por las suposiciones incorrectas [65].

2.6.1. Inteligencia Artificial

La Inteligencia Artificial (IA), fue acuñada a mediados del siglo XX, se refiere a cualquier momento en que una máquina observa y responde a su entorno [65]. En 1968 Minsky la definió como la ciencia que busca la creación de máquinas que logren realizar acciones que requerirían inteligencia como si las hiciesen los humanos. En el 2006 por Pajares y Santos, la definen como una máquina inteligente que realiza el proceso de analizar, organizar, y convertir los datos en conocimiento, el cual es información estructurada adquirida y aplicada para reducir la incertidumbre sobre una tarea específica a realizar por esta [65].

2.6.2. Redes neuronales

Las *Redes Neuronales (RN)*, Figura 2.6, modificada de [66], están constituidas por elementos que se comportan de forma similar a la neurona biológica en sus funciones más comunes y están organizados de una forma parecida a la que presenta el cerebro humano [66]. Es un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles, posteriormente se traslada a un sistema de computación compuesto por un gran número de elementos simples de procesos interconectados a partir de los cuales se procesa

la información por medio de un estado dinámico como respuesta a entradas externas. Son redes interconectadas en paralelo de elementos simples (usualmente adaptativos) organizados jerárquicamente, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hacen las neuronas humanas [67].

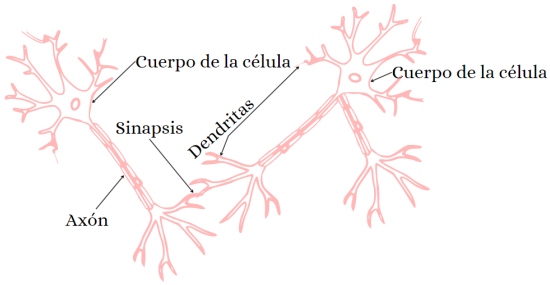


Figura 2.6: Componentes de una neurona.

2.6.3. Agente

Un agente inteligente es una entidad física o virtual capaz de percibir su ambiente mediante sensores y tener una representación parcial del mismo; es capaz de actuar sobre el entorno mediante actuadores; puede comunicarse con otros agentes, estos pueden ser humanos o no; tiene un conjunto de objetivos que se que se realizan mediante una acción autónoma y flexible que gobiernan su comportamiento y posee recursos propios [68].

2.6.4. Aprendizaje

El concepto de aprendizaje en informática, denominado también aprendizaje de máquina o automático, corresponde a programas computacionales que buscan optimizar los parámetros de un modelo usando datos de entrenamiento. Los modelos pueden ser inductivos, cuando permiten hacer predicciones sobre el futuro o bien descriptivos cuando permiten generar conocimiento a partir de los datos [68].

2.6.5. Aprendizaje supervisado

El aprendizaje supervisado implica una entrada y salida para cada pieza de datos en su conjunto de datos, el algoritmo necesita un conjunto de datos de capacitación etiquetado con las respuestas correctas para poder aprender. Esas etiquetas actúan como un maestro que supervisa el aprendizaje. A medida que el algoritmo predice si un resultado, las etiquetas ayudarán al modelo a afinarse, ver Figura 2.7, tomada de [69] . El modelo deja de aprender cuando alcanza un nivel aceptable de precisión o se queda sin datos de entrenamiento etiquetados. El aprendizaje supervisado es ideal para tareas donde el modelo necesita predecir resultados [65].

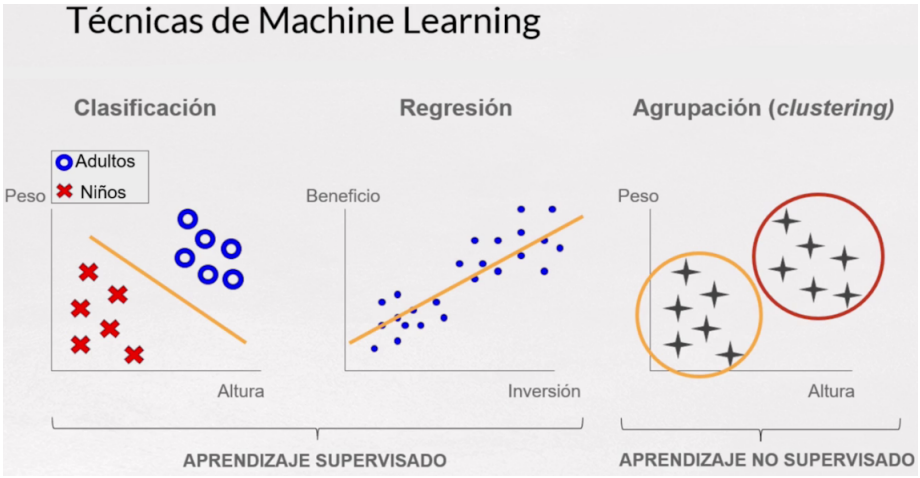


Figura 2.7: Aprendizaje supervisado y no supervisado.

2.6.6. Aprendizaje no supervisado

Aprendizaje no supervisado consta de un conjunto de datos que no tiene etiquetas con una respuesta correcta, por lo que el algoritmo saca sus propias conclusiones comparando los datos consigo mismo, como se muestra en la Figura 2.7. El objetivo es descubrir algo sobre la estructura o distribución subyacente del conjunto de datos, se usa para problemas de agrupamiento, donde los datos deben organizarse en grupos similares o problemas de asociación para averiguar qué variables se correlacionan entre sí [65].

2.6.7. Aprendizaje por refuerzo

El Aprendizaje por Refuerzo (RL, por sus siglas en inglés), es una técnica basada en realizar tareas mediante recompensas, para ello el agente tiene en consideración el estado del entorno y las recompensas obtenidas previamente, selecciona la acción con la cual estima que va a obtener una mayor recompensa [70].

Las acciones producen cambios en el entorno, por lo que se observa el efecto de la acción en el entorno, el agente obtiene la actualización del estado del entorno y la información de la recompensa, cerrando así un lazo de control como se muestra en la Figura 2.8, extraída de [71]. Si se compara con un lazo de control tradicional, el entorno es equivalente a la planta, el agente es el controlador y el actuador, y el intérprete es similar a un sensor o un observador. La relación entre el estado actual y la acción se suele denominar política, esta relación puede ser discreta o continua, dependiendo de los estados y el conjunto de acciones [38].

Con ello, el agente tiene la capacidad de adquirir de manera autónoma acciones óptimas mediante interacciones con su entorno, evitando así la necesidad de usar soluciones explici-

tas a problemas particulares. En lugar de ello, se incorpora conocimiento a través de una función que evalúa el rendimiento del robot tras cada acción. Calcula rutas óptimas para cada destino, facilitando la evitación óptima de obstáculos [72].

Figura 2.8: Aprendizaje por refuerzo.

2.6.8. Q-Learning

El Q-learning es un algoritmo de aprendizaje por refuerzo sin modelo, puede verse como un método de programación dinámica asíncrona (DP), esto permite a los agentes aprender a actuar de manera óptima en entornos de dominios markovianos, sin necesidad de un modelo explícito del entorno. El objetivo del agente es maximizar la recompensa futura acumulada, lo que se logra mediante la aproximación de la función *valor-acción Q(s,a)*, que representa la recompensa máxima esperada al tomar la acción a en el estado s. La cual se define como:

$$Q(s,a) \approx \max_{p} \left(r_t + cr_{t+1} + c^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, p\right) \tag{2.2}$$

donde: - $Q(s,a)$ es la función que representa la recompensa futura máxima alcanzable a partir del estado s al tomar la acción a, - r_t es la recompensa obtenida en el paso de tiempo t, - c es el factor de descuento, - p es la política de comportamiento.
El algoritmo puede ser inestable cuando se usan aproximadores no lineales, como redes neuronales, debido a correlaciones en las secuencias de observaciones y actualizaciones. Para mitigar estas inestabilidades, se emplean dos técnicas: experiencia de repetición (que aleatoriza las experiencias) y actualizaciones periódicas de los objetivos (para reducir las correlaciones con los valores de Q).

2.7. Algoritmos de RL

2.7.1. DDPG

El algoritmo de Gradiente de Política Determinista Profundo (DDPG, por sus siglas en inglés) se distingue por su capacidad para aprender de manera simultánea funciones Q y políticas en entornos con espacios de acción continuos. Su enfoque eficiente se manifiesta en la aproximación de $\max_a Q^*(s,a)$ mediante $\max_a Q(s,a) \approx Q(s,\mu(s))$, evitando así subrutinas de optimización costosas. Entre sus características notables, destaca su operación fuera de política, su adaptación específica a espacios de acción continuos, la conceptualización como un *Q-learning* profundo diseñado para acciones continuas, y la ausencia de soporte para paralelización en la implementación de DDPG en *Spinning Up*. Este algoritmo representa una herramienta valiosa para abordar problemas en entornos complejos y continuos, ofreciendo soluciones eficaces en la interacción entre aprendizaje de políticas y funciones Q.

2.7.2. TD3

El Gradiente de Política Determinista con Retraso Gemelo (TD3, por sus siglas en inglés en su tercera versión) representa una mejora sobre el algoritmo DDPG. Su objetivo es abordar la sobreestimación de valores Q y la fragilidad en la configuración de hiperparámetros. Las tres estrategias clave de TD3 incluyen el uso de dos funciones Q gemelas y la selección del mínimo para mitigar la sobreestimación, la actualización menos frecuente de la política en comparación con las funciones Q para estabilizar el aprendizaje, y la adición de ruido a las acciones objetivo para prevenir la explotación de errores en la función Q. Estos ajustes conducen a un rendimiento notablemente mejorado en comparación con el DDPG convencional.

2.8. Aplicación de la política TD3 en un robot diferencial

El entorno de simulación y el algoritmo de RL desarrollado por Cimurs et al. [73], se diseñó para un robot *Pionner*. Utiliza una red neuronal basada en arquitectura TD3 para entrenar la política de movimiento. Es una red de actor-crítico que permite realizar acciones en un espacio de acción continuo.

La entrada al actor-red es el estado s, que incluye las lecturas del entorno (ver Figura 2.9), el ángulo hacia la meta desde la posición del robot, y los dos parámetros de acción *'a'*, que representan la velocidad lineal a_1 y la velocidad angular a_2 del robot. El sistema se compone de dos capas completamente conectadas, cada una seguida de una función de activación *ReLU*. La capa final se vincula a la capa de salida, que contiene los parámetros de acción *'a'*.

Se aplica la función de activación tangente hiperbólica ($tanh$) en la salida para restringirla dentro del intervalo (-1, 1).

Debido a que el robot tiene una velocidad lineal máxima (v_{max}), y una velocidad angular máxima (ω_{max}), Antes de ejecutar la acción en el entorno, se escala con la Ecuación (2.3), en la cual se considera que el robot no puede ir hacia atrás, por lo que la velocidad lineal es positiva:

$$a = \left[v_{max} \left(\frac{a_1 + 1}{2} \right), \omega_{max} a2 \right] \tag{2.3}$$

El valor Q de estado-accion $Q(s, a)$ es evaluado por dos redes críticas. Estas redes utilizan un par de estado y acción como entrada, donde el estado s es procesado por una capa totalmente conectada seguida de una activación ReLU, y la acción a se introduce directamente. Además, la salida de esta capa y la acción son alimentadas a dos capas de transformación totalmente conectadas de igual tamaño τ_1 y τ_2, respectivamente. Finalmente, estas capas se combinan para obtener el resultado deseado. La combinación de las capaz se define como:

$$L_c = L_s W_{\tau_1} + a W_{\tau_2} + b_{\tau_2} \tag{2.4}$$

Donde L_c representa la capa totalmente conectada combinada, W_{τ_1} y W_{τ_2} son los pesos asociados a τ_1 y τ_2, b_{τ_2} es el sesgo correspondiente a la capa τ_2. Luego de aplicar la activación $ReLU$, se conecta a la salida con un parámetro que representa el valor de Q. La salida final de la parte crítica se determina eligiendo el valor mínimo de Q proveniente de ambas redes críticas, esto evita una sobreestimación del valor de la pareja estado-acción. El diseño completo de la red se puede apreciar en la Figura 2.9.

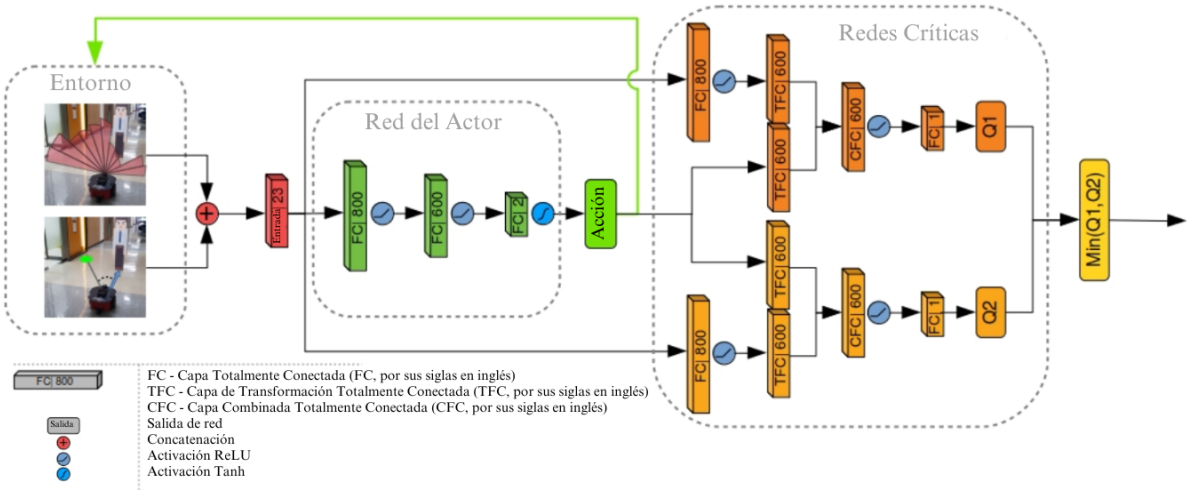


Figura 2.9: Estructura de red TD3.

La política recibe recompensas de acuerdo a la siguiente función:

$$r(s,a) = \begin{cases} r_m & \text{si } D_m < \eta_D \\ r_c & \text{si } collision \\ v - |\omega| & \text{en otro caso} \end{cases} \tag{2.5}$$

Donde:
r_m = recompensa a la meta
r_c = recompensa por colisión
D_m = Distancia a la meta
v = Velocidad lineal
$|\omega|$ = Valor absoluto de la velocidad angular
η_D = Umbral a la meta
La recompensa asociada al par estado-acción (s_t, a_t) en un tiempo t, se determina por tres condiciones diferentes.

- Si la distancia al objetivo en el paso actual, D_m, es menor que un umbral definido η_D, se otorga una recompensa positiva r_m por alcanzar el objetivo.
- En caso de colisionar, se asigna una penalización negativa, r_c.
- Si no se cumplen ninguna de estas condiciones, se otorga una recompensa inmediata basada en la velocidad lineal v y la velocidad angular ω en ese instante.

Para estimar la política de navegación que llegue al objetivo establecido, se emplea un método de recompensa atribuida de manera diferida. Este método implica que la recompensa positiva por alcanzar el objetivo se atribuye no solo al estado y acción que llevaron a su consecución, sino también de forma gradual a lo largo de los n pasos anteriores. Por lo tanto, la red aprende una política de navegación local capaz de llegar a un objetivo específico mientras evita obstáculos de manera efectiva, basándose únicamente en la información proporcionada el LiDAR. El proceso de cálculo de la recompensa diferida se describe mediante la siguiente ecuación:

$$r_{t-i} = r(s_{t-i}, a_{t-i}) + \frac{r_m}{i} \tag{2.6}$$

donde r_{t-i} representa la recompensa diferida en el paso de tiempo $t - i$, $r(s_{t-i}, a_{t-i})$ es la recompensa asociada al par estado-acción en el paso de tiempo $t - i$, y i es el índice que recorre los pasos anteriores hasta n.

Capítulo 3

Modelado y control cinemático

Entre los diversos tipos de configuración de robots móviles (diferencial, ackerman, triciclo, omnidireccional, etc) la más utilizada en los robots móviles, es la diferencial. Este tipo de direccionamiento está dado por la diferencia de velocidades de las ruedas laterales, en un solo eje, las cuales son propulsadas y controladas independientemente, proporcionando tracción y direccionamiento. Debido a esto, permite cambiar la orientación del robot sin movimientos de traslación [74, 75]. Uno de los problemas es el equilibrio del robot, por lo que necesita ruedas de apoyo en forma triangular o romboidal [76].

En este trabajo de tesis, se hará uso del robot *Turtlebot 3 (Burger)*, el cual es un robot móvil de cinemática diferencial empleado para investigación y educación. [64]. Tiene como soporte el sistema operativo ROS con el cual se pueden implementar controladores para el *TurtleBot 3*.

3.1. Modelado

El modelo y el control cinemático del robot diferencial se realizará en un punto de operación diferente al eje de acción de las ruedas, se consideran las siguientes hipótesis [77]:

- El robot se mueve en una superficie plana.
- El eje de rotación es perpendicular a la superficie en la que se desplaza el robot.
- Las ruedas del robot se mueven sin restricciones.
- El robot no tiene partes flexibles.

3.1.1. Modelo cinemático del robot

Partiendo del modelo cinemático en un punto de operación centrado en el eje de acción de las ruedas, mostrado en la Figura 3.1, cuyo modelo cinemático se expresa con la ecuación (3.1).

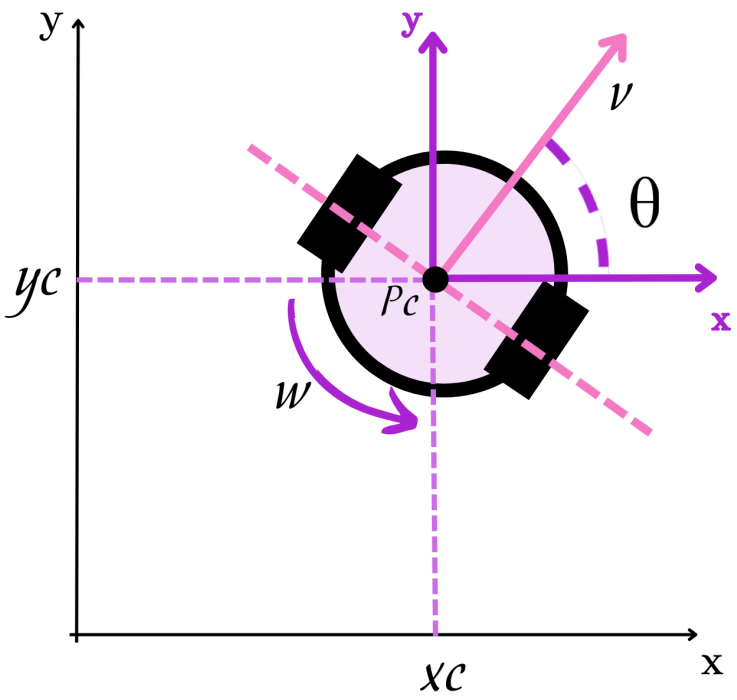


Figura 3.1: Punto en el eje de acción de las ruedas.

$$\begin{aligned} \dot{x_c} &= v\ cos\theta \\ \dot{y_c} &= v\ sen\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{3.1}$$

Donde $\dot{x_c}$ y $\dot{y_c}$ son las velocidades del punto central (P_c) en los ejes coordenados x y y, respectivamente. θ es el ángulo entre el vector de velocidad v de velocidad, con respecto al eje coordenado del punto c. Por lo que la velocidad de cambio del ángulo $\dot{\theta}$ es igual a la velocidad angular del robot ω.

3.1.2. Modelo cinemático del robot en un punto h

Para realizar el control en un punto desplazado una distancia h con respecto al centro del eje de acción de las ruedas, como se muestra en la Figura 3.2, y de acuerdo con [78, 79], la ecuación cinemática (3.2) se deriva (ecuación (3.3)) para obtener la velocidad del punto h.

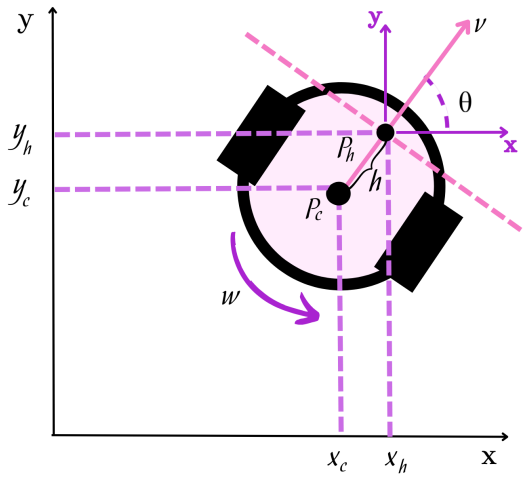


Figura 3.2: Punto desplazado una distancia h.

$$\begin{aligned} x_h &= x_c \ + \ h\ cos\theta \\ y_h &= y_c \ + \ h\ sen\theta \end{aligned} \tag{3.2}$$

Para obtener la velocidad del punto h, se obtiene la derivada de la ecuación 3.2.

$$\begin{aligned} \dot{x_h} &= \dot{x_{cp}} \ - \ h\ \dot{\theta}\ sin\theta \\ \dot{y_h} &= \dot{y_{cp}} \ + \ h\ \dot{\theta}\ cos\theta \end{aligned} \tag{3.3}$$

Se sustituye la velocidad en x y y del punto c (ecuación (3.1)), en las ecuaciones de la velocidad de punto h (ecuación (3.3)) así mismo se sustituye la derivada del ángulo θ por la velocidad angular ω.

$$\begin{aligned} \dot{x_h} &= v\ cos\theta \ - \ h\ \omega\ sin\theta \\ \dot{y_h} &= v\ sin\theta \ + \ h\ \omega\ cos\theta \end{aligned} \tag{3.4}$$

Donde x_h y y_h son las componentes en x y y del punto de operación desplazado una distancia h. Posteriormente se obtienen las componentes x y y de la velocidad del punto h. ($\dot{x_h}$, $\dot{y_h}$)

3.1.3. Modelo cinemático del robot en un punto arbitrario

El punto de control se encuentra fuera del eje de acción de las ruedas, y desplazado fuera del eje del vector v como se muestra en la Figura 3.3. Donde el θ_h es el ángulo entre el vector v y el eje del punto c, el cual es constante.

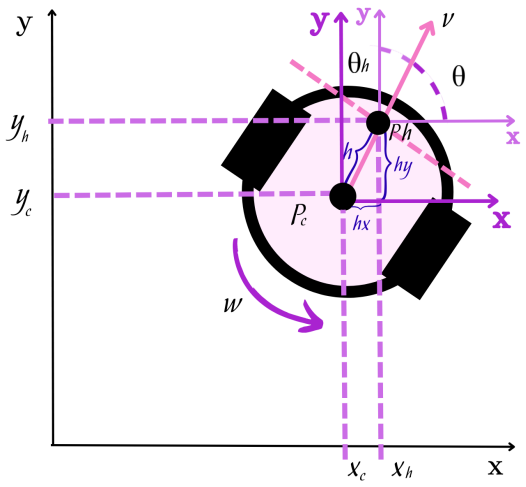


Figura 3.3: Punto fuera del eje de acción de las ruedas.

$$\begin{aligned} x_h &= x_{cp} \ + \ h \ cos(\theta + \theta_h) \\ y_c &= y_{cp} \ + \ h \ sen(\theta + \theta_h) \\ \theta_h &= constante \end{aligned} \tag{3.5}$$

Se deriva la posición del punto de control h (ecuación (3.5)) para obtener la velocidad en x y en y se presenta en la ecuación (3.6).

$$\begin{aligned} \dot{x_h} &= \dot{x_{cp}} \ - \ h \ \dot{\theta} \ sen(\theta + \theta_h) \\ \dot{y_h} &= \dot{y_{cp}} \ + \ h \ \dot{\theta} \ cos(\theta + \theta_h) \end{aligned} \tag{3.6}$$

Se sustituyen las velocidades en x, y y la derivada del ángulo θ del punto c (ecuación (3.1)), en las ecuaciones de la velocidad del punto h (ecuación (3.6)), obteniendo así la ecuación (3.7).

$$\begin{aligned} \dot{x_h} &= v \ cos\theta \ - \ h \ \omega \ sen(\theta + \theta_h) \\ \dot{y_h} &= v \ sin\theta \ + \ h \ \omega \ cos(\theta + \theta_h) \end{aligned} \tag{3.7}$$

3.2. Control cinemático

Con el control, se busca que el robot móvil siga un camino específico. Para ello, se deben obtener las leyes de control que permitan estabilizar al robot en un punto de trabajo, de manera que el error entre la posición deseada y la posición actual del robot tienda a cero. Existen dos tipos de control: por regulación y por seguimiento. El control por regulación considera la diferencia entre la posición actual y la deseada, a lo que se le denomina error, y este se multiplica por una ganancia k. Con este control se asegura que el robot llegue a un punto deseado; sin embargo, cuando la referencia se mueve de forma constante, el robot no alcanza por completo la referencia. Por lo tanto, se emplea el control por seguimiento, el cual considera la velocidad de cambio de la trayectoria, permitiendo que el sistema sea más estable y que el robot alcance la referencia.

3.2.1. Control cinemático del robot en un punto h

Partiendo del modelo del robot en un punto h, Figura 3.2, empleando la ecuación (3.1) se sustituyen los valores de $\dot{x_c}$ y $\dot{y_c}$, obteniendo la ecuación (3.4), esta ecuación se representa en forma matricial con la ecuación (3.8).

$$\begin{bmatrix} \dot{x_h} \\ \dot{y_h} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -h\ sen(\theta) \\ sen(\theta) & h\ cos(\theta) \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.8}$$

La matriz de rotación en el eje z (ecuación (3.8)) es la matriz Jacobiana, la cual representa la cinemática directa de velocidad y se expresa como:

$$J = \begin{bmatrix} cos(\theta) & -h\ sen(\theta) \\ sen(\theta) & h\ cos(\theta) \end{bmatrix} \tag{3.9}$$

El objetivo es encontrar las velocidades lineal y angular del robot, esto se puede conseguir calculando el Jacobiano inverso, ecuación(3.11), de esta manera, la cinemática inversa se expresa de la siguiente forma:

$$\begin{bmatrix} v \\ w \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x_h} \\ \dot{y_h} \end{bmatrix} \tag{3.10}$$

$$J^{-1} = \frac{1}{h} \begin{bmatrix} h\ cos(\theta) & h\ sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix} \tag{3.11}$$

Para aplicar el control del robot, el objetivo principal es que el error entre la velocidad del robot y la velocidad deseada sea cero. Este error, denominado e, se define como la diferencia entre las velocidades actuales y las deseadas, como se muestra en la ecuación (3.12).

$$\begin{aligned} \dot{e_x} &= \dot{x_h} - \dot{x_{hd}} \\ \dot{e_y} &= \dot{y_h} - \dot{y_{hd}} \end{aligned} \tag{3.12}$$

El control está diseñado para minimizar este error, de manera que, cuando $e \to 0$, se logre que la velocidad del robot coincida con la velocidad deseada en ambos ejes x y y, obteniendo la ecuación (3.13).

$$\begin{aligned} 0 = (\dot{x_h} - \dot{x_{hd}}) \Rightarrow \dot{x_h} = \dot{x_{hd}} \\ 0 = (\dot{y_h} - \dot{y_{hd}}) \Rightarrow \dot{y_h} = \dot{y_{hd}} \end{aligned} \tag{3.13}$$

Se realiza un cambio de variable de la velocidad del punto h, por la variable u como la variable de control por lo que la velocidad en x queda expresada de la siguiente forma $\dot{x_h} = u_x$, ecuación 3.14. Se agrega un término de corrección de la posición multiplicado por una ganancia k, como se muestra en la ecuación 3.15, esto se conoce como control por seguimiento.

$$\begin{aligned} u_x = \dot{x_{hd}} \\ u_y = \dot{y_{hd}} \end{aligned} \tag{3.14}$$

$$\begin{aligned} u_x = \dot{x_{hd}} - k\ (x_h - x_{hd}) \\ u_y = \dot{y_{hd}} - k\ (y_h - y_{hd}) \end{aligned} \tag{3.15}$$

Cuando se desee que el robot llegue de un punto inicial a un punto de destino y que se mantenga en esa posición específica, la velocidad deseada $\dot{x_{hd}}$ es cero. Lo que resulta en (Ecuación (3.16)) Esto es conocido como control de regulación.

$$\begin{aligned} u_x = -k\ (x_h - x_{hd}) \\ u_y = -k\ (y_h - y_{hd}) \end{aligned} \tag{3.16}$$

Ya sea que se realicen tareas de regulación o seguimiento, se puede obtener la velocidad lineal y angular mediante la cinemática inversa, la cual expresada en términos del control se muestra en la ecuación (3.17)

$$\begin{bmatrix} v \\ w \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x_h} \\ \dot{y_h} \end{bmatrix} \Rightarrow \begin{bmatrix} v \\ w \end{bmatrix} = J^{-1} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{3.17}$$

Por lo tanto, la velocidad lineal y angular que debe llevar el robot en todo momento se describe a través de las leyes de control (3.16) y (3.15). De manera que:

$$\begin{aligned} v = u_x\ cos(\theta) + u_y\ sen(\theta) \\ \omega = \frac{-u_x\ sen(\theta) + u_y\ cos(\theta)}{h} \end{aligned} \tag{3.18}$$

3.2.2. Control cinemático en un punto arbitrario

Partiendo del modelo del robot en un punto h posicionado de forma arbitraria Figura 3.3, se expresa la ecuación (3.7) de la siguiente manera:

$$\begin{bmatrix} \dot{x_h} \\ \dot{y_h} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -h\ sen(\theta + \theta_h) \\ sen(\theta) & h\ cos(\theta + \theta_h) \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.19}$$

Al igual que en el control en el punto h, la matriz de rotación en el eje z, es la matriz Jacobiana (3.19).

$$J = \begin{bmatrix} cos(\theta) & -h\ sen(\theta + \theta_h) \\ sen(\theta) & h\ cos(\theta + \theta_h) \end{bmatrix} \tag{3.20}$$

Para encontrar la velocidad lineal y la velocidad angular del robot, se calcula el determinante del Jacobiano para determinar si tiene inversa, ecuación (3.20). Hay puntos en los que el determinante es igual a cero, ecuación (3.21), por lo tanto la matriz no tiene inversa en estos casos, por lo que se limita el rango de acción y se obtiene la inversa, ecuación (3.22).

$$det(J) = h\ cos(\theta_h) \tag{3.21}$$

$$det(J) \neq 0\ si\ \theta_h \neq (2n+1)\frac{\pi}{2}$$

$$J^{-1} = \frac{1}{hcos(\theta_h)} \begin{bmatrix} h\ cos(\theta + \theta_h) & h\ sen(\theta + \theta_h) \\ -sen(\theta) & cos(\theta) \end{bmatrix} \tag{3.22}$$

$$\begin{bmatrix} v \\ w \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x_h} \\ \dot{y_h} \end{bmatrix} \tag{3.23}$$

Por lo que el control de la velocidad lineal y la velocidad angular, queda expresado en la ecuación (3.24).

$$\begin{aligned} v &= \frac{u_x\ cos(\theta + \theta_h) + u_y\ sen(\theta + \theta_h)}{cos(\theta_h)} \\ \omega &= \frac{-u_x\ sen(\theta) + u_y\ cos(\theta)}{h\ cos(\theta_h)} \end{aligned} \tag{3.24}$$

3.3. Simulación del control cinemático

Para la simulación, se ejecutó el robot *Turtlebot 3* en el entorno vacío por defecto de Gazebo, el simulador de ROS utilizado para desarrollar robots y probar algoritmos. Para ello, se

identificó el tópico */Odom*, que emite la posición y orientación del robot (odometría). A partir de este mensaje, se obtiene la posición en x y y. En cuanto a la orientación del robot, esta se encuentra representada en cuaterniones, por lo que es necesario convertirlos a ángulos de Euler. Esto permite obtener la orientación en el eje z en grados, que corresponde a la velocidad angular del robot.

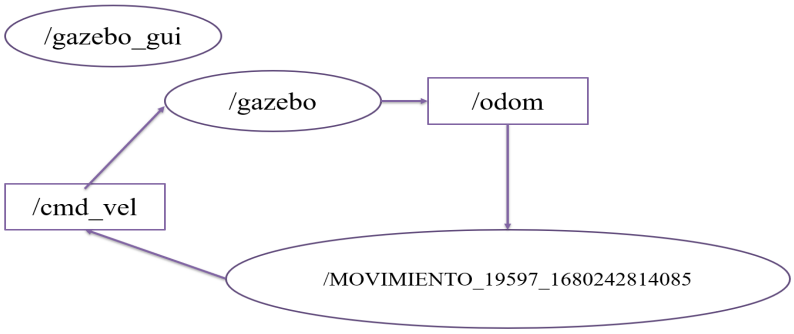


Figura 3.4: Tópicos

El tópico *cmd_vel* envía al robot la velocidad lineal y angular con la que se debe mover el robot, para ello se publican en el tópico los valores de velocidad lineal (velocidad lineal en x del tópico) y angular (velocidad angular en z). Los tópicos empleados en la simulación se muestran en el diagrama "rqt" (Figura 3.4), en el cual se puede apreciar como fue la comunicación entre el *Trutlebot 3*, y el código para el control.
Se realizaron diversas pruebas del control de acuerdo a las siguientes acciones:

a. Llegar a una posición deseada.
b. Seguir una trayectoria en un tiempo deseado.
c. Seguir una trayectoria (Lemniscata)

En las trayectorias, que dependen del tiempo, se realizó una prueba empleando el control mediante regulación, y una prueba utilizando control por seguimiento.

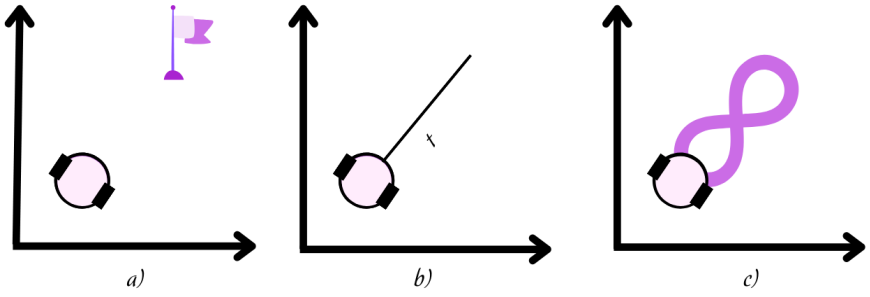


Figura 3.5: Acciones de control a) a un punto, b) una trayectoria recta y c) una trayectoria (Lemniscata)

3.3.1. Control cinemático a un punto deseado

Para la navegación a un punto específico se realiza el control por regulación para obtener las componentes x y y del punto h (ecuación (3.25)) posteriormente se obtiene el error ecuación (3.12) y finalmente la velocidad lineal y angular para controlar el robot, ecuación 3.18. Para ello, se utilizó el código disponible en https://github.com/itzchav/Control-cinematico-turtlebot/blob/main/move_ws/src/mov_turtle/src/control_punto_arg.py. La ejecución de la simulación se puede consultar en el repositorio en: https://github.com/itzchav/Control-cinematico-turtlebot/tree/main).

$$\begin{aligned} xh &= x + h\ cos(\theta) \\ yh &= y + h\ sin(\theta) \end{aligned} \tag{3.25}$$

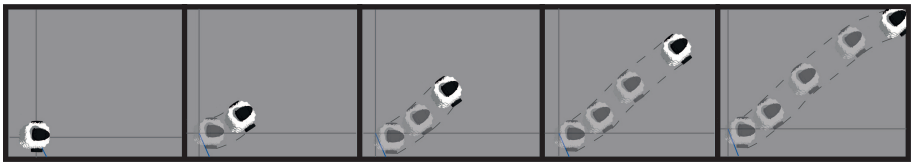


Figura 3.6: Control cinemático a un punto

En la Figura 3.6 se ejemplifica el recorrido del *Turtlebot 3* en el entorno de simulación Gazebo. Se elaboró una gráfica de las posiciones x y y hacia el punto deseado [1 m, 1 m] (Figura 3.7). En esta última, se observa un desfase en el eje x, debido a que el robot está centrado en el origen, pero el punto h que se controla está desfasado 0.05 m.

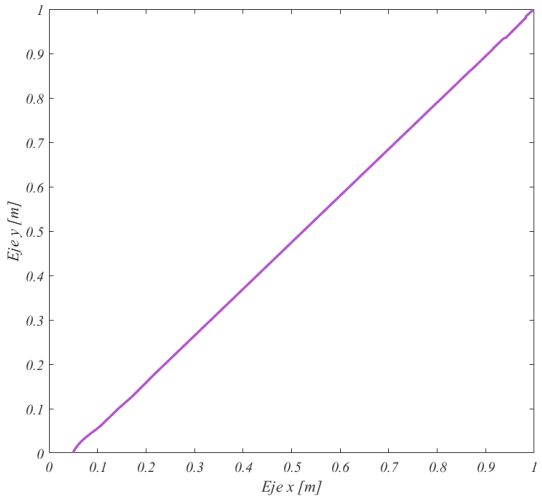
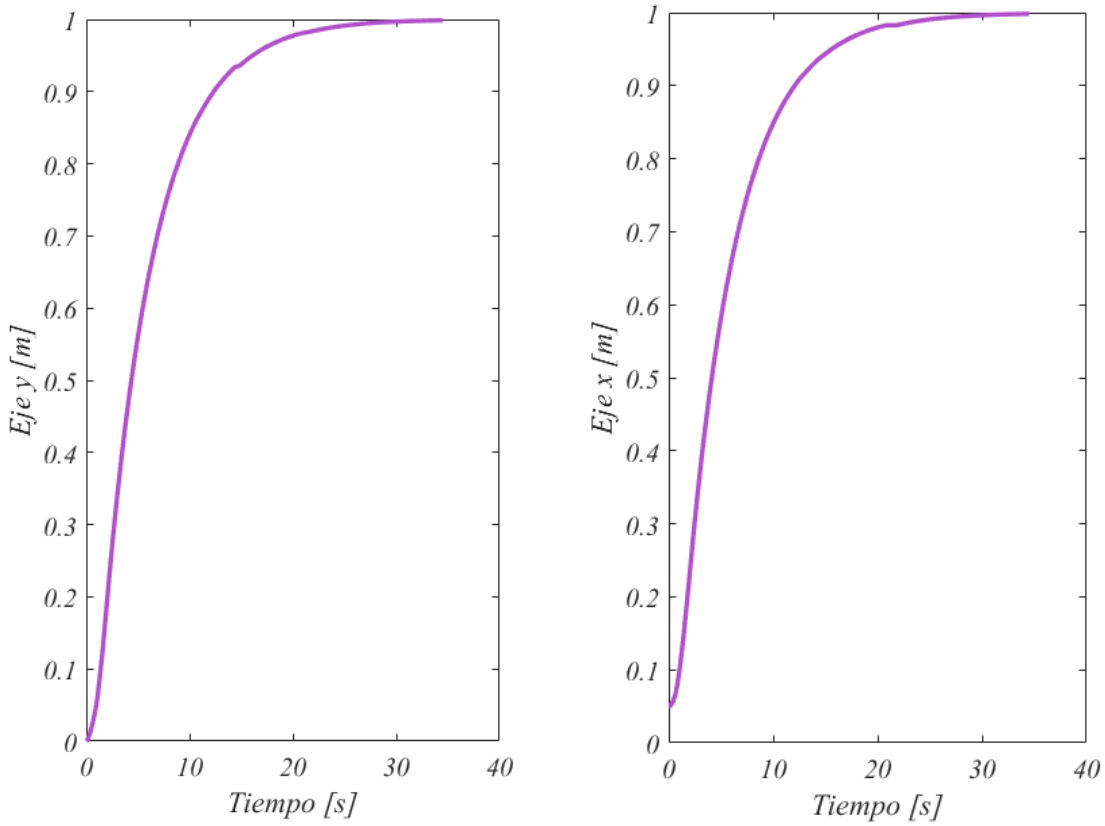


Figura 3.7: Control cinemático (Posición en x y y)

Posteriormente, se presentan las gráficas de las posiciones x y y en función del tiempo, como se muestra en la Figura 3.8.



(a) (b)

Figura 3.8: Control cinemático con respecto al tiempo a) Posición en x b) Posición en y

3.3.2. Control cinemático (Trayectoria en línea recta)

Para realizar la trayectoria de una línea recta en un tiempo determinado, se empleó la ecuación de la línea recta con respecto al tiempo, ecuación (3.26). Por lo que fue necesario obtener el tiempo de simulación y así calcular el tiempo de ejecución del robot. El código está disponible en https://github.com/itzchav/Control-cinematico-turtlebot/blob/main/move_ws/src/mov_turtle/src/control_trayectoria_recta.py.

$$\begin{aligned} x_d &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ y_d &= m x_d + b \end{aligned} \tag{3.26}$$

Donde:

a_0=x_0	x_0 = Posición inicial en x
a_1=0	x_f = Posición final en x
a_2=$(3(x_f - x_0)/(t_f^2))$	y_0 = Posición inicial en y
a_3=$(-2(x_f - x_0)/(t_f^3))$	y_f = Posición final en y
m=$(y_f - y_0)/(x_f - x_0)$	t = Tiempo de ejecución
b=$y_0 - m * x_0$	t_f = Tiempo final

En la Figura 3.9 se muestra el recorrido del *Turtlebot 3* en el entorno de simulación Gazebo en un tiempo especifico. Se realizó una gráfica de la posición con respecto x y y, para observar que el robot llegue al punto deseado y una gráfica de la posición de las coordenadas x con respecto al tiempo.

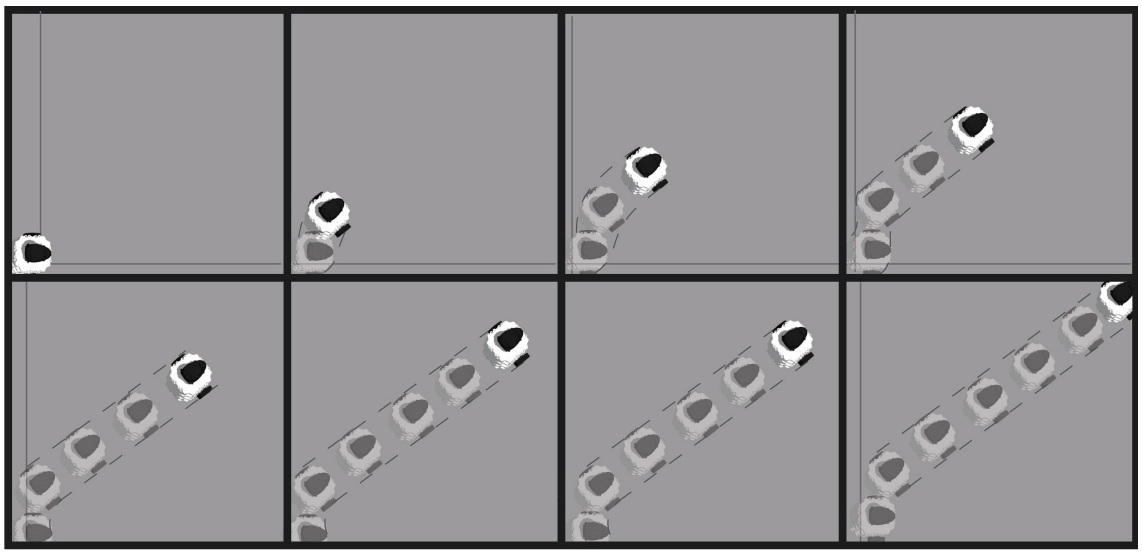
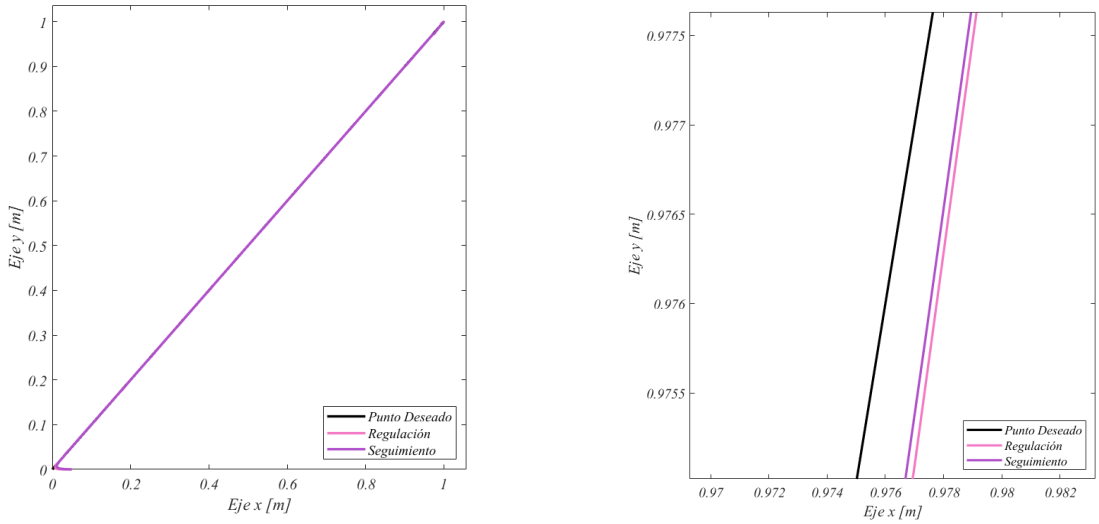


Figura 3.9: Control cinemático de trayectoria a un punto deseado

En la Figura 3.10 se observa la diferencia entre el control cinemático por regulación y el control cinemático por seguimiento, para este último se calculó la derivada de las ecuaciones de movimiento, ecuación (3.26), obteniendo así la ecuación (3.27).
Aunque ambos controles siguen la referencia, al considerar la velocidad deseada, el control cinemático por seguimiento tiene un menor error con respecto a la referencia. A diferencia del control por regulación, al no considerar la velocidad deseada, implica que el punto deseado de la trayectoria con respecto al tiempo se actualice más rápido que el control cinemático.

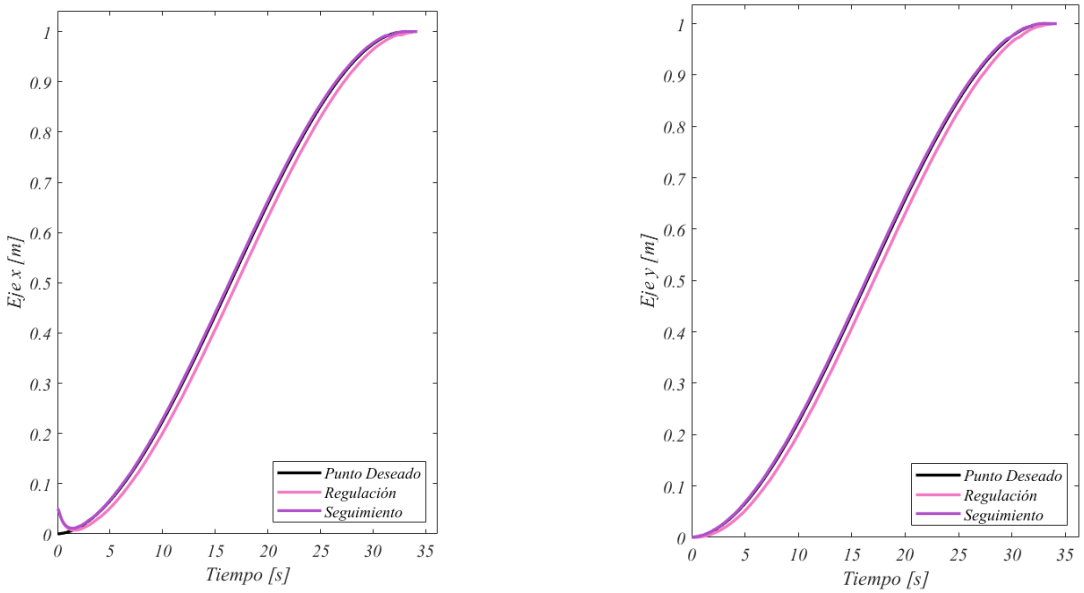
$$\begin{aligned} \dot{x_d} &= a_1 + 2a_2t + 3a_3t^2 \\ \dot{y_d} &= m\dot{x_d} \end{aligned} \tag{3.27}$$



(a) (b)

Figura 3.10: a) Trayectoria en el espacio XY y b) Ampliación para observar el control por regulación y seguimiento

En la Figura 3.11 se puede observar que para llegar a [1 m, 1 m] en un tiempo establecido, se estableció una trayectoria en linea recta, la cual actualiza el punto deseado con respecto al tiempo, en este caso, la referencia se actualiza para llegar al punto [1 m, 1 m] en 30 segundos.



(a) (b)

Figura 3.11: a) Posición x b) Posición y con respecto al tiempo

3.3.3. Control cinemático (Trayectoria Lemniscata)

Para el seguimiento de una trayectoria lemniscata, como se muestra en la Figura 3.5, se utilizaron las ecuaciones dependientes del tiempo (3.28) para describir dicha trayectoria. En el control por seguimiento, se calculó la derivada para obtener la velocidad de la referencia (ecuación (3.29)), donde t representa el tiempo de ejecución. La variable a ajusta la amplitud de la trayectoria, mientras que b permite desplazar el centro con respecto al eje y. El código correspondiente está disponible en el siguiente enlace: https://github.com/itzchav/Control-cinematico-turtlebot/blob/main/move_ws/src/mov_turtle/src/control_trayectoria_cata_ph.py. Además, la ejecución de la simulación se puede consultar en el repositorio en: https://github.com/itzchav/Control-cinematico-turtlebot/tree/main.

$$\begin{aligned} x_d &= \frac{a\ cos(t)}{2 + sin^2(t)} \\ y_d &= \frac{a\ sin(t)cos(t)}{2 + sin^2(t)} + b \end{aligned} \tag{3.28}$$

$$\begin{aligned} \dot{x_d} &= \frac{-sin(t)(2 + sin^2(t)) - sin(2t)cos(t)}{(2 + sin^2(t))^2} \\ \dot{y_d} &= \frac{cos(2t)(2 + sin^2(t)) - sin(2t)sin(t)cos(t)}{(2 + sin^2(t))^2} \end{aligned} \tag{3.29}$$

En la Figura 3.12 se el recorrido del robot de la parte izquierda de la trayectoria Lemniscata, en el simulador Gazebo, en ella se muestra una sombra de donde el robot ha pasado, con la finalidad de poder observa la forma de la trayectoria recorrida por el robot.

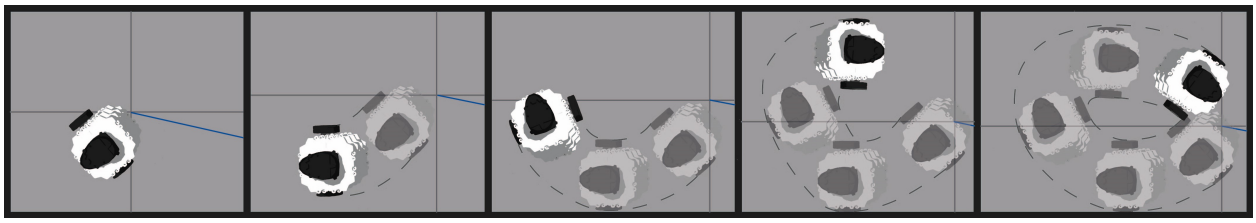


Figura 3.12: Trayectoria Lemniscata (lado izquierdo)

Posteriormente en la Figura 3.13 se observa el recorrido completo del robot, en el cual se logra apreciar la trayectoria Lemniscata.

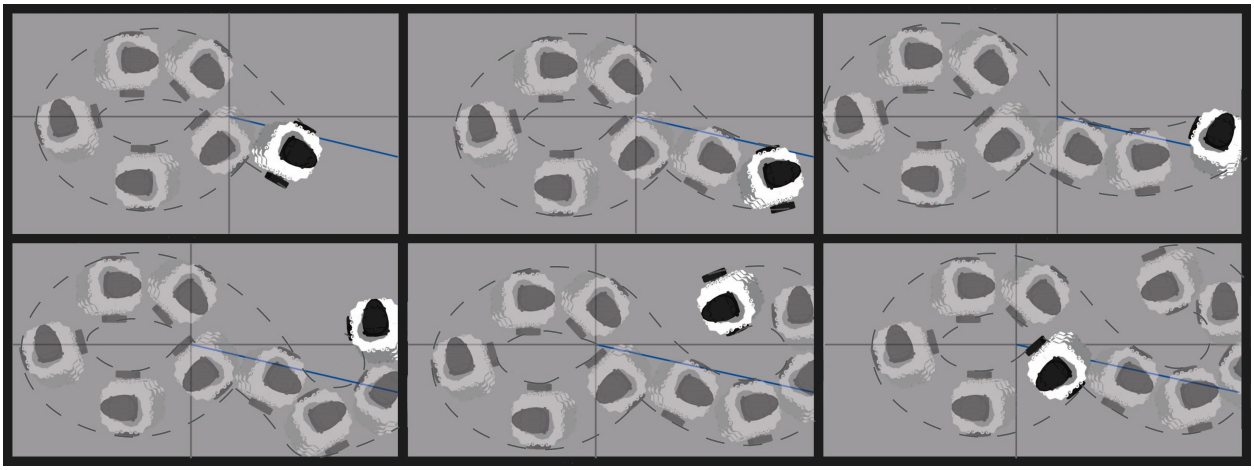


Figura 3.13: Control de trayectoria (Lemniscata)

Se realizó una gráfica de la posición en x y y (Figura 3.14), en la que se muestra la trayectoria deseada de color negro, el control por regulación y por seguimiento, se observa que el control por seguimiento tiene un menor error que el control por regulación, en la Figura 3.15 se muestra la posición en x y y con respecto al tiempo.

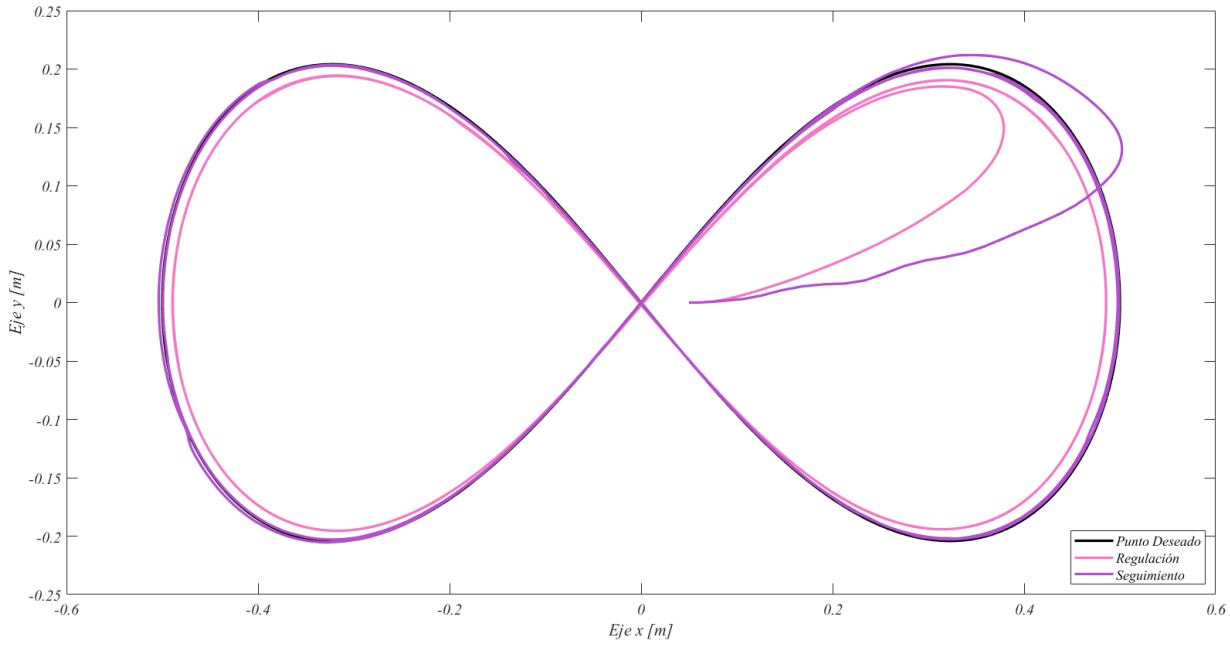
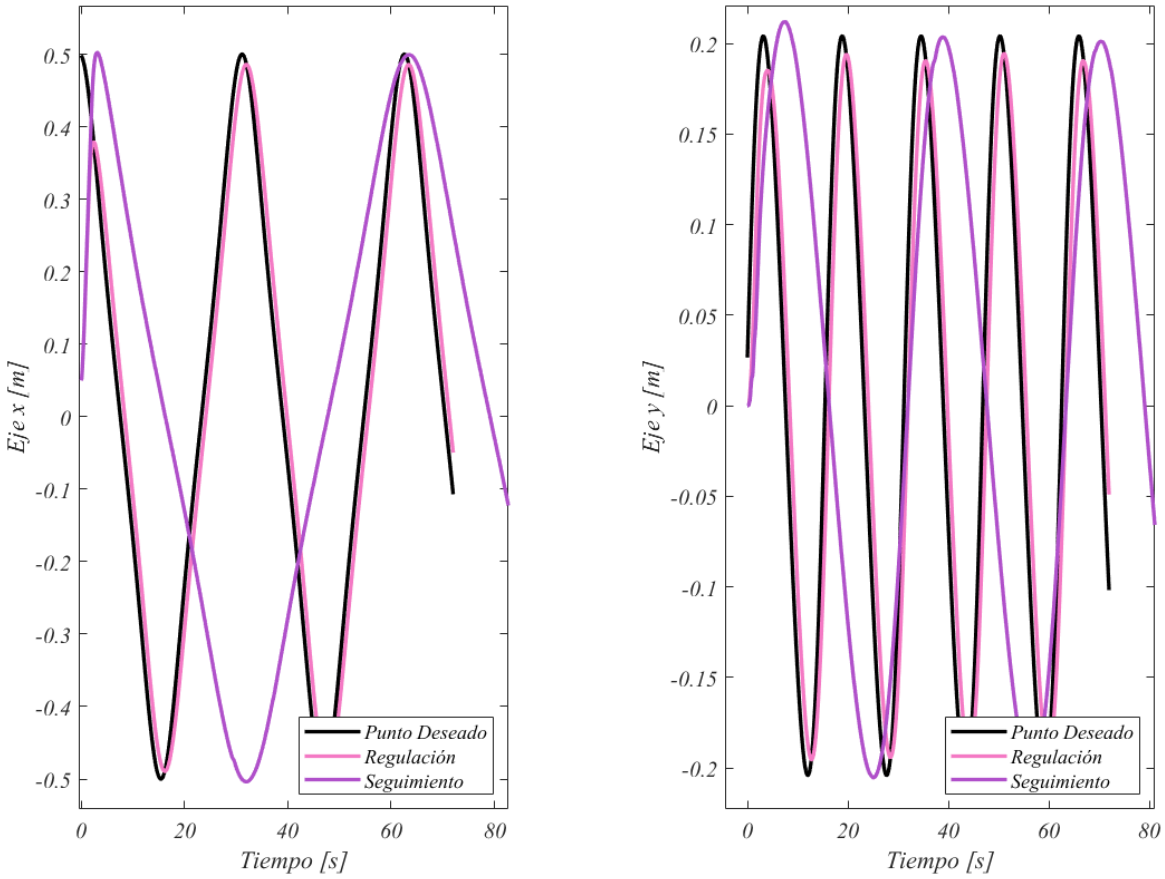


Figura 3.14: Posición x y y del robot al realizar una trayectoria de Lemniscata



(a) (b)

Figura 3.15: a) Posición x b) Posición y con respecto al tiempo

Capítulo 4

Entorno de simulación

En este capítulo se presenta el entorno de simulación 3D del edificio de la "División de Estudios de Posgrado" de la UTM, que tiene una geometría triangular poco convencional y una superficie aproximada de $494m^2$. Las aulas, oficinas y el laboratorio están distribuidos como se muestra en la Figura 4.1. Se elaboró el modelo 3D en CAD, el cual se exporta a ROS Gazebo, junto con las adaptaciones necesarias para obtener el mapa mediante SLAM. También se describen las modificaciones realizadas para el entrenamiento del modelo utilizando RL, así como las adaptaciones del entorno de prueba para el modelo de RL una vez entrenado.

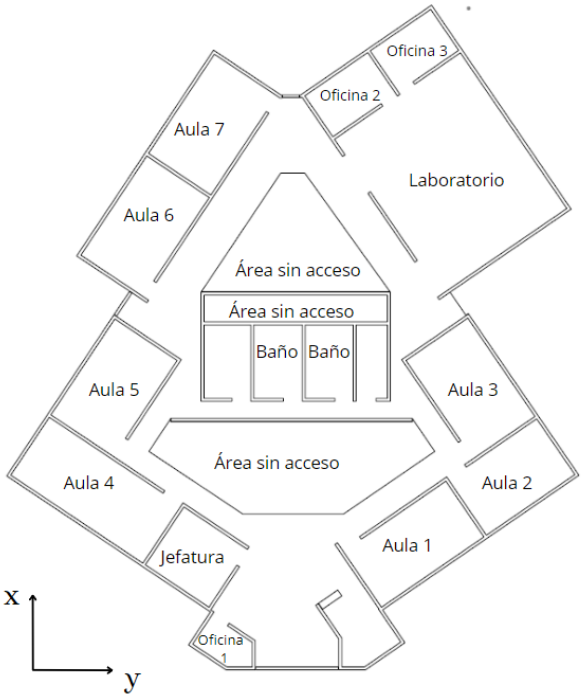
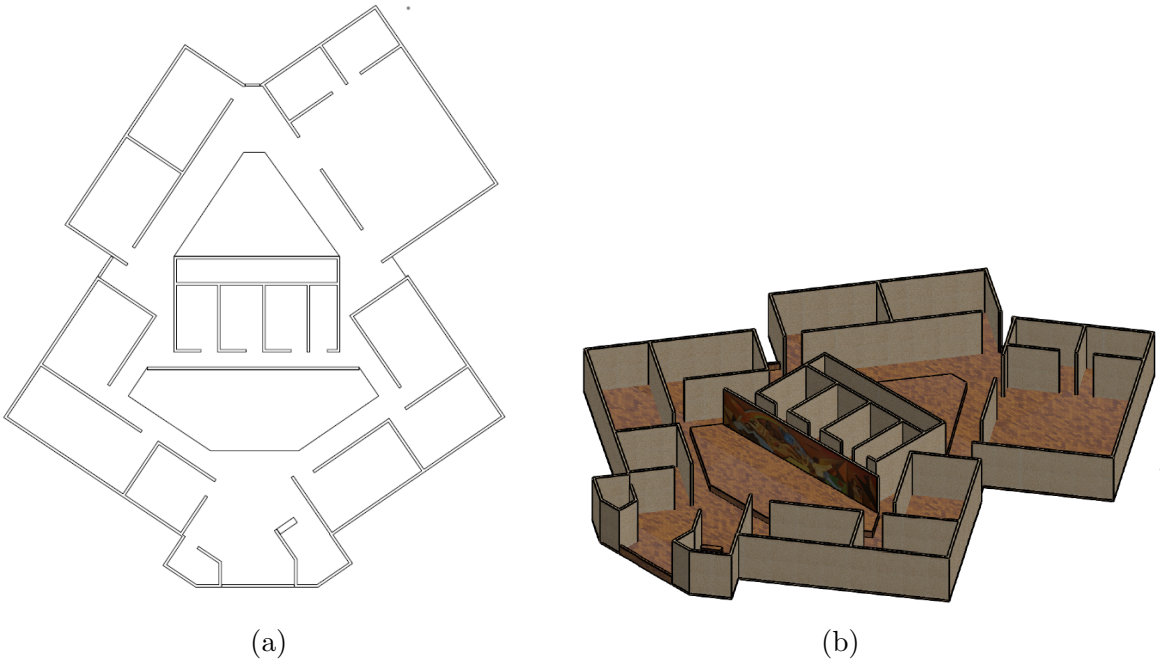


Figura 4.1: Distribución de los espacios en la División de Estudios de Posgrado

4.1. Modelo 3D del entorno.

Partiendo del plano del edificio de la "División de Estudios de Posgrado" de la UTM, proporcionado por el Departamento de obras (Figura A.1, en el Anexo D), se realizó el CAD en *SolidWorks*® como se muestra en la Figura 4.2a, se muestra una vista isométrica del entorno 3D en la Figura 4.2b.



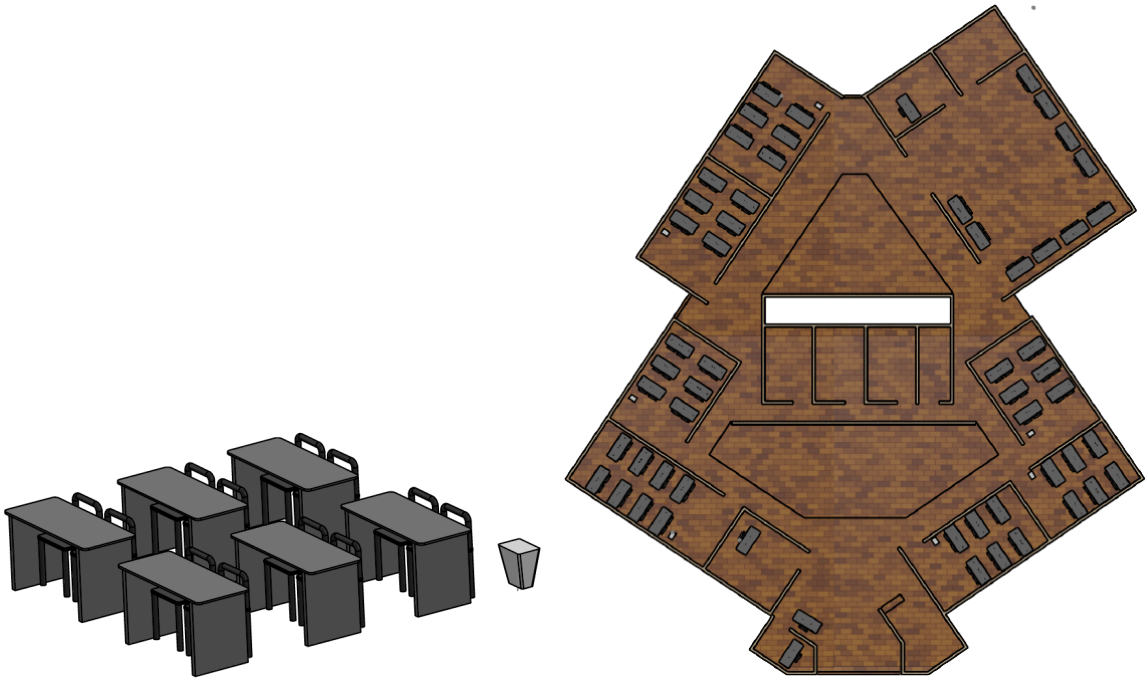
(a) (b)

Figura 4.2: Modelo 3D de la 'División de Estudios de Posgrado' a) Vista superior b) Vista isométrica.

Las consideraciones que se tomaron en cuenta para la realización del modelo son las siguientes:

- Las medidas se tomaron de acuerdo al plano.
- Debido a que el robot no puede bajar ni subir escalones, se establecieron muros para limitar esas áreas.
- Se omitieron las puertas, con la finalidad de que el robot pueda obtener el mapa de la planta baja del edificio.

Se consideraron obstáculos como escritorios, sillas y botes de basura (Figura 4.3a), los cuales están distribuidos en el entorno, tal como se muestra en la Figura 4.3b.



(a) (b)

Figura 4.3: Modelo 3D de la 'División de Estudios de Posgrado' con obstáculos a) Obstáculos. b) Vista superior con obstáculos.

4.1.1. Simulación del entorno 3D en Gazebo

El Modelo 3D fue exportado al simulador Gazebo en ROS, convirtiendo el archivo CAD en formato *'*.urdf*, para las texturas se necesitan los archivos en formato *'*.stl'* y *'*.dae'* en un archivo *'*.world'*, Para finalmente llamar los elementos en un archivo *'*.launch'*. Como se muestra en la Figura 4.4.

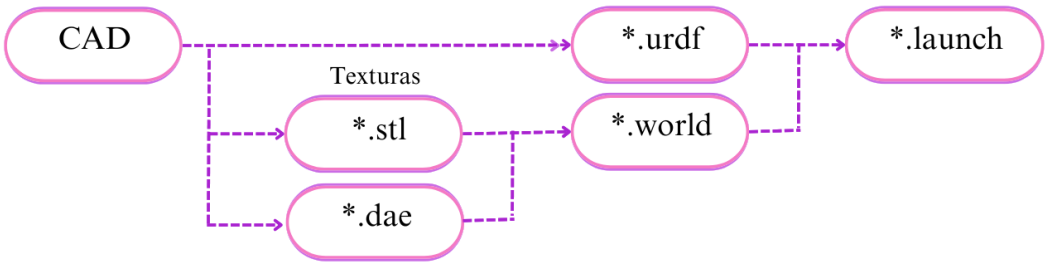


Figura 4.4: Diagrama de distribución de archivos

El diagrama de árbol en la Figura 4.5 representa la ubicación de las carpetas y archivos, los cuales se pueden consultar en el siguiente repositorio: https://github.com/itzchav/Entorno-Division-Estudios-Posgrado/tree/main. En la Figura 4.6, se presenta el resultado del entorno 3D en el simulador Gazebo.

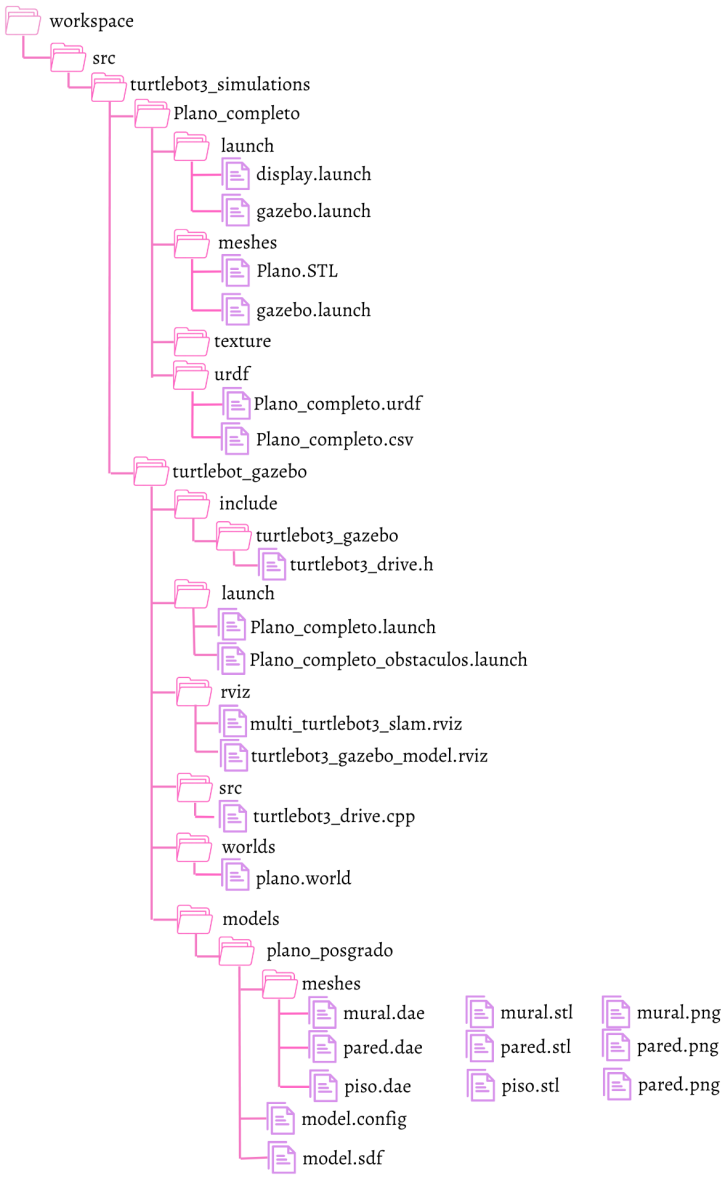


Figura 4.5: Árbol de archivos

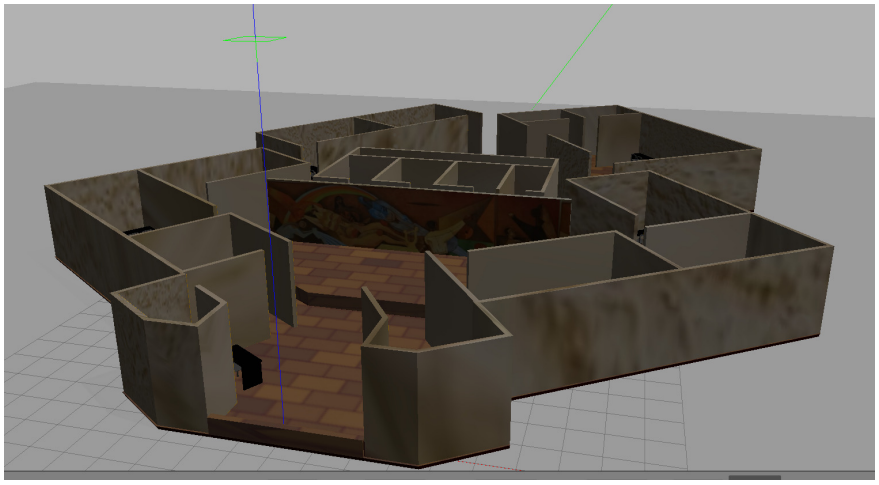


Figura 4.6: Simulación del entorno.

4.2. Integración de SLAM en el entorno

Se modificaron los parámetros del alcance del sensor LiDAR de 3 m a 12 m y el número de lecturas de 360 a 1147, en función de las características del sensor LiDAR disponible en el laboratorio de la División de Estudios de Posgrado de la UTM. Para generar el mapa utilizando el método SLAM, se ejecuta un archivo *launch* del código *Gmapping* en ROS, el cual se detalla en la sección C.1, este se incluyó en el código *launch* del plano, como se muestra en la sección C.2. La Figura 4.7 presenta la primera lectura del entorno realizada con SLAM. En el siguiente repositorio `https://github.com/itzchav/Entorno-Division-Estudios-Posgrado/tree/main` se pueden consultar los códigos para implementar *Gmapping*.

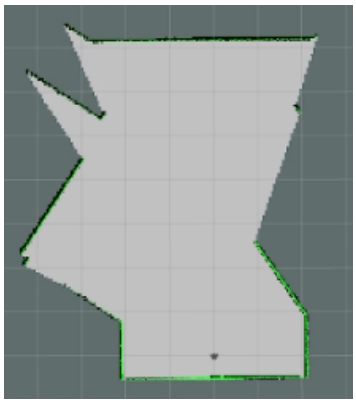


Figura 4.7: Ejecución de SLAM.

4.3. Integración del entorno en ROS Gazebo con RL

4.3.1. Entorno de entrenamiento

Partiendo del estudio realizado por Cimurs et al. [73], en el que se utiliza un robot *Pioneer*, se realizaron las adaptaciones para entrenar el *Turtlebot 3*. Para integrar el RL se utilizan dos códigos: uno que se comunica con Gazebo para obtener información del sensor y asignar al robot la velocidad deseada por el modelo, y otro que modifica los hiperparámetros y define las capas de la red neuronal. Para el entrenamiento se configuró que el punto de inicio y la meta se asignen de forma aleatoria, así mismo el sistema se reinicia cuando llega a la meta o colisiona, por otro lado unicamente se consideran 20 lecturas del sensor a 180 grados, en la parte frontal. Esto se ejemplifica en el diagrama de la Figura 4.8.

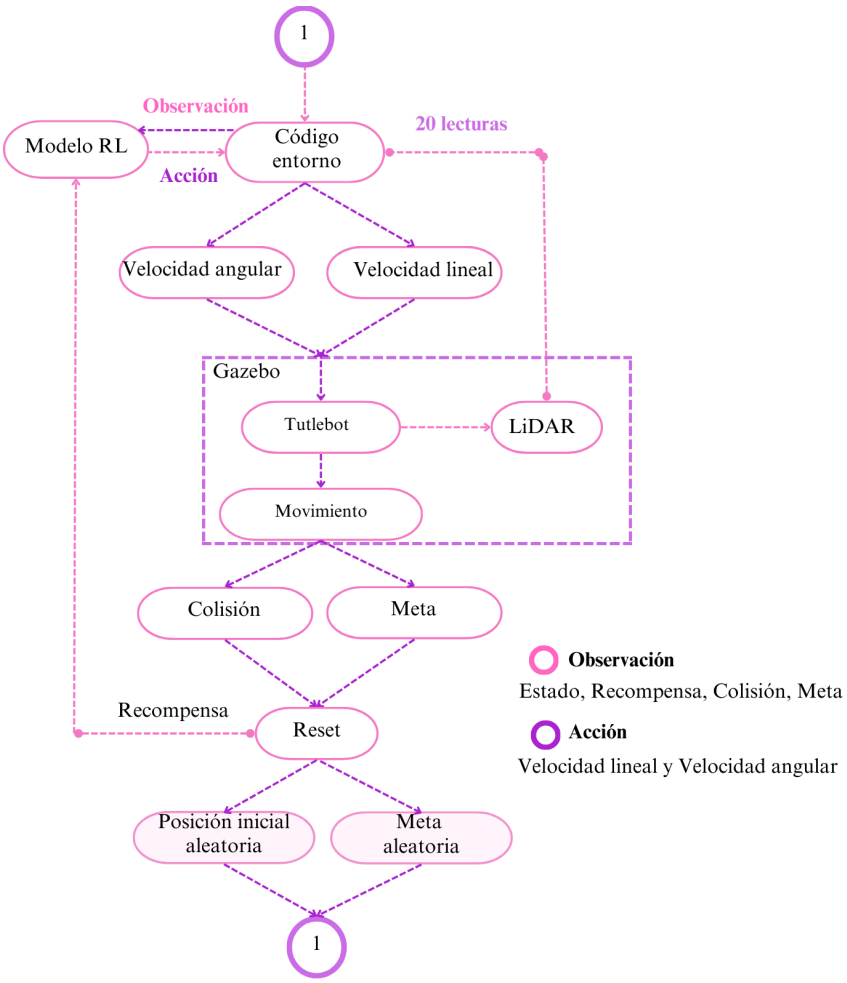


Figura 4.8: Diagrama del código de entrenamiento

El entorno de simulación en el que se realiza el entrenamiento se muestra en la Figura 4.9. Se realizaron ajustes en el código *'*.launch'* para cambiar el robot *Pioneer* por el *TurtleBot*, el cual se puede consultar en el siguiente enlace: https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/assets/multi_robot_scenario.launch. Además, se modificó el código en que se configura el entorno de simulación en Gazebo para el TurtleBot3. Este archivo incluye el lanzamiento de un mundo vacío y genera el modelo del robot, el cual se puede consultar en: https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/catkin_ws/src/multi_robot_scenario/launch/empty_world_2.launch.

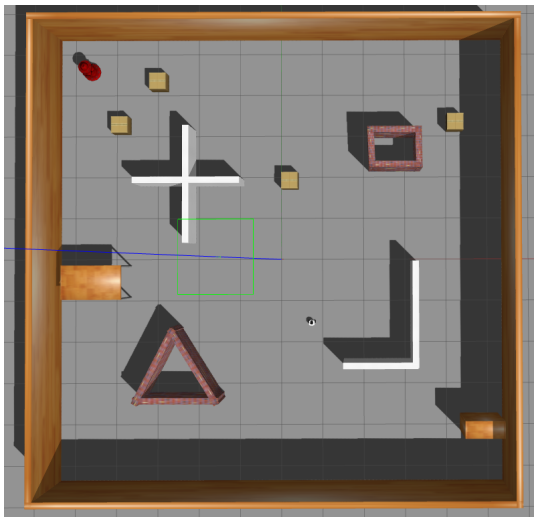
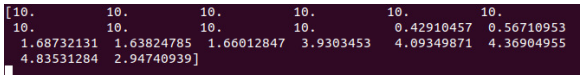
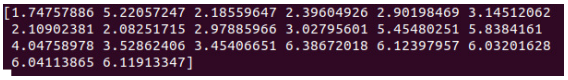


Figura 4.9: Entorno de simulación para el entrenamiento

En el código del entorno, se reemplazó el tópico que obtiene la información del sensor: el tópico $/velodyne_points$ se cambió a $/scan$. Se realizaron ajustes en el rango de lectura del sensor al ejecutar el código con el robot *Turtlebot 3*. Se observó que el vector *gaps*, que almacena las lecturas (ver Figura 4.10a), no registra los datos de manera adecuada. Esto se debe a que, al no obtener un dato de lectura correspondiente al ángulo deseado, se le asigna un valor de 10. Por lo tanto, la configuración original, que utilizaba un rango de -90 a 90 grados, no mostraba los resultados esperados.



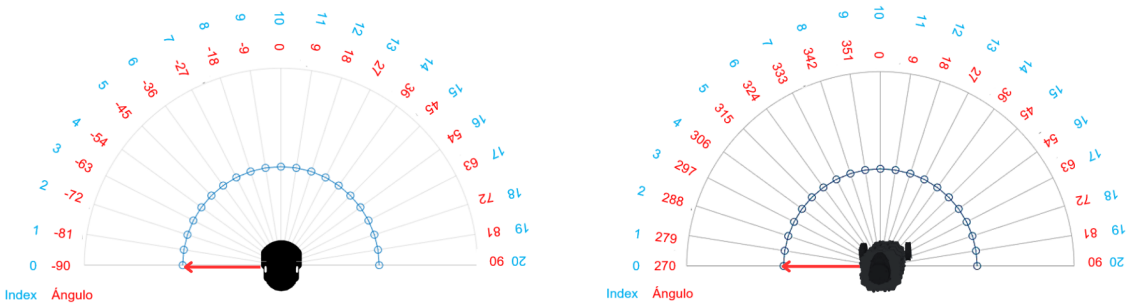
(a)



(b)

Figura 4.10: Lecturas del sensor LiDAR a) Vector de lecturas -90 a 90, b) Vector de lecturas 270 a 90

En la Figura 4.11a se describe el ángulo asociado al índice del vector (*index*). En consecuencia, se procedió a ajustar el rango de lectura de -270 a 360 grados y de 0 a 90 grados, como se ilustra en la Figura 4.11b. Esta modificación se realizó con el propósito de lograr el registro adecuado de las 20 lecturas de distancia, resultando en el vector mostrado en la Figura 4.10b. Estos cambios se pueden consultar en el siguiente enlace: https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/velodyne_env.py



(a) (b)

Figura 4.11: Relación entre los ángulos y el vector de lectura a) Robot *Pionner* b) Robot *Turtlebot3*

Para el entrenamiento del modelo, se modificó la recompensa asociada a los obstáculos. Las recompensas se ajustaron de acuerdo con la siguiente función:

$$r(s,a) = \begin{cases} r_m & \text{si } D < \eta_D \\ r_c + D & \text{si } D < \eta_C \\ v - |\omega| & \text{en otro caso} \end{cases} \tag{4.1}$$

Donde:
r_m = recompensa a la meta
r_c = recompensa por colisión
D = Distancia
η_D = Umbral donde se considera que llego a la meta
η_C = Umbral donde se considera que colisionó

El valor de la recompensa al colisionar depende de la distancia a la meta, expresándose como $r_c -$distancia. De esta manera, si el robot colisiona a una mayor distancia de la meta, recibirá una recompensa negativa más alta.
En el código para el entrenamiento, el cual se puede consultar en https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/train_turtlebot.py se modificaron los hiperparámetros, del modelo de acuerdo a la Tabla 4.1.

Tabla 4.1: Hiperparámetros de la red

Parámetro	Valor
environment_dim	20
action_dim	2
max_action	1
batch_size	40
discount	0.99999
policy_noise	0.15
noise_clip	0.5
policy_freq	2
buffer_size	1×10^6
eval_freq	5×10^3
max_ep	500
eval_ep	10
expl_noise	1
expl_decay_steps	500000
expl_min	0.1
max_timesteps	1×10^6

4.3.2. Entorno de prueba

Para las pruebas en el entono 3D de la División de Estudios de Posgrado (Figura 4.12), se modificó el funcionamiento de acuerdo al diagrama de la Figura 4.13, se modifica el código del entorno: https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/plano_env.py que es llamado en el código de prueba, el cual se puede consultar en el siguiente enlace: https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/test_plano.py. Para navegar en el entorno se asignó la posición inicial de un punto inicial de una trayectoria definida, así mismo la meta de cada episodio es un punto sucesivo de la trayectoria, en este caso el entorno no se reinicia cada que el robot llega a una meta, lo hace una vez que llega al final de la trayectoria.

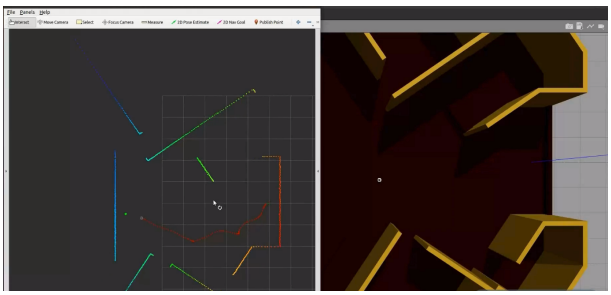


Figura 4.12: Entorno de prueba

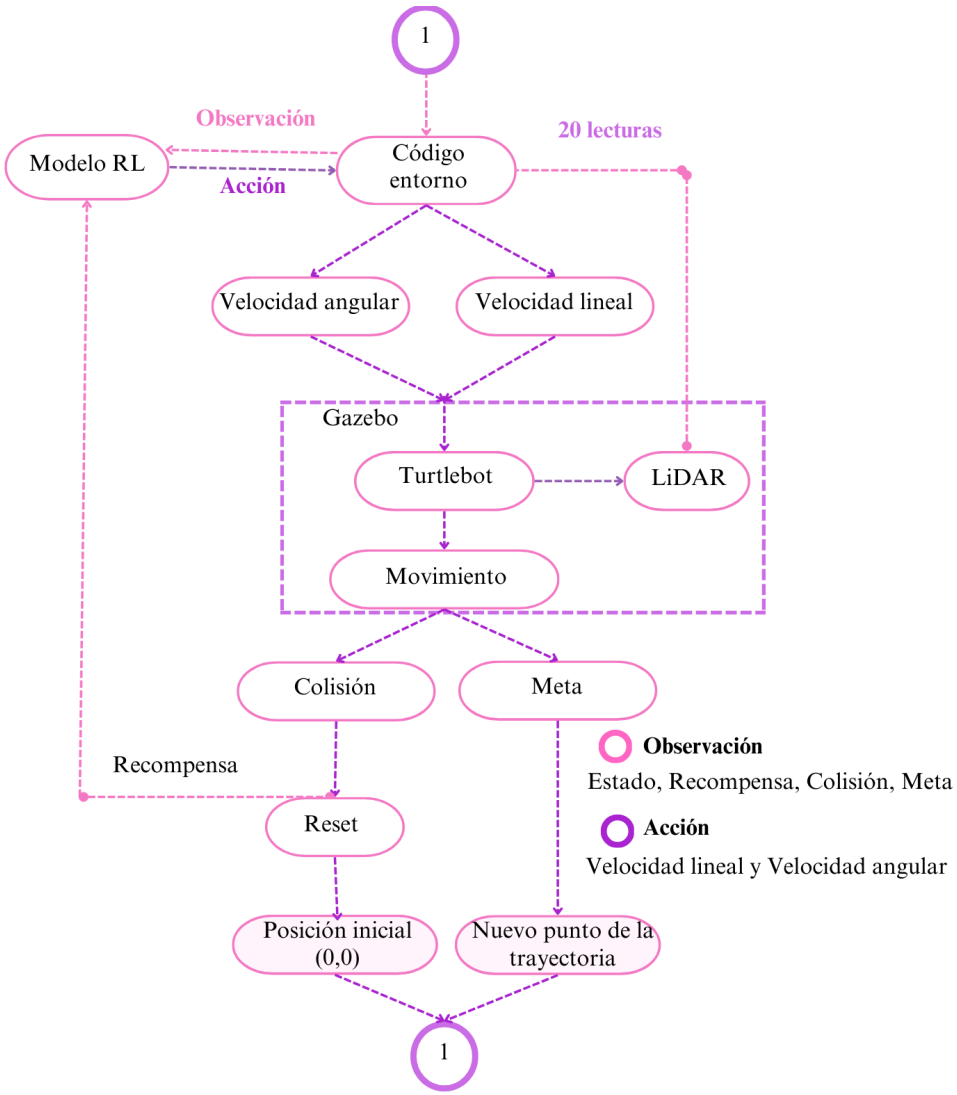


Figura 4.13: Diagrama del código de prueba

Capítulo 5

Navegación *offline*

La navegación *offline* se refiere al proceso en el cual, primero, se obtiene un mapa completo del entorno y, posteriormente, se genera una trayectoria predefinida hacia la meta (Figura 5.1). Esta trayectoria se asigna a un algoritmo de navegación que guía al robot a lo largo del recorrido. En este trabajo se emplean dos técnicas diferentes como algoritmo de navegación, el control cinemático y el modelo de RL, para recorrer las posiciones de la trayectoria hasta alcanzar la meta.

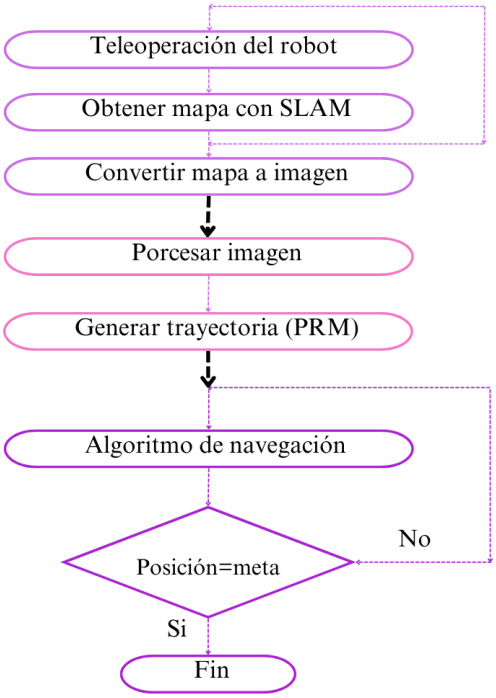


Figura 5.1: Navegación *offline*

5.1. Localización y Mapeo Simultáneo (SLAM)

Para dotar al *Turtlebot3* con la habilidad de generar un mapa del entorno en el que se desplaza el robot, se optó por la implementación del algoritmo SLAM. Este algoritmo permite la generación progresiva del mapa a medida que el robot navega en un entorno, El método de SLAM empleado es *Gmapping* el cual está basado en láser, ROS cuenta con el paquete *Gmapping* de *OpenSlam*, el cual proporciona un nodo llamado *slam_gmapping*. A partir de datos láser y de la posición recopilados por un robot móvil, se generó con *slam_gmapping* un mapa de ocupación en 2D [80].

5.1.1. Mapeo

El mapa se obtuvo realizando un recorrido teleoperado con el robot *Turtlebot 3* utilizando un teclado, en el pseudocódigo 1 y en la Figura 5.2 se detalla como obtener el mapa.

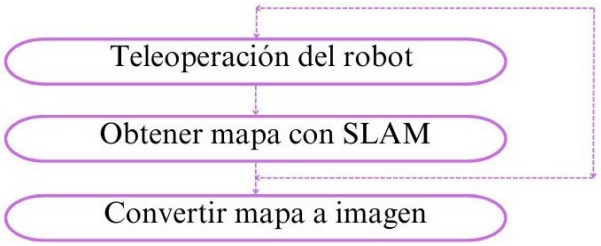


Figura 5.2: Diagrama secuencial del proceso de mapeo.

Algoritmo 1 Pseudocódigo para el mapeo

```
1: función SLAM
2:     mientras Ejecución hacer:
3:         Mover robot (Teleoperado)
4:         Generar mapa a partir de la nueva posición (SLAM)
5:     fin mientras
6:     Convertir mapa a imagen
7: fin función
```

Partiendo del pseudocódigo 1 a continuación se presentan los pasos necesarios para realizar el mapeo en ROS, así como los comandos correspondientes para cada paso: Partiendo del pseudocódigo 1, a continuación se presentan los pasos necesarios para realizar el mapeo en ROS, así como los comandos correspondientes para cada paso:

- **Ejecución del entorno junto con *Gmapping*:** El archivo de lanzamiento correspondiente se puede consultar en el siguiente enlace: https://github.com/itzchav/E

ntorno-Division-Estudios-Posgrado/blob/main/src/turtlebot3_simulations/turtlebot3_gazebo/launch/plano_completo_gmapping.launch o en el Apéndice C.2. Para iniciar el entorno con Gmapping, se debe ejecutar el siguiente comando:

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo plano_completo.launch

```

- **Teleoperación del robot:** Para teleoperar el robot, se debe ejecutar el comando nativo de ROS (ver Figura 5.3):

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

```

- **Guardado del mapa generado:** Una vez generado el mapa completo, se debe ejecutar el siguiente comando para guardar la imagen del mapa:

```
rosrun map_server map_saver -f map

```

- **(Opcional) Binarización de la imagen del mapa:** Si se desea obtener una imagen binarizada del mapa, se puede utilizar el código disponible en el siguiente enlace: https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/codigos_offline/Rviz_binarizado.py, que permite obtener la imagen y binarizarla.

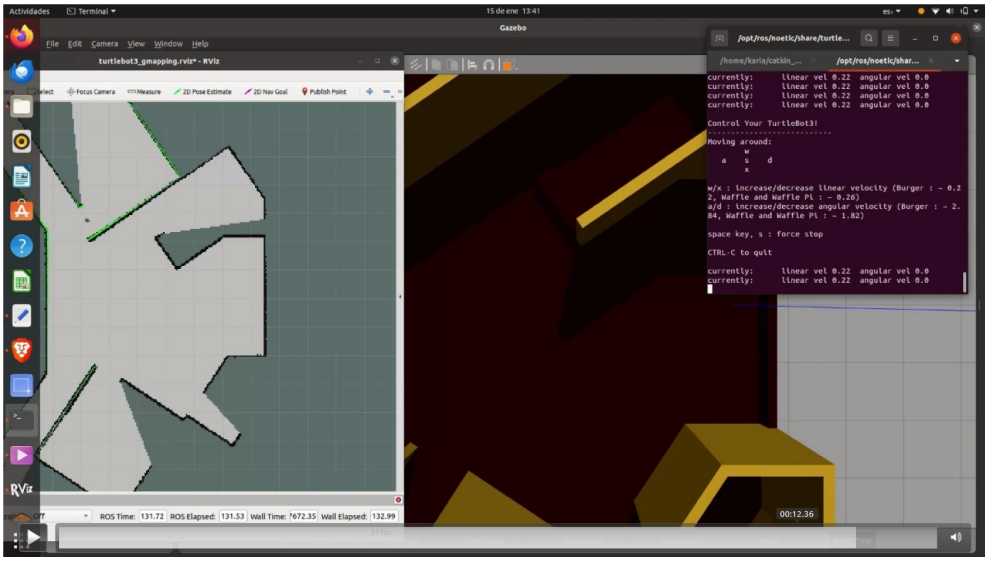


Figura 5.3: Teleoperación del robot para obtener el mapa con SLAM.

5.1.2. Mapas obtenidos con SLAM

A continuación se presenta el resultado del mapeo empleando el pseudocódigo 1 en el entorno sin obstáculos (Figura 5.4). El código de lanzamiento del entorno con *Gmapping* puede consultarse en el siguiente enlace: https://github.com/itzchav/Entorno-Division-Estudios-Posgrado/blob/main/src/turtlebot3_simulations/turtlebot3_gazebo/launch/plano_completo_gmapping.launch.

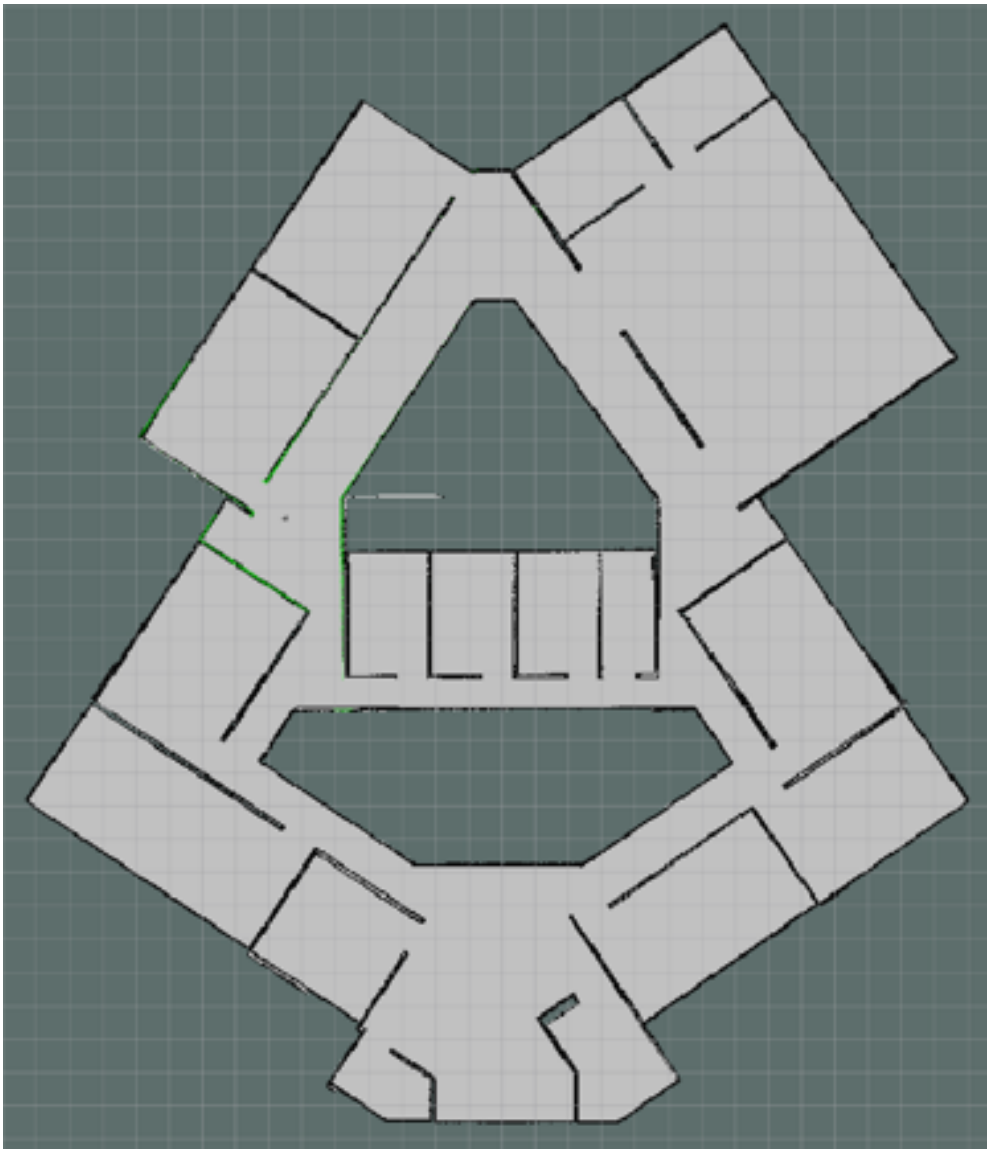


Figura 5.4: Mapa obtenido con SLAM mostrado en RVIZ.

De manera análoga, se efectuó un recorrido en un entorno con obstáculos, y el mapa obtenido se presenta en la Figura 5.5, el código del entorno con obstáculos, ejecutando *Gmapping* se se puede consultar en el siguiente enlace: https://github.com/itzchav/Entorno-Division-Estudios-Posgrado/blob/main/src/turtlebot3_simulations/turtlebot3_gazebo/launch/plano_completo_obstaculos_noetic.launch.

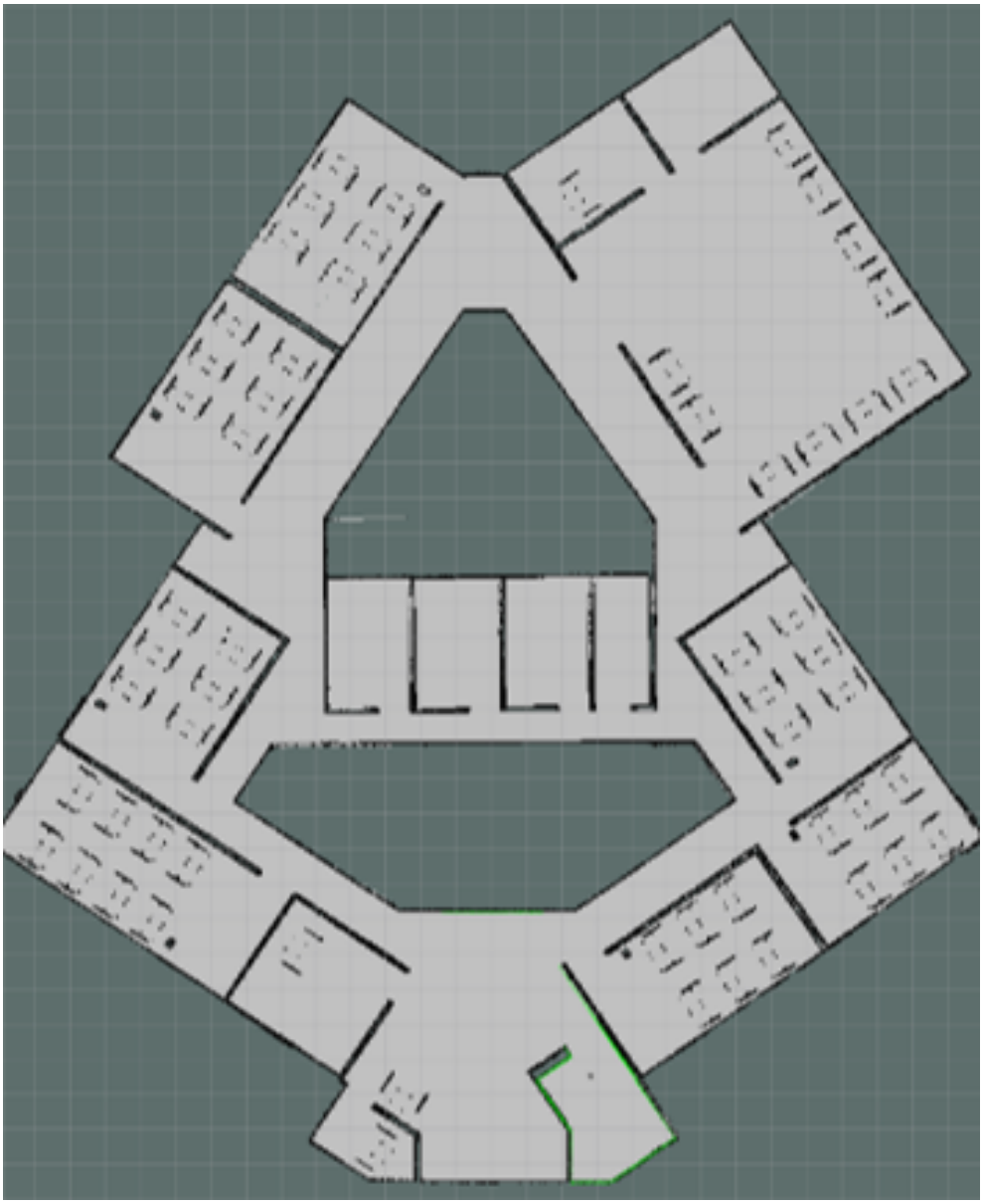


Figura 5.5: Mapa con obstáculos obtenido con SLAM mostrado en RVIZ.

El resultado del mapa del entorno sin obstáculos binarizado, empleando el código del enlace https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/codigos_offline/Rviz_binarizado.py se muestra en la Figura 5.6.

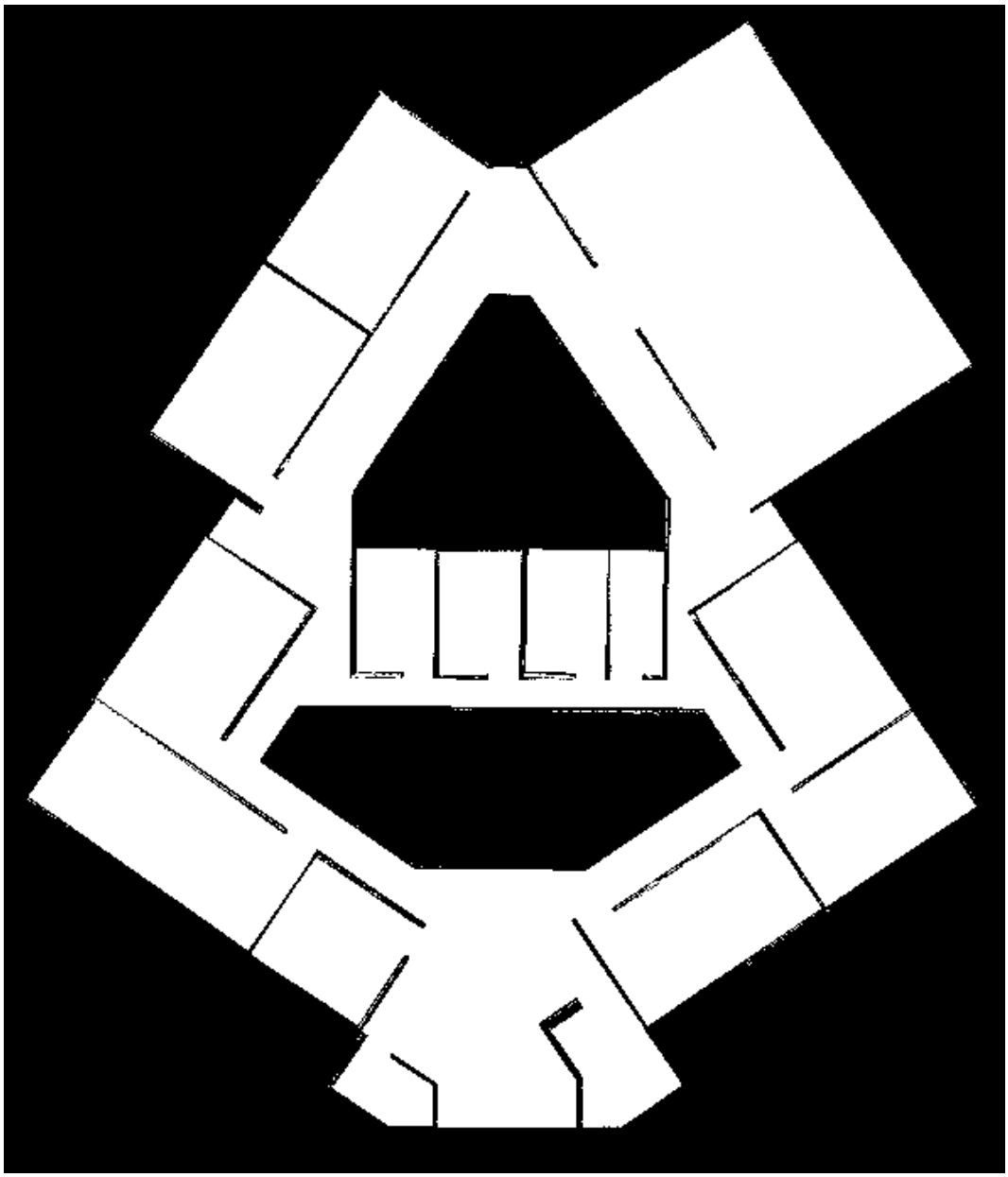


Figura 5.6: Mapa binarizado obtenido con SLAM.

El resultado del mapa del entorno con obstáculos binarizado, empleando el código del enlace https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/codigos_offline/Rviz_binarizado.py se muestra en la Figura 5.7.

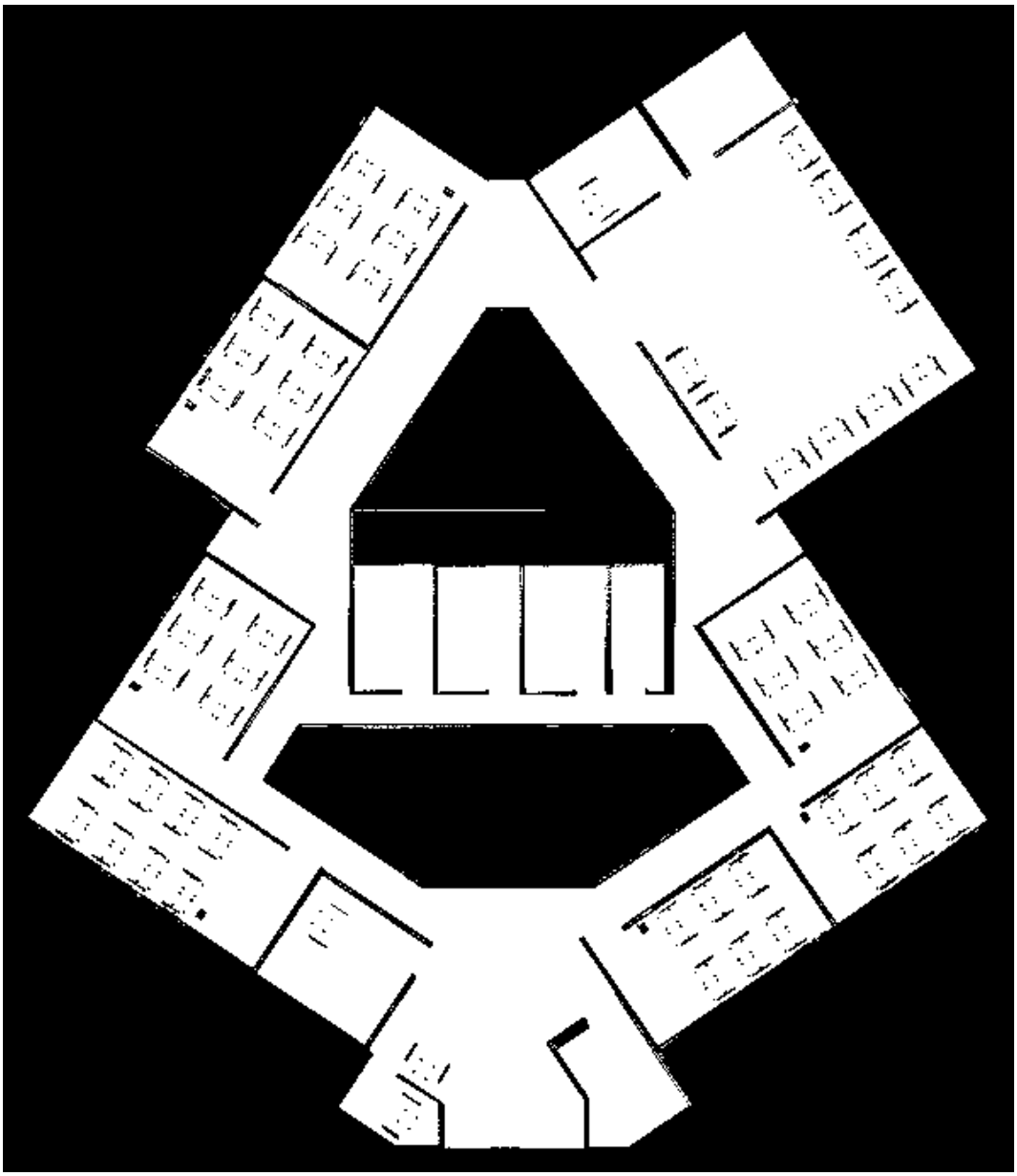


Figura 5.7: Mapa con obstáculos binarizado obtenido con SLAM.

5.2. PRM

Una vez obtenido el mapa con SLAM, es necesario generar la trayectoria. Para ello, se procesa la imagen, se binariza y se dilatan los elementos que representan obstáculos, estableciendo una distancia de seguridad en píxeles para evitar colisiones (ver pseudocódigo 2). Posteriormente, se emplea PRM para generar una trayectoria (ver pseudocódigo 11), y finalmente el vector de píxeles obtenido con PRM se convierte a metros. Para convertir los píxeles a metros, se utiliza la ecuación 5.1. El proceso completo se ilustra en la Figura 5.9.

$$resolucion\ x = \frac{(max\ pixel\ x - min\ pixel\ x)}{distnacia\ entre\ pixeles} \tag{5.1}$$

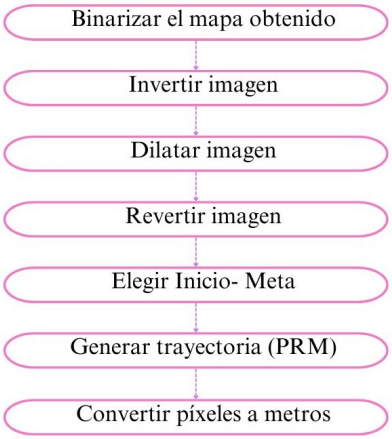


Figura 5.8: Diagrama de flujo PRM.

Algoritmo 2 Pseudocódigo procesamiento de imagen

```
1: función ProcesarImagen(Imagen de RVIZ)
2:     Imagen binarizada = Binarizar(Imagen de RVIZ)
3:     Imagen invertida = Invertir(Imagen binarizada)
4:     Imagen dilatada = Dilatar(Imagen invertida)
5:     Imagen final = Invertir(Imagen dilatada)
6:     return Imagen final
7: fin función
```

Algoritmo 3 Pseudocódigo PRM

```
1: función PRM(Imagen procesada, Inicio, Meta, NumMuestras, Vecinos)
2:     Muestras = GenerarMuestrasAleatorias(NumMuestras)
3:     Grafo = ConstruirGrafo(Muestras, Vecinos)
4:     Camino = BuscarCamino(Grafo, Inicio, Meta)
5:     Vector de posiciones = ConvertirPixelesAMetros(Camino)
6:     return Vector de posiciones
7: fin función
```

El código para obtener la trayectoria en el entorno sin obstáculos se puede consultar en https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/codigos_offline/Prm_code.py, así mismo para el entorno con obstáculos se puede consultar en el siguiente enlace https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/codigos_offline/Prm_code_obstaculos.py. Los parámetros que se deben de modificar para generar una trayectoria son los siguientes:

Inicio en y	Destino en y	Nodos
Inicio en x	Destino en x	Distancia

Tabla 5.1: Parámetros para el PRM

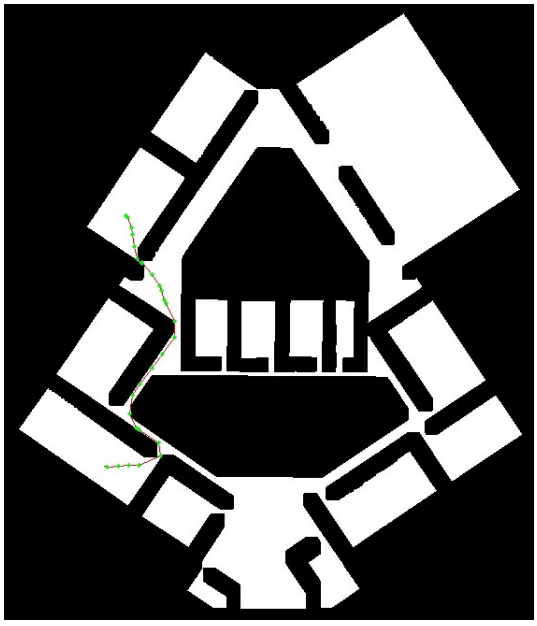


Figura 5.9: Trayectoria generada.

5.3. PRM-SLAM *offline*

Una vez obtenida la trayectoria en metros, se modificó el algoritmo de navegación de manera que, al llegar a un punto deseado, se asigna como destino el siguiente punto de la trayectoria de forma iterativa, hasta alcanzar el final de la misma. El proceso de PRM-SLAM *offline*, utilizando el control cinemático, se muestra en la Figura 5.10.

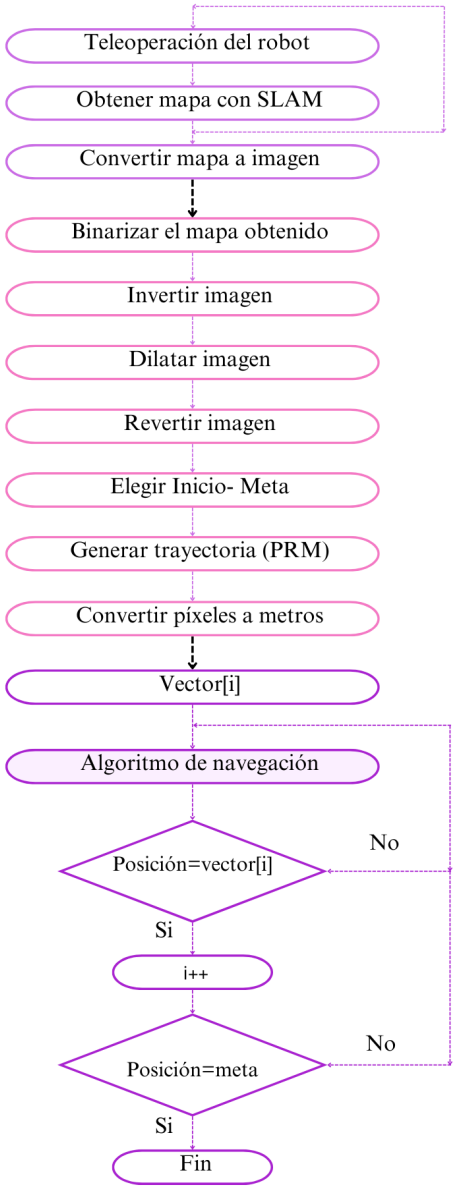


Figura 5.10: Diagrama PRM-SLAM.

5.3.1. PRM-SLAM control cinemático

Para realizar las pruebas, se modificó el control cinemático explicado en el Capítulo 3, de manera que, al llegar a un punto deseado, se asigna como destino el siguiente punto de la trayectoria. Esto se muestra en el pseudocódigo 4.

Algoritmo 4 Pseudocódigo control cinemático

```
1: función ControlCinemático(x, y, xhd, yhd)
2:     Definir: Ganancia (k), Punto de operación (h), Distancia (d)
3:     Asignar control (ux, uy)
4:     Calcular las velocidades del robot (v, w) mediante (3.18)
5:     return v, w
6: fin función
7: función RecorrerTrayectoria(x, y, ruta_x, ruta_y)
8:     mientras meta <> posición hacer
9:         si distancia control < d entonces
10:            si bandera < tamaño(xd) entonces
11:                ControlCinemático(x, y, ruta_x[i], ruta_y[i])
12:                i = i + 1
13:            fin si
14:        fin si
15:    fin mientras
16: fin función
```

En siguiente enlace se puede consultar el código del control cinemático adaptado para seguir trayectorias: https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/src/prm_slam/src/prm_slam_offline.py. Para ejecutarlo es necesario:

- Cambiar el vector de posiciones de acuerdo a la trayectoria obtenida con PRM, definida por la variable *path_metros*.
- Ejecutar en la terminal los siguientes comandos

```
cd ~/prm_slam_ws/
source devel/setup.bash
rosrun prm_slam prm_slam_offline.py
```

5.3.2. PRM-SLAM-RL

Basado en el trabajo de Cimurs et al. [73], se realizaron adaptaciones para entrenar el modelo que permita al *Turtlebot 3* llegar a una meta y evadir obstáculos, así como las modificaciones

necesarias para poder realizar las pruebas en el entorno simulado de la División de Estudios de Posgrado. Estas adaptaciones se describen en el Capítulo 4. Para realizar las pruebas con RL, es necesario seguir los siguientes pasos:

- Ejecutar los siguientes comandos

```
$ export ROS_HOSTNAME=localhost
$ export ROS_MASTER_URI=http://localhost:11311
$ export ROS_PORT_SIM=11311
$ export GAZEBO_RESOURCE_PATH=~/DRL-robot-navigation/catkin_ws/src/
    multi_robot_scenario/launch
$ source ~/.bashrc
$ cd ~/DRL-robot-navigation/catkin_ws
$ source devel_isolated/setup.bash
```

- Modificar el punto de inicio del robot en el entorno. (Se realiza la modificación de x2 y y2 en el código del entorno en el que se desea ejecutar https://github.com/itzchav/Navegacion-SLAM-RL/tree/main/TD3/assets, por ejemplo para el plano con obstáculos se modifica el código del siguiente enlace: https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/assets/plano_completo_obstaculos_rviz.launch

- Modificar en el código (https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/test_plano.py) la línea que importa el entorno de RL(https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/plano_env.py)

```

    from plano_env import GazeboEnv

```

  La línea que exporta el entorno de Gazebo.

```

    env = GazeboEnv("plano_completo_obstaculos_rviz.launch",
    environment_dim)

```

- Se cambia la trayectoria generada con PRM, para ello se modifica el *goal_list* en el código https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/plano_env.py.

- Finalmente se ejecuta el código de prueba https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/test_plano.py

```

    python3 test_plano.py

```

5.3.3. Pruebas PRM-SLAM *offline*

Se realizaron tres pruebas para evaluar el comportamiento del robot en función del algoritmo de navegación. En la Prueba 1 se empleó el control cinemático utilizando la trayectoria obtenida con PRM. En la Prueba 2 se analizó lo qué ocurría cuando los puntos no se proporcionaban correctamente al robot. Finalmente, en la Prueba 3 se evaluó cómo se comportó el robot usando el modelo de RL ante perturbaciones en los datos de la trayectoria. Para observar el comportamiento del robot, se determinó una meta a menos de 10 metros (Figura 5.11). El video de las pruebas realizadas se puede consultar en https://www.youtube.com/watch?v=P-nvfPDxhXI

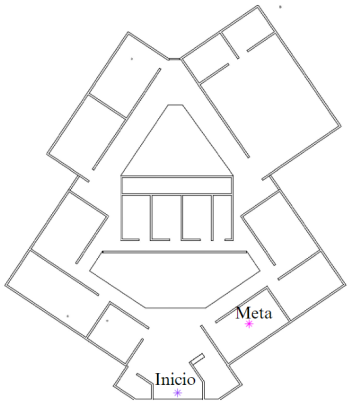


Figura 5.11: Inicio y meta de las pruebas.

Para generar la trayectoria se emplearon los parámetros presentados en la Tabla 5.2, obteniendo así la ruta que se puede observar en la Figura 5.12. Los puntos que se obtuvieron con PRM convertidos a metros, se muestran en la Tabla 5.3.

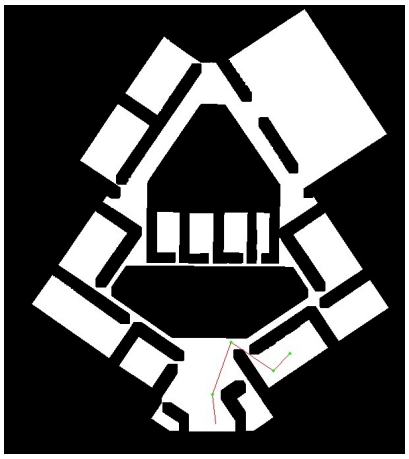


Figura 5.12: Trayectoria generada.

Tabla 5.2: Parámetros para el PRM Prueba 1 (5 puntos)

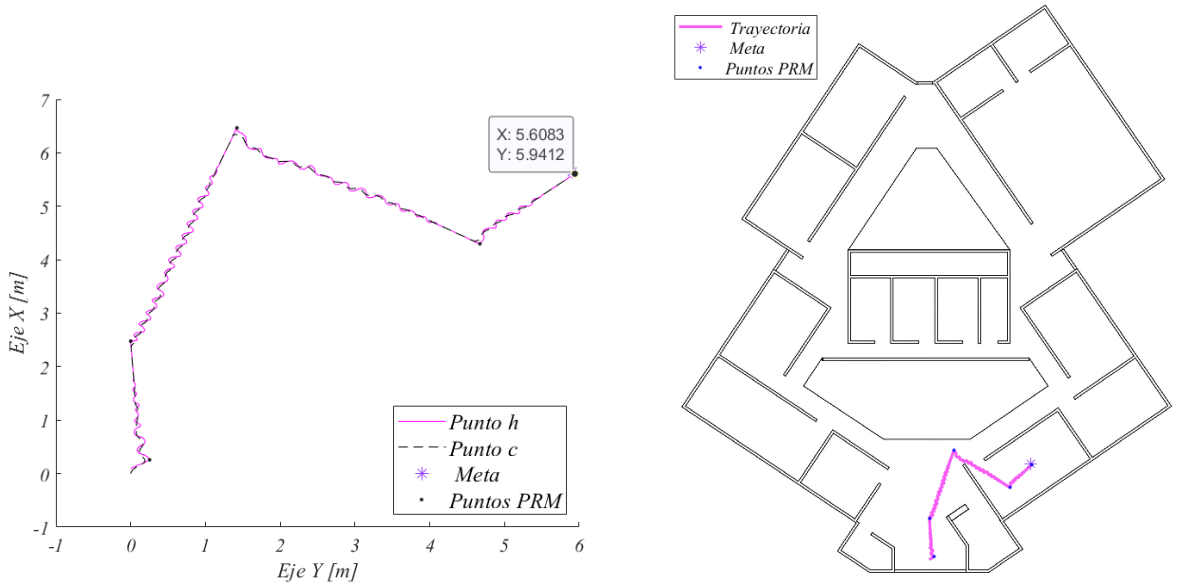
Punto de Inicio (y, x)	**Punto de Destino (y, x)**	**Nodos**	**Distancia (pixeles)**
(692, 430)	(547, 580)	300	100

Tabla 5.3: Vector de posiciones de la Prueba 1 (5 puntos)

Coordenadas x	**Coordenadas y**
0.258060035	-0.253895536
2.478921303	0
6.466376761	-1.421815
4.295989613	-4.671677857
5.608316725	-5.941155536

Prueba 1

Se realizó la primera prueba empleando la trayectoria obtenida con PRM. La trayectoria seguida por el robot se muestra en la Figura 5.13a, en la cual se observa el recorrido medido desde el centro del robot y desde el punto de control (punto H). Por otro lado, en la Figura 5.13b se muestra la trayectoria recorrida en el entorno.



(a) (b)

Figura 5.13: Resultados de la prueba 1. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

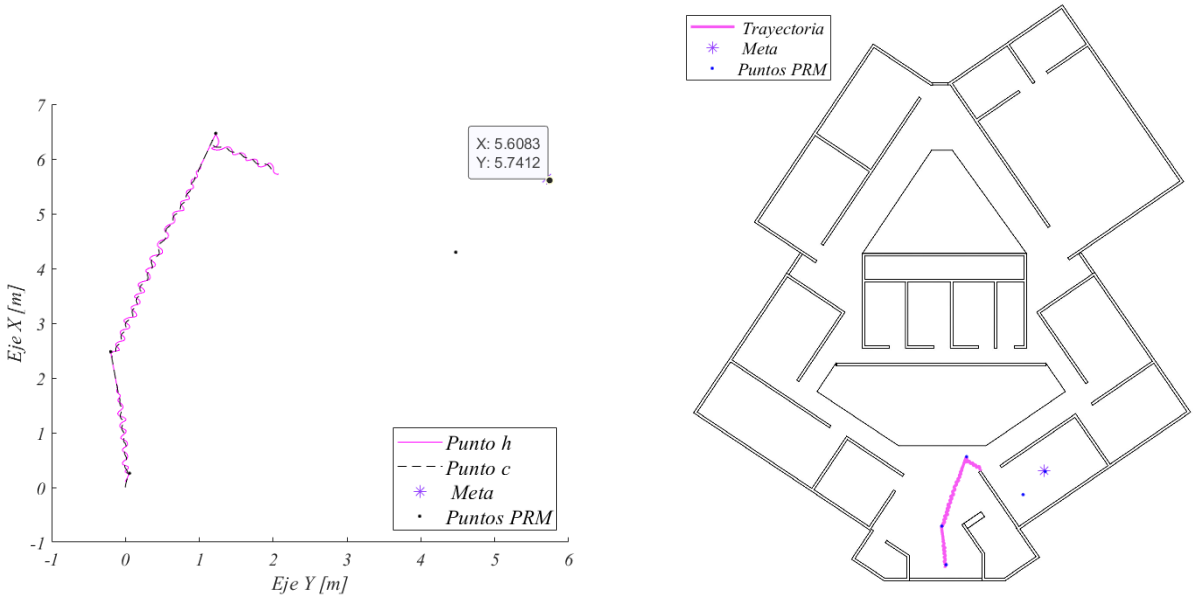
Prueba 2

Se realizó una segunda prueba desplazando los puntos a 0.2m en y para observar el comportamiento del robot, en caso de que haya un error al convertir los pixeles a metros. La distancia se eligió considerando que entre dos puntos de la trayectoria podría haber un obstáculo. En la Tabla 5.4 se muestran los nuevos puntos de la trayectoria.

Tabla 5.4: Vector de posiciones de la Prueba 2 (5 puntos)

Coordenadas x	**Coordenadas y**
0.258060035	-0.053895536
2.478921303	0.2
6.466376761	-1.221815
4.295989613	-4.471677857
5.608316725	-5.741155536

La trayectoria del centro del robot y del punto de control (punto h) se muestra en la Figura 5.14a. Se puede observar que no llega a la meta, en la Figura 5.14b la gráfica de la trayectoria con respecto al entorno, se observa que el robot colisiona.

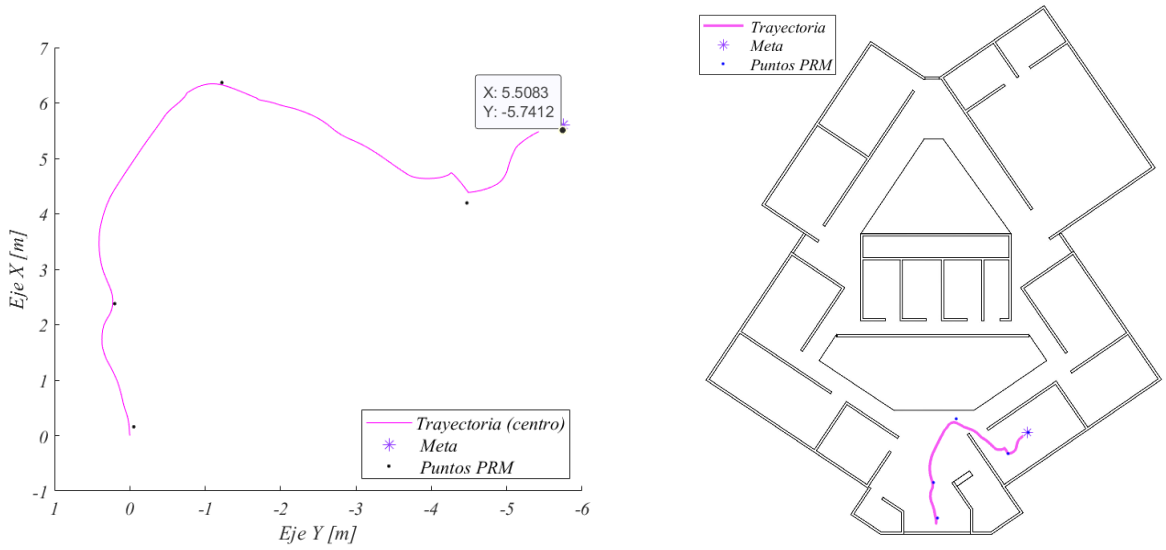


(a) (b)

Figura 5.14: Resultados de la prueba 2. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

Prueba 3

Como alternativa, ante la colisión del robot, una vez que se ha obtenido el mapa del entorno y la trayectoria, se cambia el algoritmo de navegación, el control cinemático es sustituido por el modelo de RL. En este caso el modelo esta entrenado para evitar colisionar y llegar a un punto deseado, por lo que se espera que la trayectoria que tiene objetos intermedios logre llegar a la meta. Se utilizaron los puntos de la Prueba 2 para llegar al Aula 1, los cuales se presentan en la Tabla 5.4. En la Figura 5.14b se observa que con el control cinemático el robot colisiona, al emplear el modelo de RL el robot evade el obstáculo con el cual el robot colisionó utilizando el control cinemático (Figura 5.15b).



(a) (b)

Figura 5.15: Resultados de la prueba 3. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

5.4. Evaluación del Modelo

Se realizaron cuatro pruebas, tomando en cuenta la distribución de los espacios en el edificio, como se muestra en la Figura 4.1. La primera prueba inicia en la entrada del edificio y se dirige al Aula 1, con una distancia menor a 10 m. La segunda prueba comienza en el Aula 4 y termina en el Aula 6, con una distancia entre 10 y 20 m. Finalmente, se llevaron a cabo dos pruebas a una distancia mayor a 20 m: la primera, desde la entrada hasta la parte trasera del edificio, y la segunda, desde el Aula 2 hasta el Aula 7, cuya trayectoria pasa por los sanitarios y resulta ser más compleja (Figura 5.16). El video de las pruebas se puede consultar en https://www.youtube.com/watch?v=1TfGGCmBoH4.

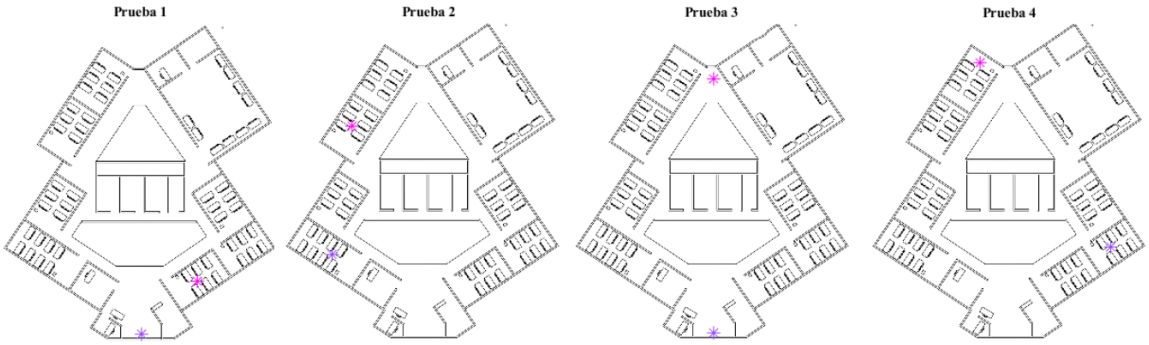


Figura 5.16: Puntos de inicio y final de las pruebas para evaluación

En la Tabla 5.5 se concentra la información de la posición inicial del robot y el destino de cada prueba. Se realizó una tabla para evaluar el rendimiento del modelo en términos de su capacidad para completar trayectorias y alcanzar los puntos asignados. Se evalúan la cantidad de 'Pruebas' exitosas en relación con el total de pruebas realizadas. Esto permite analizar la frecuencia con la que el modelo alcanza la meta sin fallos. También se evalúan los 'Puntos Totales', que indican la cantidad de puntos que el modelo debería cubrir en cada trayectoria, evaluando cuántos puntos realmente recorre. Finalmente, se evalúa cuántos 'Puntos Dados' de la trayectoria representan los puntos efectivos asignados al modelo. Si el modelo colisiona en alguna trayectoria, los puntos restantes no se le asignan, lo que permite evaluar así la precisión y el control del modelo en el recorrido.

Tabla 5.5: Pruebas realizadas

Prueba	**Posición Inicial (x, y)**	**Inicio**	**Destino**	**Distancia (m)**
Prueba 1	(32.700508, -8.362269)	Entrada	Aula 1	< 10
Prueba 2	(24.666824, -18.216396)	Aula 4	Aula 6	$10 - 20$
Prueba 3	(32.700508, -8.362269)	Entrada	Parte trasera	> 20
Prueba 4	(23.642405, 2.257198)	Aula 2	Aula 7	> 20

5.4.1. Prueba 1 (Meta menor a 10m)

Se determinó la trayectoria en el entorno con obstáculos (Figura 5.17) y se evaluó el rendimiento del modelo a una distancia menor a 10 m con una trayectoria de 15 puntos generada con 2500 nodos, y con un número de vecinos más cercanos de 200, los puntos se muestran en la Tabla 5.7. La distancia euclidiana entre el punto inicial y la meta es de 8.6 metros.

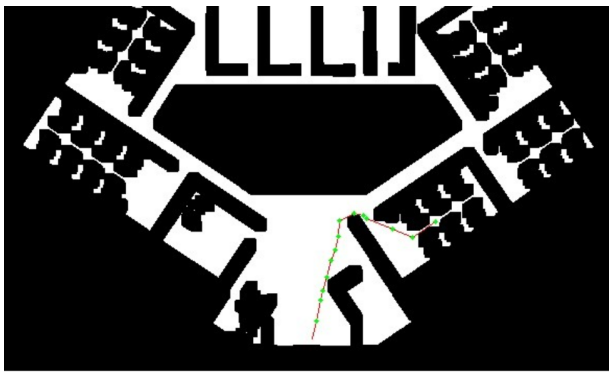


Figura 5.17: Trayectoria obtenida con PRM (Prueba 1)

Punto de Inicio (y, x)	**Punto de Destino (y, x)**	**Nodos**	**Distancia (pixeles)**
(692, 430)	(547, 580)	300	100

Tabla 5.6: Parámetros para el PRM

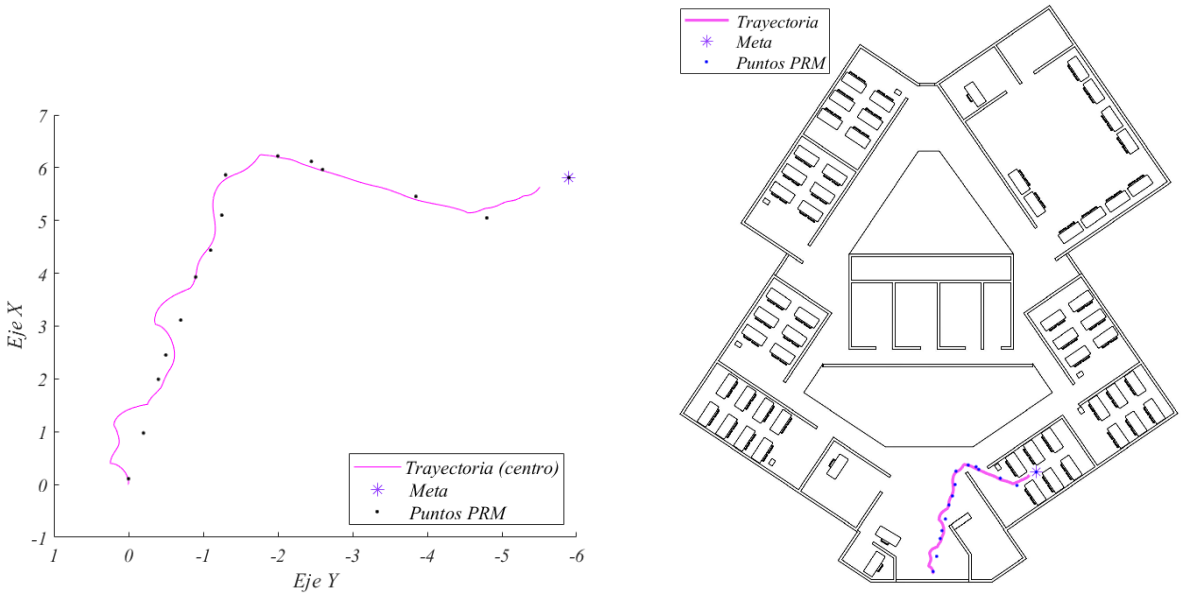
Tabla 5.7: Trayectoria de la Prueba 1 (Meta menor a 10m)

	X	**Y**
1	-0.090776	0.200000
2	0.774904	0.000096
3	1.793352	-0.199807
4	2.251653	-0.299759
5	2.913644	-0.499663
6	3.728402	-0.699567
7	4.237626	-0.899470
8	4.899617	-1.049398

	X	**Y**
9	5.663453	-1.099374
10	6.019909	-1.799037
11	5.918065	-2.248820
12	5.765298	-2.398748
13	5.256074	-3.648146
14	4.848695	-4.597689
15	5.612530	-5.697159

Entorno con obstáculos.

En la Figura 5.18a se presentan los puntos de la trayectoria, detallados en la Tabla 5.7, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.18b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en su entorno.



(a) (b)

Figura 5.18: Resultados de la prueba con obstáculos de evaluación 1. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

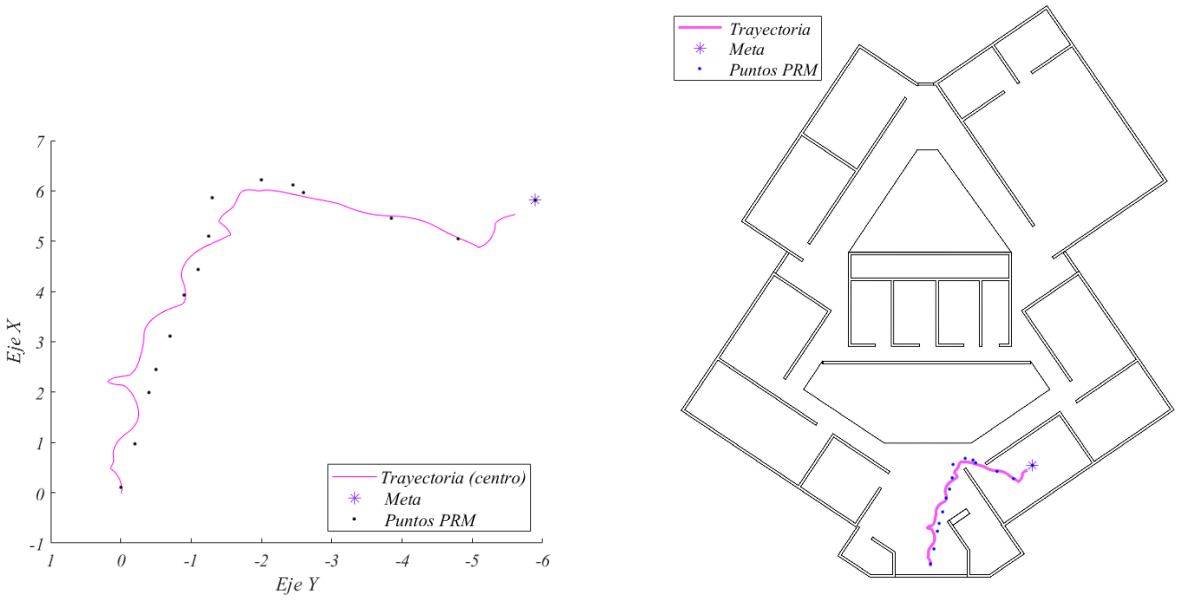
En el Anexo E.1 se muestra una Tabla E.1 con 30 pruebas realizadas. En la Tabla 5.8 se concentran los resultados del desempeño del modelo, que muestra que falló 1 vez, por lo que tuvo un 96.67 % de éxito. En la prueba en la que falló, colisionó en el punto 4, lo que le impidió ver los 11 puntos restantes. Ante esto, logró un 97 % de los 450 puntos de las 30 trayectorias. Finalmente, se evaluó el desempeño con respecto a los 440 puntos totales que realmente se le asignaron al robot, considerando los 11 puntos que no se le asignaron debido a la colisión, teniendo un 99.77 % de éxito.

Tabla 5.8: Desempeño del modelo en la Prueba 1 (con obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	29	1	96.67
Puntos Totales	450	439	11	97.78
Puntos Dados	440	439	1	99.77

Entorno sin obstáculos.

En la Figura 5.19a se presentan los puntos de la trayectoria en el entorno sin obstáculos, detallados en la Tabla 5.7, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.22b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en su entorno.



(a) (b)

Figura 5.19: Resultados de la prueba sin obstáculos de evaluación 1. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

En el Anexo E.1 se muestra una Tabla E.2 con 30 pruebas realizadas. En la Tabla 5.9 se concentran los resultados del desempeño del modelo en el entorno sin obstáculos. El modelo falló 6 veces, logrando un 80 % de éxito. En las pruebas en las que colisionó, no se le asignaron los 57 puntos restantes, logrando un 88.67 % de los 450 puntos de las 30 trayectorias. Finalmente, se evaluó el desempeño con respecto a los 440 puntos totales que realmente se le asignaron al robot, considerando los 57 puntos que no se le asignaron debido a las colisiones, por lo que en realidad el modelo tuvo un 98.5 % de éxito.

Tabla 5.9: Desempeño del modelo en la Prueba 1 (sin obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	24	6	80
Puntos Totales	450	393	57	88.67
Puntos Dados	399	393	6	98.50

Se puede observar que el robot tuvo un mejor desempeño en el entorno con obstáculos. Esto puede deberse tanto a la influencia de los obstáculos como a la dirección con la que el robot se aproxima a la meta.

5.4.2. Prueba 2 (Meta mayor a 10m)

Se determinó la trayectoria en el entorno con obstáculos (Figura 5.20) y se evaluó el rendimiento del modelo a una distancia euclidiana de 13.94 m, con una trayectoria de 26 puntos generada con 2600 nodos, y con un número de vecinos más cercanos de 200, los puntos se muestran en la Tabla 5.11.

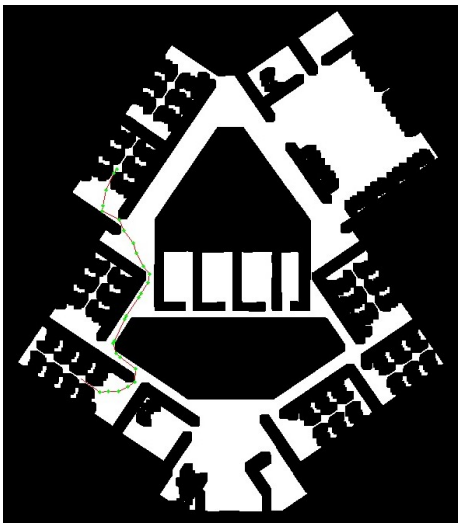


Figura 5.20: Trayectoria obtenida con PRM (Prueba 2)

Tabla 5.10: Parámetros para el PRM (Prueba 2)

Punto de Inicio (y, x)	**Punto de Destino (y, x)**	**Nodos**	**Distancia (pixeles)**
(527, 235)	(257, 281)	2600	200

Entorno con obstáculos.

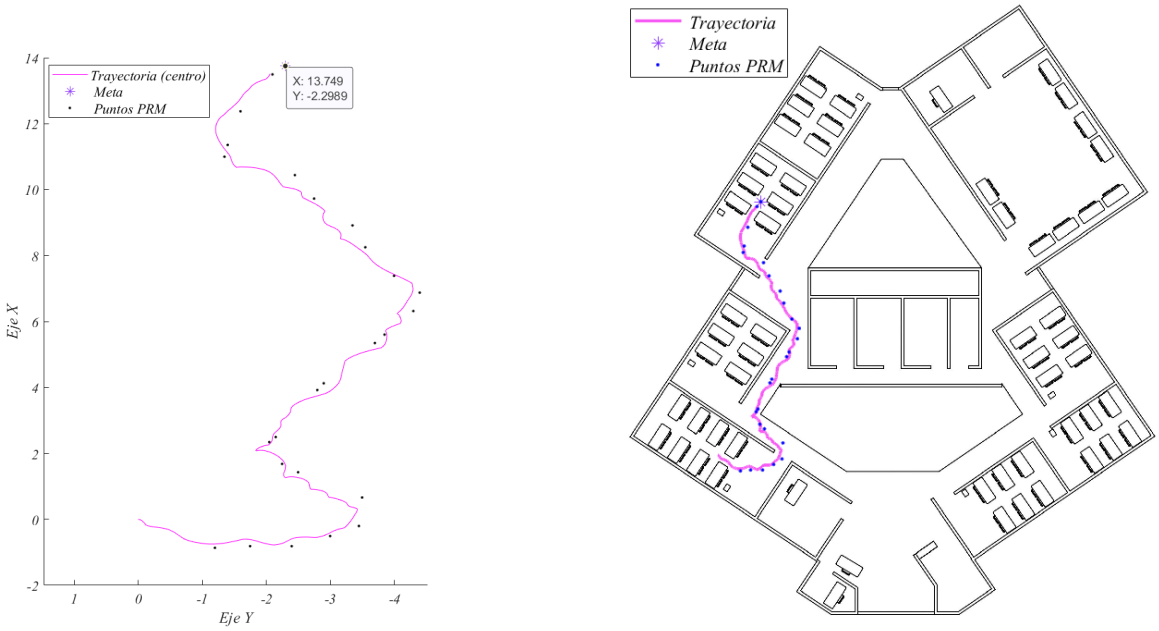
En la Figura 5.21a se presentan los puntos de la trayectoria, detallados en la Tabla 5.11, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.21b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en el entorno.

Tabla 5.11: Trayectoria de la Prueba 2 (Meta mayor a 10m)

	X	Y
1	-0.765680	-1.099422
2	-0.714758	-1.649157
3	-0.714758	-2.298844
4	-0.409224	-2.898555
5	-0.103690	-3.548339
6	0.761991	-3.598315
7	1.525827	-2.398796
8	1.780439	-2.148917
9	2.442429	-1.949013

	X	Y
10	2.595197	-2.048965
11	4.021023	-2.698652
12	4.224713	-2.798604
13	5.446850	-3.598218
14	5.701462	-3.748146
15	6.414375	-4.197929
16	6.974521	-4.297881
17	7.483745	-3.998074
18	8.349426	-3.548291

	X	Y
19	9.011417	-3.248387
20	9.826175	-2.848676
21	10.539088	-2.448820
22	11.099234	-1.249350
23	11.455691	-1.299326
24	12.474138	-1.499230
25	13.594431	-1.998989
26	13.949043	-2.198892



(a) (b)

Figura 5.21: Resultados de la prueba con obstáculos de evaluación 2. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

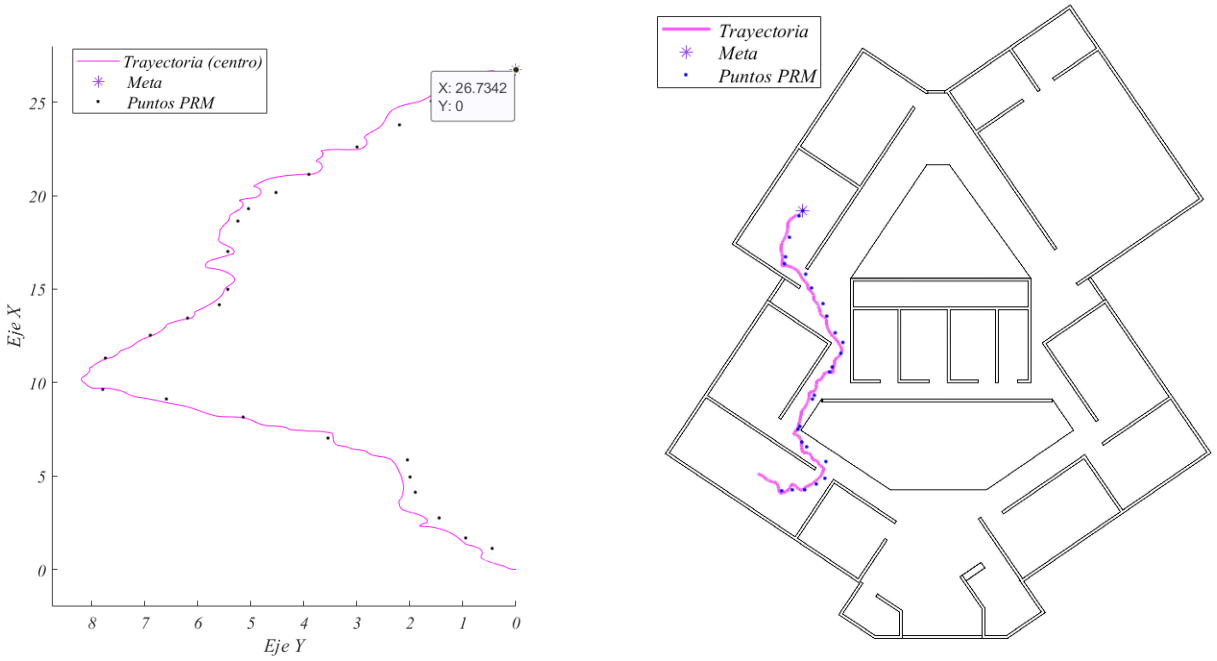
En el Anexo E.1 se muestra una Tabla E.3 con 30 pruebas realizadas. En la Tabla 5.12 se concentran los resultados del desempeño del modelo. El modelo falló 16 veces, logrando un 46.67 % de éxito. En las pruebas en las que colisionó, no se le asignaron los 201 puntos restantes, logrando un 76.28 % de los 780 puntos de las 30 trayectorias. El modelo tuvo un 97.31 % de éxito respecto a los 595 puntos totales que realmente se le asignaron al robot, considerando los 201 puntos que no se le asignaron debido a las colisiones.

Tabla 5.12: Desempeño del modelo en la Prueba 2 (con obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	14	16	46.67
Puntos Totales	780	579	201	76.28
Puntos Dados	595	579	16	97.31

Entorno sin obstáculos.

En la Figura 5.19a se presentan los puntos de la trayectoria en el entorno sin obstáculos, detallados en la Tabla 5.7, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.22b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en el entorno.



(a) (b)

Figura 5.22: Resultados de la prueba sin obstáculos de evaluación 2. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

En el Anexo E.1 se muestra una Tabla E.4 con 30 pruebas realizadas. En la Tabla 5.13 se concentran los resultados del desempeño del modelo. El modelo falló 15 veces, logrando un 50 % de éxito. En las pruebas en las que colisionó, no se le asignaron los 174 puntos restantes, logrando un 79.62 % de los 780 puntos de las 30 trayectorias. El modelo tuvo un 97.58 % de éxito con respecto a los 621 puntos totales que realmente se le asignaron al robot,

considerando los 174 puntos que no se le asignaron debido a las colisiones.

Tabla 5.13: Desempeño del modelo en la Prueba 2 (sin obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	15	15	50.00
Puntos Totales	780	606	174	79.62
Puntos Dados	621	606	15	97.58

5.4.3. Prueba 3 (Meta mayor a 20m)

Se determinó la trayectoria en el entorno con obstáculos (Figura 5.26) y se evaluó el rendimiento del modelo a una distancia euclidiana de 26.73 metros, con una trayectoria de 26 puntos generada con 1200 nodos, y con un número de vecinos más cercanos de 500, los puntos se muestran en la Tabla 5.14.

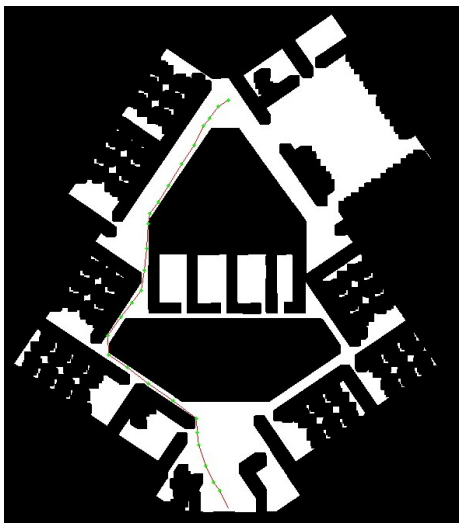


Figura 5.23: Trayectoria obtenida con PRM (Prueba 3)

Entorno con obstáculos

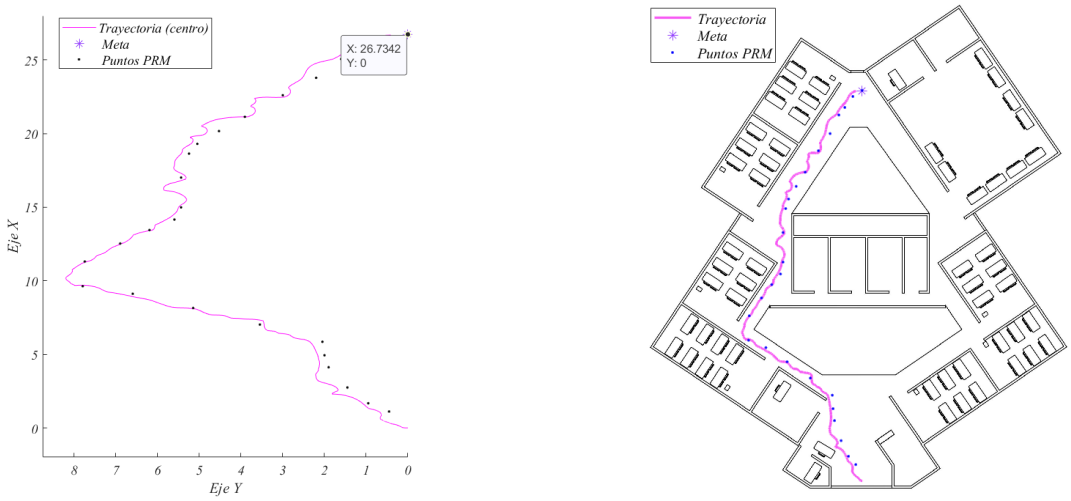
En la Figura 5.24a se presentan los puntos de la trayectoria, detallados en la Tabla 5.14, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.24b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en el entorno.

Tabla 5.14: Trayectoria de la Prueba 3 (Meta mayor a 20m)

	X	Y
1	1.2203	0.4497
2	1.7804	0.9495
3	2.8498	1.4493
4	4.2247	1.8991
5	5.0395	1.9990
6	5.9561	2.0490
7	7.1273	3.5483
8	8.2476	5.1475
9	9.2170	6.5969

	X	Y
10	9.7317	7.7963
11	11.4048	7.7463
12	12.6269	6.8967
13	13.5435	6.1970
14	14.2583	5.5973
15	15.0822	5.4374
16	17.1081	5.4375
17	18.7376	5.2475
18	19.3996	5.0476

	X	Y
19	20.2634	4.5278
20	21.2328	3.9081
21	22.6986	2.9986
22	23.8808	2.1989
23	25.1538	1.5992
24	25.6630	1.1994
25	26.4269	0.6497
26	26.8342	0.0000



(a) (b)

Figura 5.24: Resultados de la prueba con obstáculos de evaluación 3. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

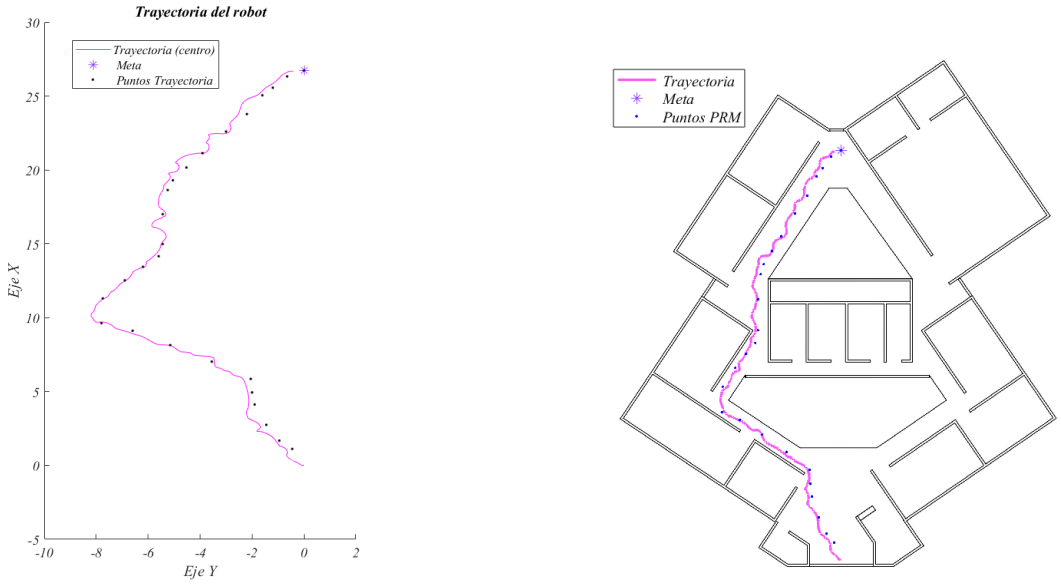
En el Anexo E.1 se muestra una Tabla E.5 con 30 pruebas realizadas. En la Tabla 5.15 se concentran los resultados del desempeño del modelo, el cual falló 14 veces, logrando un 53.33 % de éxito. En las pruebas en las que colisionó, no se le asignaron 262 puntos restantes, logrando un 69.38 % de los 810 puntos de las 30 trayectorias. El modelo tuvo un 97.51 % de éxito con respecto a los 562 puntos totales que realmente se le asignaron al robot, considerando los 262 puntos que no se le asignaron debido a las colisiones.

Tabla 5.15: Desempeño del robot en la Prueba 3 (con obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	16	14	53.33
Puntos Totales	810	548	262	69.38
Puntos Dados	562	548	14	97.51

Entorno sin obstáculos

En la Figura 5.25a se presentan los puntos de la trayectoria en el entorno sin obstáculos, detallados en la Tabla 5.14, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.25b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en el entorno.



(a) (b)

Figura 5.25: Resultados de la prueba sin obstáculos de evaluación 3. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

En el Anexo E.1 se muestra una Tabla E.6 con 30 pruebas realizadas. En la Tabla 5.16 se concentran los resultados del desempeño del modelo, el cual falló 14 veces, logrando un 53.33 % de éxito. En las pruebas en las que colisionó, no se le asignaron 262 puntos restantes, logrando un 69.38 % de los 810 puntos de las 30 trayectorias. El modelo tuvo un 97.51 % de éxito con respecto a los 562 puntos totales que realmente se le asignaron al robot, considerando los 262 puntos que no se le asignaron debido a las colisiones.

Tabla 5.16: Desempeño del robot en la Prueba 3 (sin obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	19	11	63.33
Puntos Totales	810	617	193	77.53
Puntos Dados	628	617	11	98.25

5.4.4. Prueba 4 (Meta mayor a 20m)

Se determinó la trayectoria en el entorno con obstáculos (Figura 5.26) y se evaluó el rendimiento del modelo a una distancia euclidiana de 23.85 metros, con una trayectoria de 41 puntos generada con 2600 nodos, y una distancia de 120, los puntos se muestran en la Tabla 5.14.

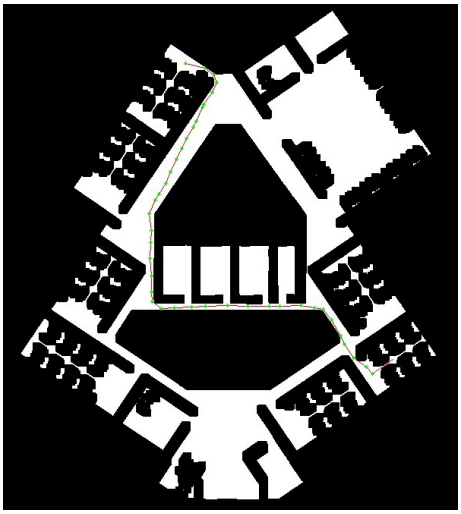


Figura 5.26: Trayectoria obtenida con PRM (Prueba 3)

Tabla 5.17: Parámetros para el PRM (Prueba 2)

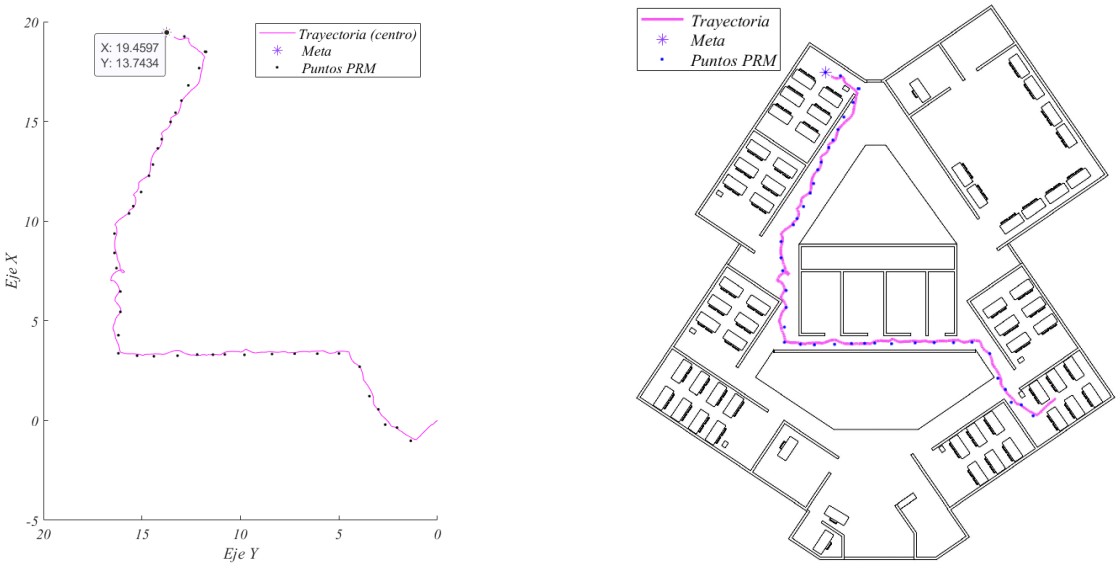
Punto de Inicio (y, x)	**Punto de Destino (y, x)**	**Nodos**	**Distancia (pixeles)**
(514, 643)	(124, 370)	2600	120

Entorno con obstáculos

En la Figura 5.27a se presentan los puntos de la trayectoria, detallados en la Tabla 5.18, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.27b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en el entorno.

Tabla 5.18: Trayectoria de la Prueba 4 (Meta mayor a 20m)

	X	Y
1	-0.917	1.249
2	-0.255	1.949
3	-0.102	2.549
4	0.662	2.899
5	1.324	3.348
6	2.797	3.848
7	3.463	4.898
8	3.453	6.000
9	3.452	7.147
10	3.432	8.296
11	3.394	9.695
12	3.413	10.695
13	3.403	11.295
14	3.408	12.094
15	3.353	13.094
16	3.323	14.293
17	3.353	15.143
18	3.473	16.092
19	4.379	16.092
20	5.551	15.992
21	6.569	15.992
22	7.740	16.192
23	8.504	16.292
24	9.472	16.292
25	10.486	15.553
26	10.846	15.343
27	11.559	14.943
28	12.374	14.543
29	12.934	14.343
30	13.749	14.093
31	14.207	13.893
32	15.073	13.444
33	15.531	13.194
34	16.142	12.894
35	16.906	12.544
36	17.772	11.994
37	18.587	11.644
38	18.594	11.694
39	19.357	12.744
40	19.560	13.643



(a) (b)

Figura 5.27: Resultados de la prueba con obstáculos de evaluación 4. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

En el Anexo E.1 se muestra una Tabla E.7 con 30 pruebas realizadas. En la Tabla E.7 se concentran los resultados del desempeño del modelo, el cual falló 17 veces, logrando un 43.33 % de éxito. En las pruebas en las que colisionó, no se le asignaron 416 puntos, logrando un 67.56 % de los 1230 puntos de las 30 trayectorias. El modelo tuvo un 97.95 % de éxito

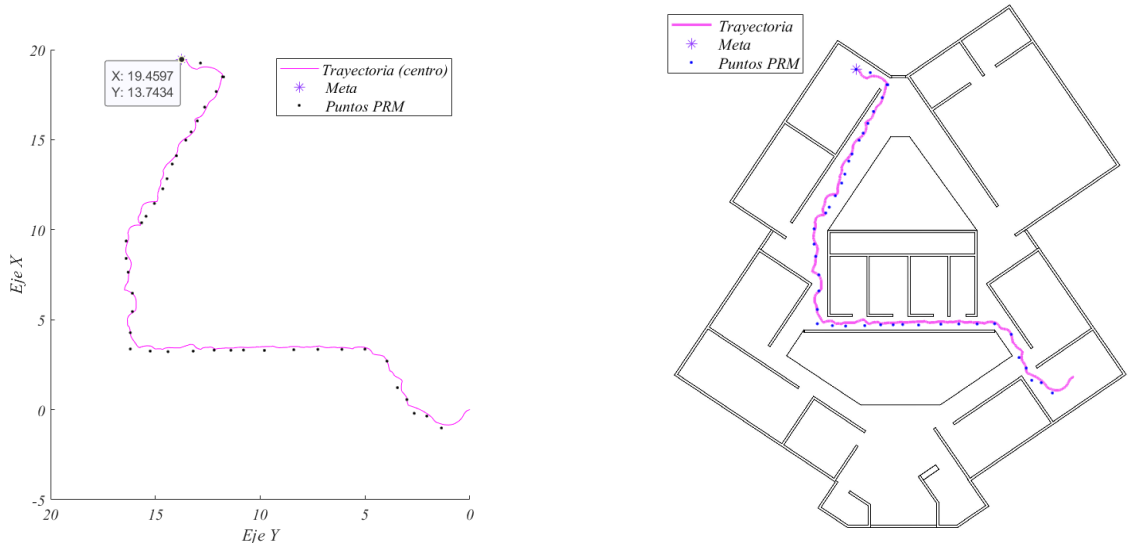
con respecto a los 831 puntos totales que realmente se le asignaron al robot, considerando los 416 puntos que no se le asignaron debido a las colisiones.

Tabla 5.19: Desempeño del robot en la Prueba 4 (con obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas	30	13	17	43.33
Puntos Totales	1230	814	416	67.56
Puntos Dados	831	814	17	97.95

Entorno sin obstáculos

En la Figura 5.28a se presentan los puntos de la trayectoria en el entorno sin obstáculos, detallados en la Tabla 5.18, así como la trayectoria seguida por el robot al implementar el modelo de RL. Asimismo, en la Figura 5.28b se pueden observar tanto los puntos de la trayectoria como el recorrido del robot en el entorno.



(a) (b)

Figura 5.28: Resultados de la prueba sin obstáculos de evaluación 4. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

En el Anexo E.1 se muestra una Tabla E.6 con 30 pruebas realizadas. En la Tabla 5.16 se concentran los resultados del desempeño del modelo, el cual falló 27 veces, logrando un 32.5 % de éxito. En las pruebas en las que colisionó, no se le asignaron 356 puntos, logrando un 73.25 % de los 1230 puntos de las 30 trayectorias. Finalmente, se evaluó el desempeño

con respecto a los 901 puntos totales que realmente se le asignaron al robot, considerando los 356 puntos que no se le asignaron debido a las colisiones. Así, el modelo tuvo un 97.00 % de éxito.

Tabla 5.20: Desempeño del robot en la Prueba 4 (sin obstáculos)

Métrica	**Total**	**Completados**	**Faltantes**	**Porcentaje (%)**
Pruebas Completas	40	13	27	32.50
Puntos Totales	1230	874	356	73.25
Puntos Dados	901	874	27	97.00

5.5. Conclusiones

En los experimentos realizados, se ha observado que un incremento en la distancia implica un incremento en la cantidad de puntos de la trayectoria (Figura 5.30), lo que implica que aumente la dificultad para llegar a una meta a mayor distancia.

En la Tabla 5.21 se presentan los resultados en porcentaje de las cuatro pruebas realizadas y el numero total de 'Puntos' de cada trayectoria. La columna de 'Pruebas Completas', registra la cantidad de pruebas exitosas en relación con el total de pruebas realizadas. Esto permite analizar la frecuencia con la que el modelo alcanza la meta sin fallos. También se evalúan los porcentaje de 'Puntos Totales', que indican la cantidad de puntos que el modelo debería cubrir en cada trayectoria, evaluando cuántos puntos realmente recorre. Finalmente, se evalúa el porcentaje de 'Puntos Dados' de la trayectoria, los cuales representan los puntos asignados como submetas al modelo.

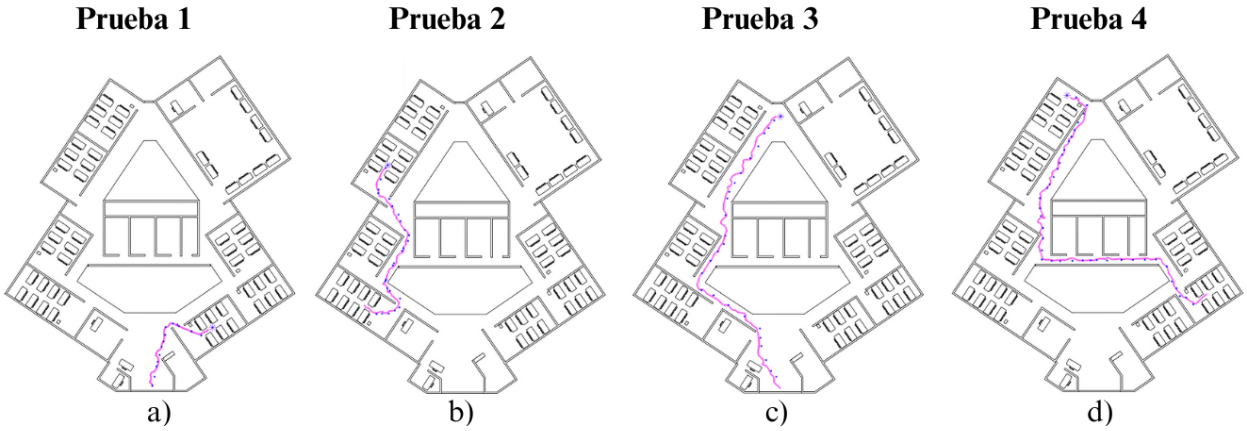


Figura 5.29: Trayectorias en el entorno con obstáculos a)<10, b)10-20, c)>20, d)>20

Tabla 5.21: Pruebas realizadas con obstáculos

Prueba	Puntos	Dist. (m)	%Completas	%Puntos Totales	%Puntos Dados
1	15	< 10	96.67 %	97.78 %	99.77 %
2	26	$10 - 20$	46.67 %	76.28 %	97.31 %
3	26	> 20	53.33 %	69.38 %	97.51 %
4	40	> 20	43.33 %	67.56 %	97.95 %

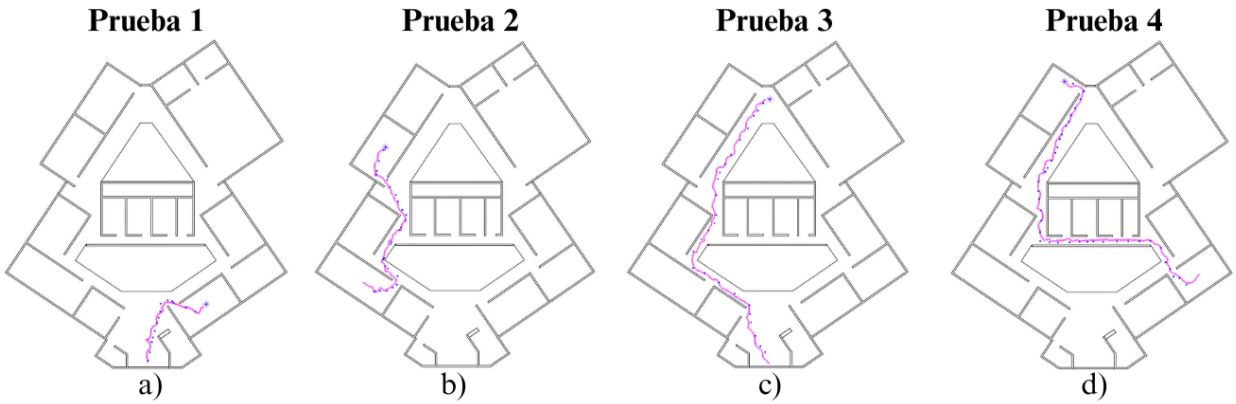


Figura 5.30: Trayectorias en el entorno sin obstáculos a)<10, b)10-20, c)>20, d)>20

Tabla 5.22: Pruebas realizadas sin obstáculos

Prueba	Puntos	Dist. (m)	%Completas	%Puntos Totales	%Puntos Dados
1	15	< 10	80 %	88.67 %	98.50 %
2	26	$10 - 20$	50 %	79.62 %	97.58 %
3	26	> 20	63.33 %	77.53 %	98.25 %
4	40	> 20	32.5 %	73.25 %	97.00 %

Esto se debe a que el modelo fue entrenado con un único punto final como objetivo, en lugar de una trayectoria completa. Por esta razón, el desempeño se evalúa en función de los puntos alcanzados. En la Tabla 5.21 y en la Tabla 5.22, se observa que en todas las pruebas el modelo ha mostrado un desempeño superior al 95 % en relación con los puntos realmente asignados, lo que indica un buen rendimiento. También se observa que, en la Prueba 1 y en la Prueba 4, con obstáculos, el modelo obtiene mejores resultados en cuanto a los puntos totales alcanzados; sin embargo, en términos de completar las pruebas, el porcentaje es menor. Esto sugiere que, aunque el robot recorre una mayor distancia, no logra completar en su totalidad la trayectoria asignada.

Es importante destacar que el entrenamiento no se llevó a cabo en el entorno de prueba

debido a su complejidad; los intentos de entrenar en este entorno no lograron converger a resultados satisfactorios. Sin embargo, las pruebas en un entorno desconocido con diversos obstáculos sugieren que el modelo es robusto, ya que mantiene su funcionalidad en condiciones distintas a las del entrenamiento. Además, entrenar al robot para alcanzar puntos aleatorios con diferentes inicios permitió que siguiera trayectorias variadas, evitando que aprendiera un único camino específico.

Al analizar la complejidad del entorno, especialmente en la Prueba 4, se observó un mayor número de fallos cuando el robot recorrió un pasillo con condiciones diferentes a las del entorno de entrenamiento, como la presencia de puertas y paredes a menos de un metro de distancia. Aunque la distancia euclidiana entre el punto inicial y el final en esta prueba es similar a la de la Prueba 3, la distancia real recorrida es considerablemente mayor debido a las condiciones del entorno y al mayor número de puntos en la trayectoria. Esto resulta en un desempeño inferior en términos de finalización de la trayectoria. No obstante, si se considera el porcentaje de puntos alcanzados, los resultados superan el 65 %, lo cual, aunque no ideal, es satisfactorio dadas las condiciones de prueba y la complejidad del entorno, ya que en condiciones con menor complejidad, se obtienen mejores resultados.

Capítulo 6

Navegación *online*

En este trabajo se considera la navegación *online* como el proceso que combina la obtención del mapa mientras el robot navega. Esto implica que la adquisición del mapa y la definición de los puntos deseados para alcanzar la meta se realizan conforme el robot obtenga información del entorno, hasta llegar al punto deseado, como se detalla en el diagrama de la Figura 6.1. A diferencia de la navegación *offline*, en la navegación *online* el robot no dispone de un mapa previo del entorno; únicamente tiene conocimiento de la meta a alcanzar. Para llegar a ella, debe explorar y descubrir el entorno en tiempo real, lo que le permite navegar en áreas que no han sido previamente mapeadas. Esto es una ventaja, ya que facilita la navegación en entornos desconocidos.

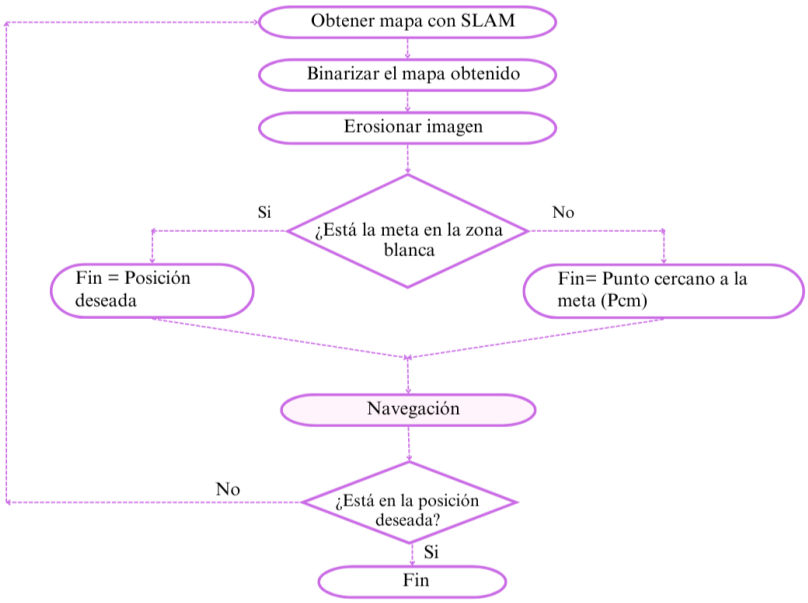


Figura 6.1: Navegación *online*

6.1. PRM-SLAM *online*

El algoritmo PRM-SLAM realiza la navegación mediante control cinemático. A partir del mapa binarizado, se genera la trayectoria, tomando como punto de inicio la posición del robot. Para detectar el punto final, se verifica si la meta se encuentra en la zona libre que ha mapeado el robot (zona blanca). En caso de que no esté, se generan puntos aleatorios y se selecciona el Punto más Cercano a la Meta (PCM). Si la meta está en la zona mapeada, esta se selecciona como punto de destino. El proceso se describe en el pseudocódigo 5. Así mismo la representación gráfica se muestra en la Figura 6.2

Algoritmo 5 Pseudocódigo PRM-SLAM Propuesto

```
1: función PRMSLAM
2:     Definir meta_x,meta_y
3:     mientras Posición <> Meta hacer
4:         ImagenRVIZ=EjecutarSLAM
5:         ProcesarImagen(ImagenRVIZ)
6:         pix_x,pix_y=CalcularPixelOrigen(mapa)
7:         Definir muestras(m), vecinos(v) iniciales
8:         Definir distancia_meta
9:         Definir distancia_Pcm
10:        x, y = Odometría
11:        distancia = Distancia(x,y,meta_x, meta_y)
12:        si distancia < distancia_meta entonces
13:            ControlCinematico((x, y, meta_x, meta_y))
14:        si no
15:            si La Meta está en la imagen entonces
16:                Ruta = PRM(Imagen, pr, Meta, m, v)
17:            si no
18:                Pcm = ObtenerPcm(Imagen)
19:                mientras Ruta==None hacer
20:                    Ruta = PRM(Imagen, pr, Pcm, m, v)
21:                    Pcm = ObtenerNuevoPcm(Imagen)
22:                    Limpiar PcmNoAccesible
23:                    Guardar PCM y puntos de la trayectoria (PT)
24:                    RecorrerTrayectoria((x, y, pcm_x, pcm_y))
25:                fin mientras
26:            fin si
27:        fin si
28:    fin mientras
29: fin función
```

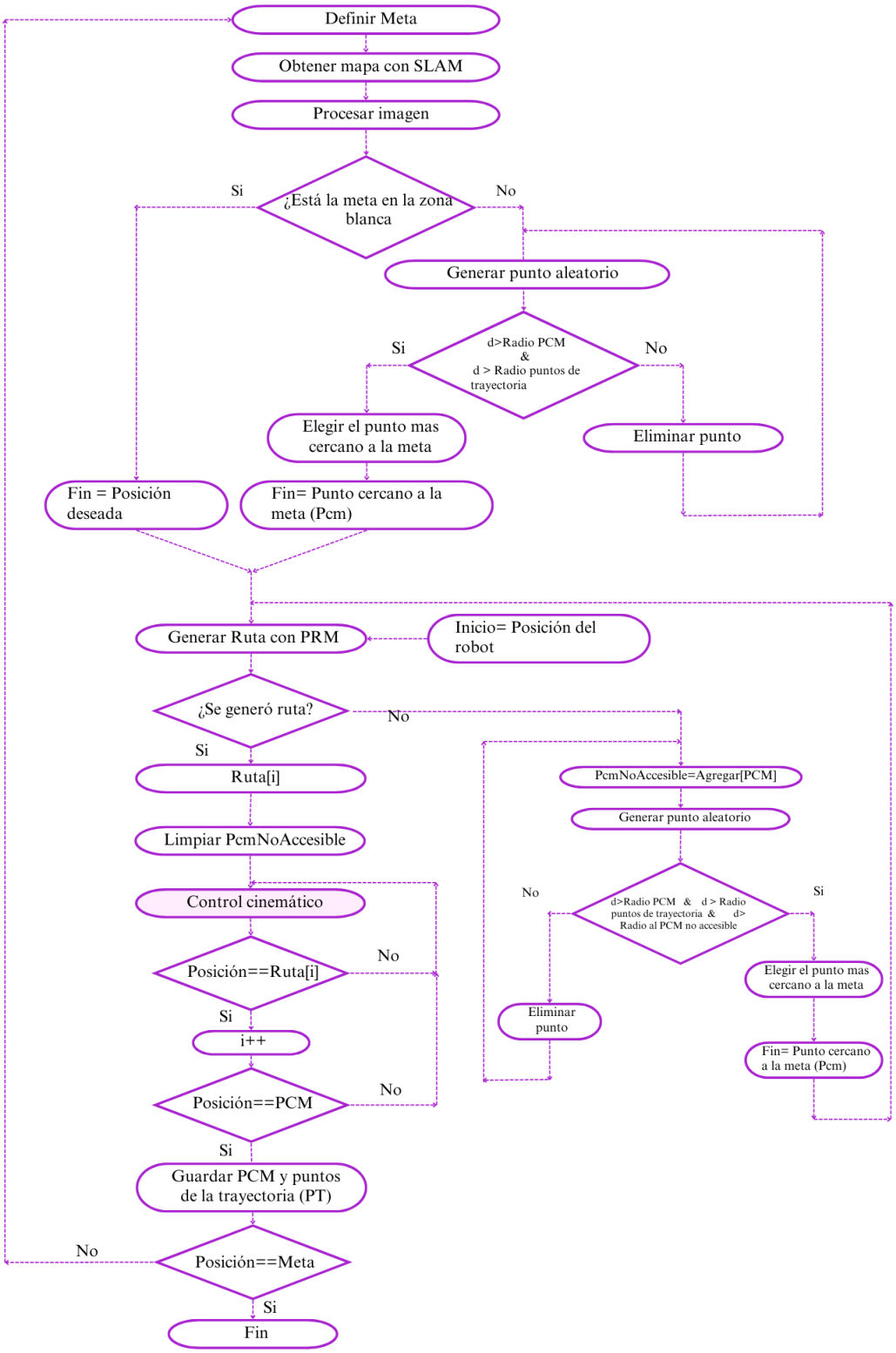


Figura 6.2: PRM-SLAM-Control Cinemático

6.1.1. Ejecución del PRM-SLAM *online*

Para ejecutar el PRM-SLAM *online* es necesario:

- Determinar la meta.
- Ejecutar en la terminal los siguientes comandos

```
cd ~/prm_slam_ws/
source devel/setup.bash
rosrun prm_slam prm_slam_online.py

```

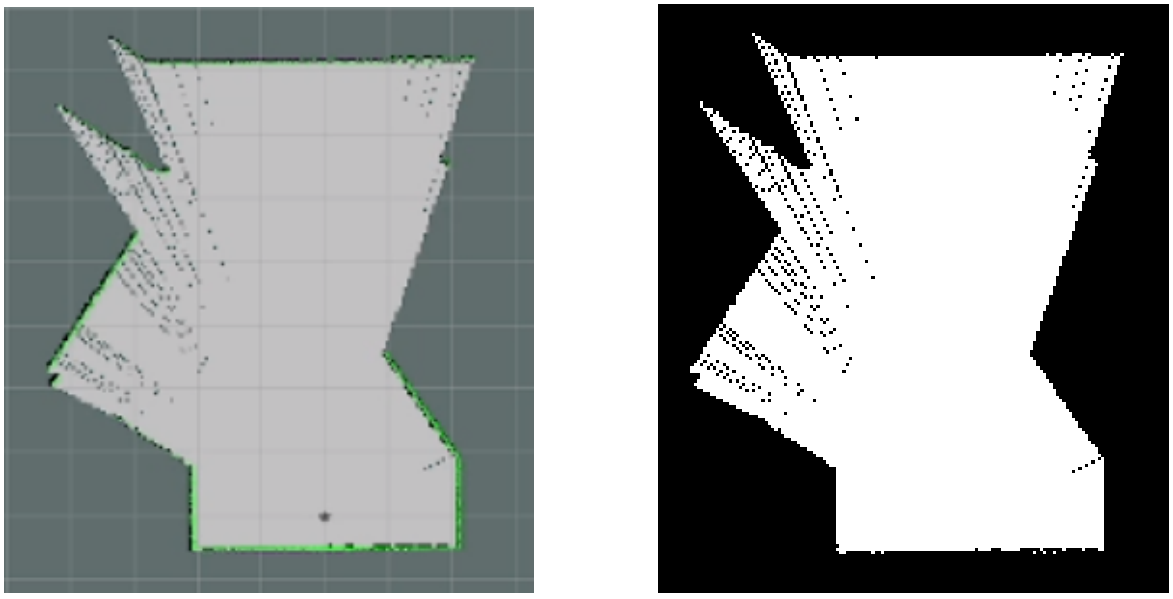
El video se puede consultar en: https://www.youtube.com/watch?v=ReEV5yGQyYA. El código para realizar la prueba se puede consultar en el Anexo D o https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/src/prm_slam/src/prm_slam_online.py, en el siguiente repositorio se encuentra el *workspace* https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/README.md.

Mapeo

Primero se obtiene el mapa siguiendo el pseudocódigo 6, en el cual se ejecuta el SLAM, a diferencia del proceso para obtener el mapa con SLAM del psudocódigo 1 el robot no se teleopera, pues se va a obtener el mapa conforme el robot navegue, por otro lado se mezcla con el pseudocódigo 2 para que al obtener la lectura del mapa, se binarice la imagen, este proceso se muestra en la Figura 6.3.

Algoritmo 6 Pseudocódigo para el mapeo

```
1: función ProcesarImagen(ImagenRVIZ)
2:     ImagenBinarizada = Binarizar(ImagenRVIZ)
3:     ImagenFinal = Erosionar(ImagenBinarizada)
4:     return ImagenFinal
5: fin función
6: función EjecutarSLAM
7:     mientras Ejecución hacer
8:         GenerarMapaDesdePosicionActual()          ▷ Actualiza el mapa
9:     fin mientras
10:    ImagenDeRVIZ = Mapa
11:    ProcesarImagen(ImagenDeRVIZ)
12: fin función
```



(a) Mapa en RVIZ (b) Imagen binarizada

Figura 6.3: Mapa inicial

Posteriormente se erosiona la imagen, para evitar que el robot navegue cerca de los obstáculos, estableciendo un radio de seguridad y así evitar colisiones, este proceso se muestra en la Figura 6.4.

(a) Imagen binarizada (b) Imagen procesada

Figura 6.4: Procesamiento de imagen

Una vez que se obtiene la imagen binarizada, es necesario ubicar la posición inicial del robot en la imagen. En el pseudocódigo 7 se describe cómo se obtiene el píxel en el que se encuentra el robot. Esto es fundamental para determinar el punto inicial de la trayectoria. En la primera iteración, la posición del robot coincide con el píxel de origen. Sin embargo, en las iteraciones siguientes, para obtener la posición del robot en relación con la imagen, es necesario convertir las distancias medidas en metros (obtenidas con la odometría) a píxeles. Esta conversión se realiza utilizando el pseudocódigo 8.

Algoritmo 7 Pseudocódigo para calcular el Píxel de Origen en X e Y

```
1: función CalcularPixelOrigen(origen_x, origen_y, x_min, x_max, y_min, y_max,
   width, height)
2:     Esperar hasta que la transformación de '/map' a '/odom' esté disponible
3:     Obtener la transformación y guardar la translación en trans
4:     origen_global_x= origen_x + trans[0]
5:     origen_global_y= origen_y + trans[1]
6:     Calcular resx y resy
7:     pixel_origen_x = origen_global_x / resx
8:     Altura - pixel_origen_y = origen_global_y / resy
9:     return (pixel_origen_x, pixel_origen_y)
10: fin función
```

Algoritmo 8 Pseudocódigo para metros a píxeles

```
1: función MetrosAPixeles(x, y, pixel_origen_x, pixel_origen_y, resx, resy )
2:     pixel_x = pixel_origen_x + round(x / resx)
3:     pixel_y = pixel_origen_y - round(y / resy)
4:     return (pixel_x, pixel_y)
5: fin función
```

Obtener PCM

Dado que la meta no se encuentra en el área que ha mapeado el robot, es necesario seleccionar el PCM. Este será el punto final que se asignará al PRM. Para elegir este punto, es necesario restringir un área de acuerdo con los PCM seleccionados anteriormente, limitando también el área cercana a los puntos de la trayectoria para evitar que el robot regrese a estas zonas.

Para obtener el PCM es necesario realizar la conversión de pixeles a metros (pseudocódigo 10) y la conversión de metros a píxeles (pseudocódigo 8).

Algoritmo 9 Obtener PCM

```
1: función ObtenerPcm(imagenBinaria, imagenColor, centros, ruta, nodos)
2:     Definir DistanciaMinima(d), DistanciaAPuntosTrayectoria(dt)
3:     puntosGuardadosPixeles = MetrosAPixeles(centros)
4:     puntosRutaPixeles = MetrosAPixeles(ruta)
5:     para nodo en rango(nodos) hacer
6:         repetir
7:             px = GenerarNumeroAleatorio(0, ancho_imagen - 1)
8:             py = GenerarNumeroAleatorio(0, ancho_imagen - 1)
9:             si (px, py) en espacioBlanco entonces
10:                distancias = Distancia(puntosGuardadosPixeles, (px, py))
11:                distanciaRuta = Distancia(puntosRutaPixeles, (px, py))
12:                si todas las distancias ≥ d y todas las distanciasRuta ≥ dt entonces
13:                    xm, ym = PixelesAMetros(px, py)
14:                    distancia = Distancia((xm, ym), (metaX, metaY))
15:                    si distancia < distanciaPCM entonces
16:                        pcm = (px, py)
17:                        distanciaPCM = distancia
18:                    fin si
19:                    break
20:                fin si
21:            fin si
22:        hasta que verdadero
23:    fin para
24:    return pcm
25: fin función
```

Algoritmo 10 Pseudocódigo para píxeles a metros

```
1: función PixelesAMetros(pixel_x, pixel_y, origen_global_x, origen_global_y, x_max,
   y_max, origen_y)
2:     y = (-origen_y + y_max + origen_global_y)
3:     metros_x = origen_global_x + (pixel_x) * resx
4:     metros_y = y - (pixel_y) * resy
5:     return (metros_x, metros_y)
6: fin función
```

En la Figura 6.5 se puede observar el punto de origen en color verde oscuro, los puntos aleatorios generados en color naranja y el PCM en color verde claro. También se aprecia que la imagen ha sido reducida para evitar elegir puntos cercanos a un obstáculos.

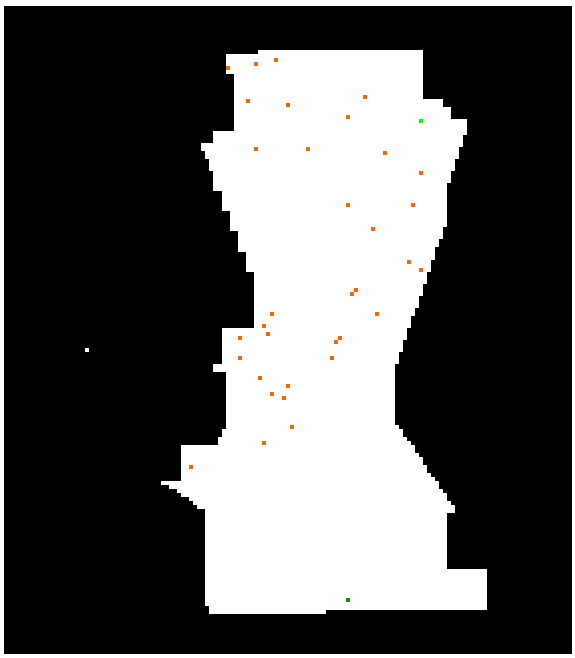


Figura 6.5: Obtención de PCM

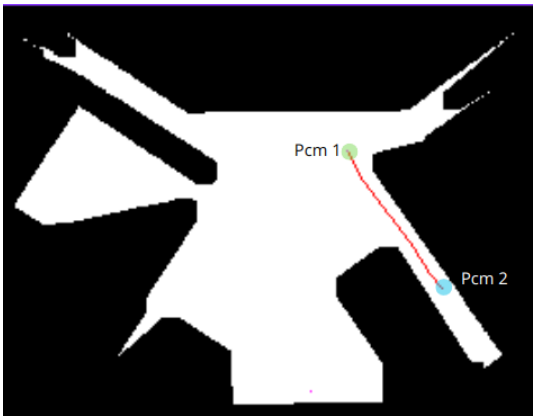
PRM

Posteriormente se obtiene la primer ruta con PRM, la cual se observa en la Figura 6.6. Se repite el algoritmo para obtener la segunda y la tercer ruta como se muestra en la Figura 6.7. Con cada iteración es importante aumentar el número de muestras, ya que al crecer el área blanca puede que las muestras iniciales no sean suficientes para lograr generar una trayectoria.

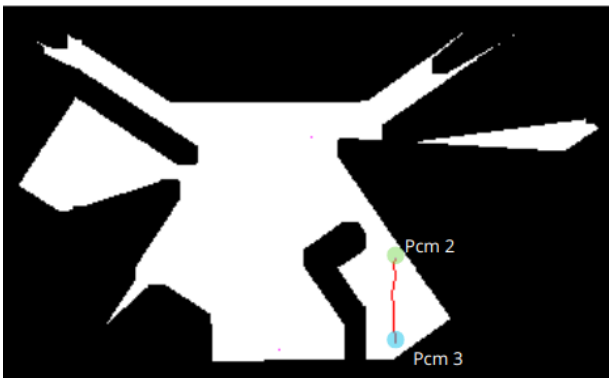
Algoritmo 11 Pseudocódigo PRM

```
1: función PRM(ImagenProcesada, Inicio, Meta, NumMuestras, Vecinos)
2:     Muestras = GenerarMuestrasAleatorias(NumMuestras)
3:     Grafo = ConstruirGrafo(Muestras, Vecinos)
4:     Camino = BuscarCamino(Grafo, Inicio, Meta)
5:     VectorDePosiciones = ConvertirPixelesAMetros(Camino)
6:     return VectorDePosiciones
7: fin función
```

Figura 6.6: Primer trayectoria obtenida con PRM



(a) Segunda trayectoria obtenida con PRM



(b) Tercer trayectoria obtenida con PRM

Figura 6.7: Rutas generadas

Generar nuevo PCM

Sin embargo, puede presentarse el caso en que el PCM se encuentre en una zona a la que el robot no puede acceder, o bien que no se pueda generar la trayectoria. Como se muestra en la Figura 6.8, el punto rojo está en un área donde, al erosionar la imagen, la distancia de seguridad impide el paso del robot. Por lo tanto, es necesario buscar un punto alternativo, para ello, se utiliza el mismo principio para restringir el área del PCM, para evitar que se vuelva a generar el PCM en la zona no accesible, este código se repite hasta obtener un punto valido para generar la trayectoria, obteniendo así la ruta de la Figura 6.9.

Algoritmo 12 Obtener PCM al no encontrar ruta

```
1: función ObtenerNuevoPcm(imagenBinaria, imagenColor, centros, ruta, nodos, PcmNoAccesible)
2:     PcmNoAccesiblePixeles = MetrosAPixeles(PcmNoAccesible)
3:     para nodo en rango(nodos) hacer
4:         repetir
5:             px = GenerarNumeroAleatorio(0, ancho_imagen - 1)
6:             py = GenerarNumeroAleatorio(0, ancho_imagen - 1)
7:             si (px, py) en espacioBlanco entonces
8:                 distancias = Distancia(puntosGuardadosPixeles, (px, py))
9:                 distanciaRuta = Distancia(puntosRutaPixeles, (px, py))
10:                distanciaPunto = Distancia(PcmNoAccesiblePixeles, (px, py))
11:                si todas las distancias ≥ d y todas las distanciasRuta ≥ dt y todas las distanciasPunto ≥ d  entonces
12:                    xm, ym = PixelesAMetros(px, py)
13:                    distancia = Distancia((xm, ym), (metaX, metaY))
14:                    si distancia < distanciaPCM entonces
15:                        pcm = (px, py)
16:                        distanciaPCM = distancia
17:                    fin si
18:                    break
19:                fin si
20:            fin si
21:        hasta que verdadero
22:    fin para
23:    return pcm
24: fin función
```

Figura 6.8: PCM no accesible

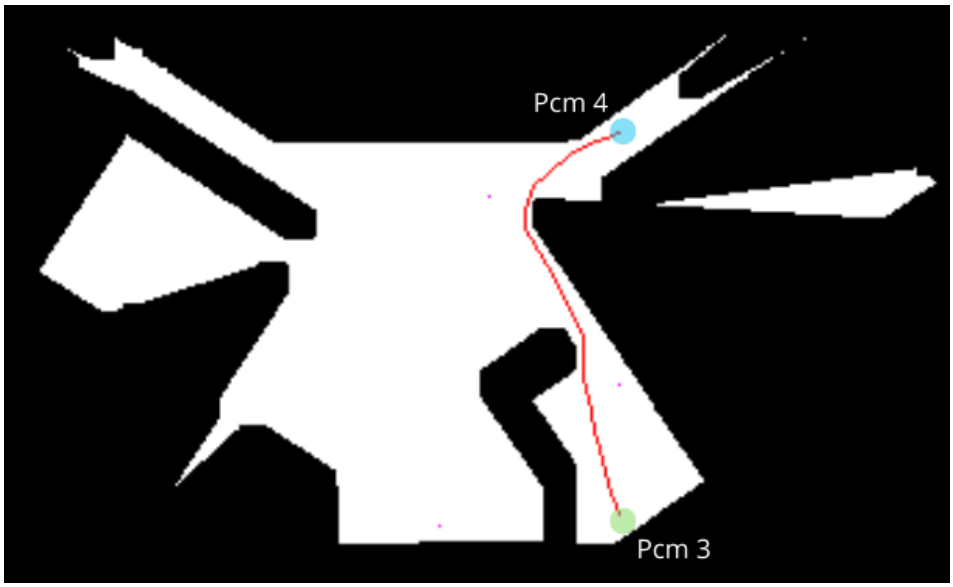
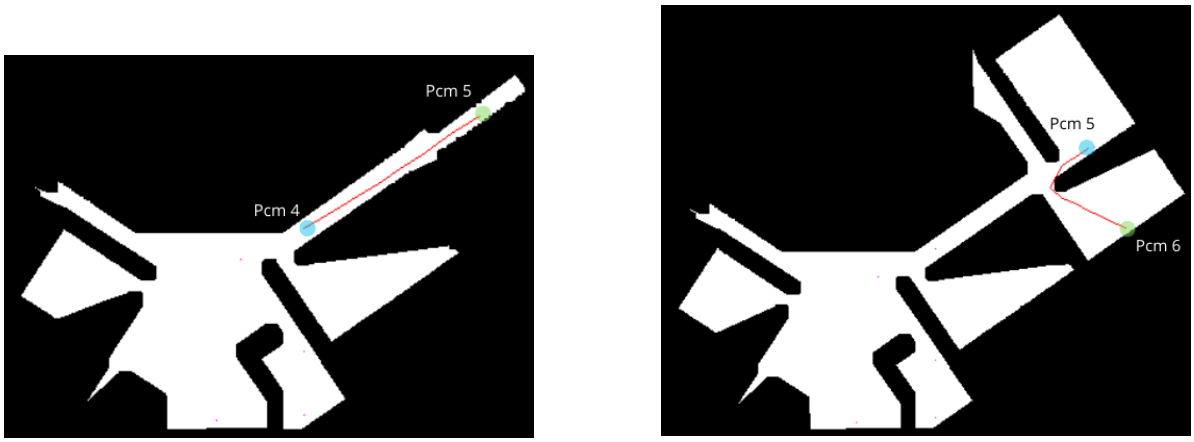


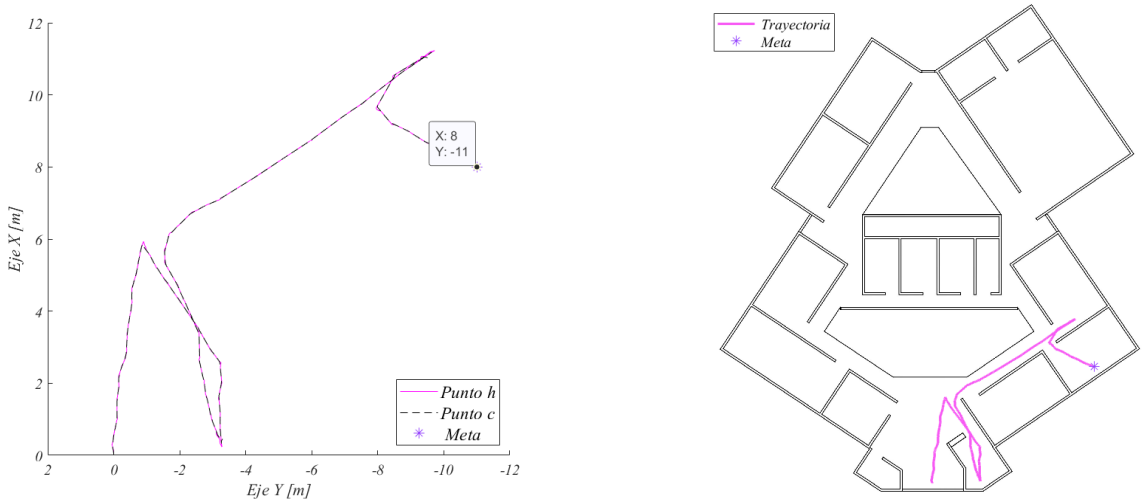
Figura 6.9: Cuarta trayectoria obtenida con PRM

Finalmente, se obtienen las últimas rutas para llegar a la meta [9m, -11m] (Figura 6.10). Al alcanzar la distancia establecida, se asigna el punto deseado del control cinemático a la meta, asegurando así que se llegue correctamente. Debido a que las conversiones entre metros y píxeles pueden tener menor precisión, se opta por asignar un área cercana a la meta que garantice la llegada al punto deseado con exactitud. No se puede seleccionar el punto una vez que está en la zona blanca, ya que entre la última posición del robot y el punto deseado pueden presentarse obstáculos. En la Figura 6.11b se observa el recorrido completo del robot.



(a) Quinta trayectoria obtenida con PRM (b) Última trayectoria obtenida con PRM

Figura 6.10: Últimas rutas generadas



(a) (b)

Figura 6.11: Resultados de la prueba PRM-SLAM *Online* sin obstáculos de evaluación 4. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

6.2. SLAM-RL

Para la navegación dinámica utilizando SLAM y RL, se omite la generación de trayectorias con PRM, como se describe en el pseudocódigo 13. Por lo tanto, se modifica el algoritmo de la Figura 6.2; se obtiene el mapa y se procesa la imagen. Sin embargo, se ajusta la obtención del PCM, cuando se generan los puntos aleatorios se realiza dentro de un diámetro limitado, cercano a la ubicación del robot, la obtención del PCM se describe en el pseudocódigo 14 para proporcionarle este punto de destino al modelo de RL.

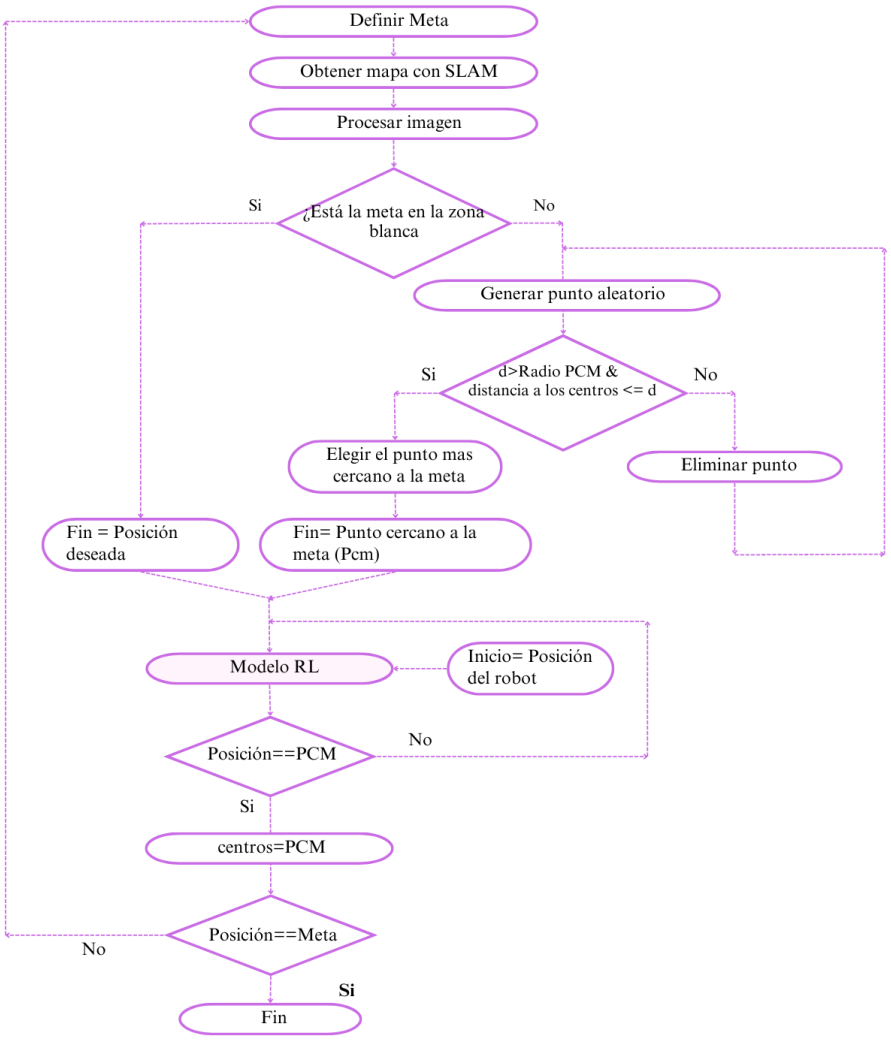


Figura 6.12: SLAM-RL

Algoritmo 13 Pseudocódigo PRM-SLAM Propuesto

```
1: función PRMSLAM
2:     Definir meta_x,meta_y
3:     mientras Posición <> Meta hacer
4:         ImagenRVIZ=EjecutarSLAM
5:         ProcesarImagen(ImagenRVIZ)
6:         pix_x,pix_y=CalcularPixelOrigen(mapa)
7:         Definir muestras(m), vecinos(v) iniciales
8:         x, y = Odometría
9:         si La Meta está en la imagen entonces
10:            ModeloRL((x, y, pcm_x, pcm_y))
11:        si no
12:            Pcm = ObtenerPcm(Imagen)
13:            Guardar PCM y puntos de la trayectoria (PT)
14:            ModeloRL((x, y, pcm_x, pcm_y))
15:        fin si
16:    fin mientras
17: fin función
```

6.2.1. Ejecución del SLAM-RL

Para ejecutar el SLAM-RL es necesario seguir los siguientes pasos:

- Ejecutar los siguientes comandos mostrados (Los códigos del repositorio se pueden consultar en el Anexo D)

```
$ export ROS_HOSTNAME=localhost
$ export ROS_MASTER_URI=http://localhost:11311
$ export ROS_PORT_SIM=11311
$ export GAZEBO_RESOURCE_PATH=~/DRL-robot-navigation/catkin_ws/src/
    multi_robot_scenario/launch

$ source ~/.bashrc
$ cd ~/DRL-robot-navigation/catkin_ws
$ source devel_isolated/setup.bash
```

- Modificar en el código de prueba (https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/test_plano.py)

  La línea que importa el entorno de RL en conjunto con el SLAM(https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/plano_env_slam_sin_prm.py)

```
    from plano_env_slam_sin_prm import GazeboEnv
```

```
3
```

La línea que exporta el entorno de Gazebo, el cual tambien ejecuta el SLAM.

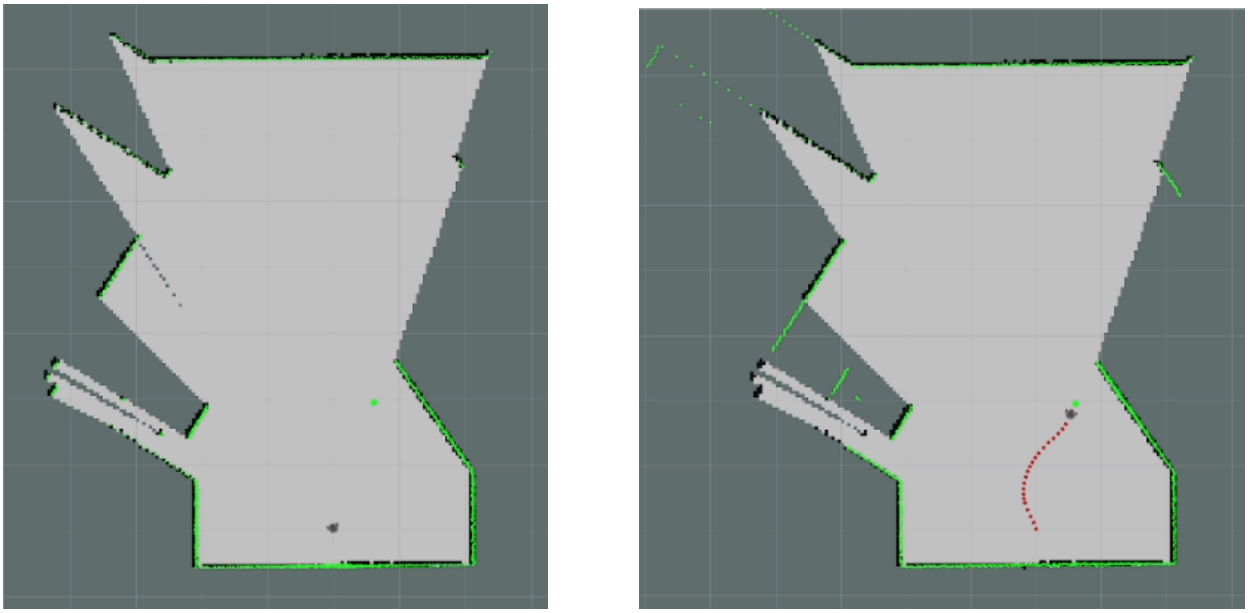
```
1
2     env = GazeboEnv("plano_completo_obstaculos_noetic.launch",
    environment_dim)
```

- Finalmente se ejecuta el código de prueba https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/test_plano.py

```
1
2     python3 test_plano.py
3
```

6.2.2. Prueba SLAM-RL

Se realizó la prueba hasta el punto [9 m, -10.5 m], que va de la Entrada al Aula 2. Se modifica la meta de acuerdo con la prueba realizada con el PRM-SLAM con el control cinemático, ya que esta se llevó a cabo en un entorno con obstáculos. En la Figura 6.13a se muestra la primer lectura y en la Figura 6.13b la trayectoria que siguió el robot con el modelo de RL para alcanzar el primer PCM. El video de esta prueba se puede consultar en: https://www.youtube.com/watch?v=ReEV5yGQyYA.

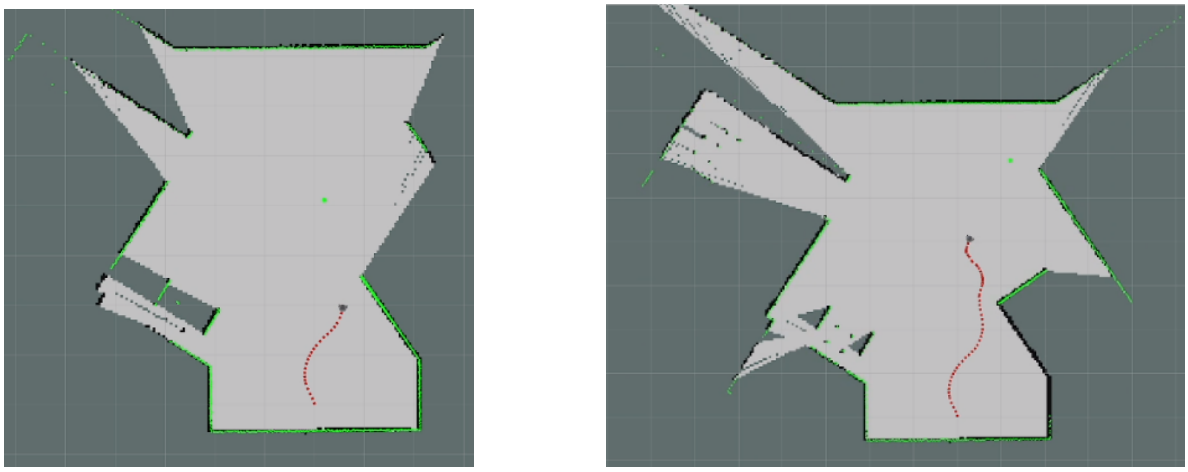


(a) Primer PCM obtenido (b) Llegada al primer punto

Figura 6.13: Primer PCM

Algoritmo 14 Obtener PCM

```
1: función ObtenerPcm(imagenBinaria, imagenColor, centros, ruta, nodos)
2:     Definir DistanciaMinima(d), DistanciaACentros(dc)
3:     puntosGuardadosPixeles = MetrosAPixeles(centros)
4:     para nodo en rango(nodos) hacer
5:         repetir
6:             px = GenerarNumeroAleatorio(0, ancho_imagen - 1)
7:             py = GenerarNumeroAleatorio(0, ancho_imagen - 1)
8:             si (px, py) en espacioBlanco entonces
9:                 distanciaAlCentro = Distancia(odom_x,odom_y, (px, py))
10:                si distanciaAlCentro <= dc entonces
11:                    distancias = Distancia(puntosGuardadosPixeles, (px, py))
12:                    si todas las distancias ≥ d entonces
13:                        xm, ym = PixelesAMetros(px, py)
14:                        distancia = Distancia((xm, ym), (metaX, metaY))
15:                        si distancia < distanciaPCM entonces
16:                            pcm = (px, py)
17:                            distanciaPCM = distancia
18:                        fin si
19:                        break
20:                    fin si
21:                fin si
22:            fin si
23:        hasta que verdadero
24:    fin para
25:    return pcm
26: fin función
```



(a) Segundo punto (b) Tercer punto

Figura 6.14: Generación de puntos para llegar a la meta

Se obtuvo un nuevo PCM, una vez que se llegó al primer punto asignado, como se muestra en la Figura 6.14. Posteriormente se van generando los puntos aleatorios al descubrir el mapa con SLAM (Figura 6.15). Finalmente una vez que la meta esta en una zona en la que ha mapeado el robot, se llega a la meta como se muestra en la Figura 6.15.

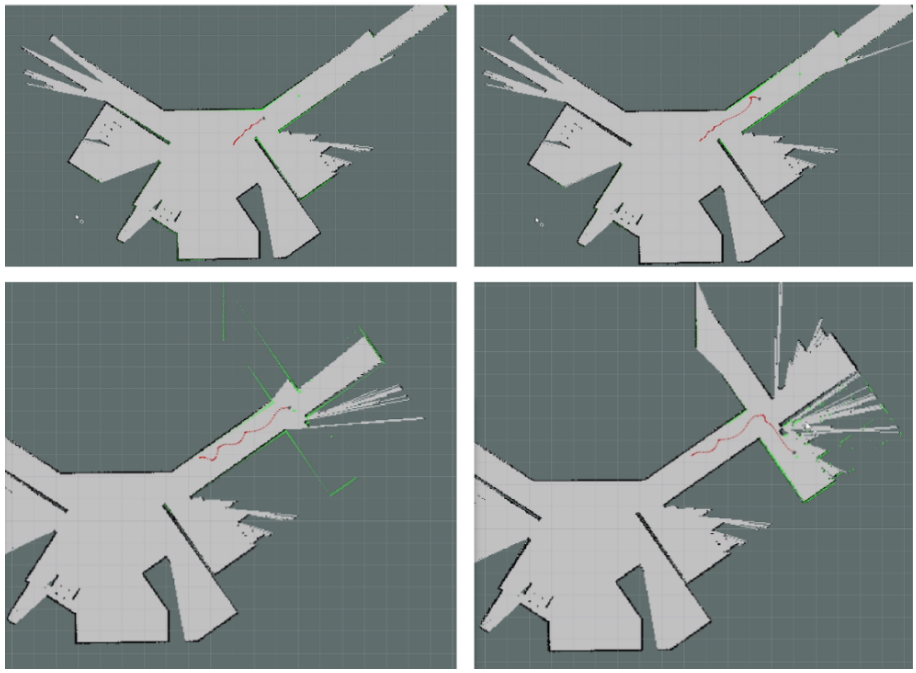


Figura 6.15: Recorrido del robot con SLAM-RL

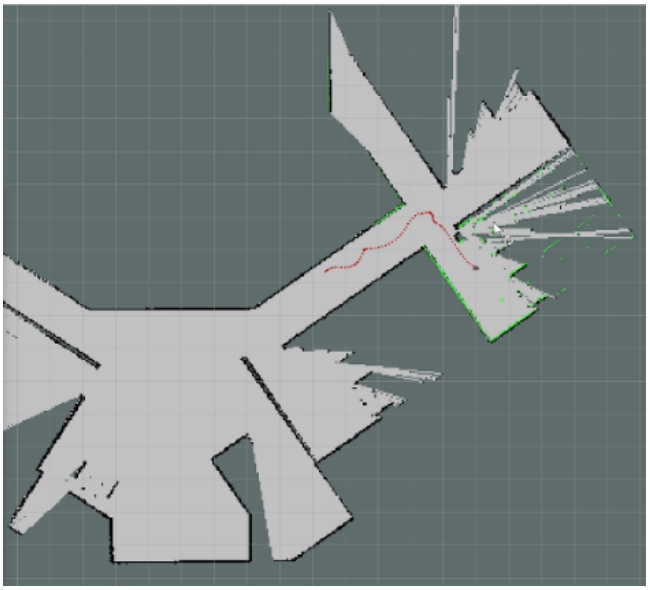


Figura 6.16: Penúltimo PCM generado

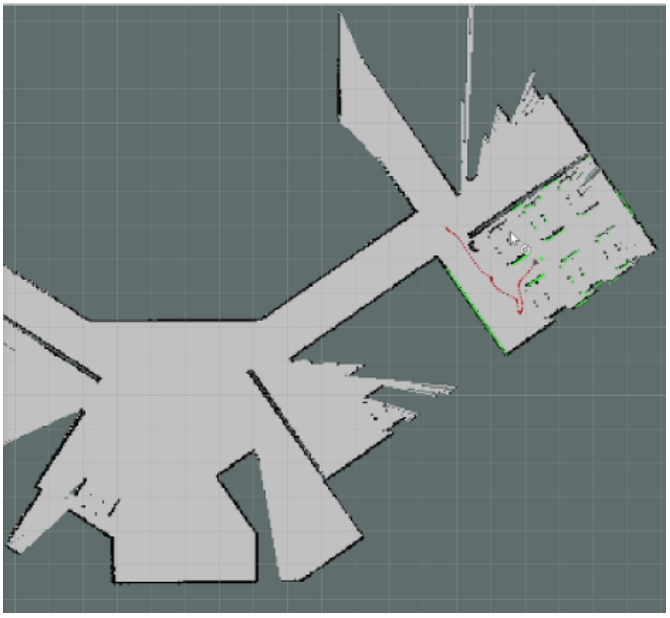
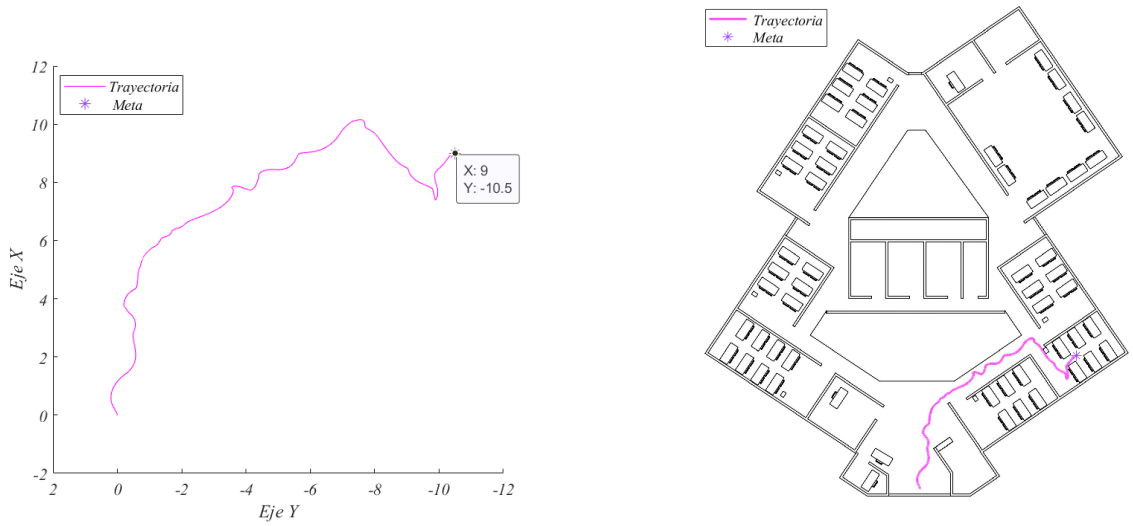


Figura 6.17: El robot llega a la meta

La trayectoria que recorrió el robot durante la prueba se muestra en la Figura 6.18a, así como el recorrido con respecto al entorno (Figura 6.18b).



(a) (b)

Figura 6.18: Resultados de la prueba SLAM-RL *Online* con obstáculos de evaluación 4. a) Trayectoria realizada en el plano, b) Visualización de la trayectoria en el entorno.

Conclusiones

En este trabajo se implementó un algoritmo de navegación autónoma basado en RL para un robot móvil de configuración diferencial, logrando que navegara de manera segura en un entorno estático.

El objetivo principal fue realizar la navegación dentro de un edificio de la UTM, específicamente en la División de Estudios de Posgrado. Para esto, se creó un entorno virtual 3D que representara el área de interés. Posteriormente, se generó el mapa del entorno utilizando el algoritmo Gmapping-SLAM, y se diseñó un algoritmo de navegación basado en RL que emplea este mapa como referencia para la toma de decisiones del robot.

Inicialmente, se realizó la navegación *offline* en el cual se ejecutó el algoritmo SLAM para generar un mapa del entorno, mediante teleoperación del robot, posteriormente, se obtuvo una trayectoria con PRM a partir del mapa obtenido, y se realizaron pruebas con control cinemático. Aunque las primeras pruebas fueron satisfactorias, se detectó que, en caso de un desfase o una asignación incorrecta de la trayectoria, podían ocurrir colisiones. Por este motivo, se decidió utilizar un modelo de RL, entrenado para llegar a una meta mientras evade obstáculos, lo que resultó en una navegación más robusta.

Para el obtener el modelo RL, el entrenamiento del modelo se realizó en un entorno pequeño con obstáculos que cambiaban de lugar con cada iteración de forma aleatoria. Tras obtener el modelo, se extrapoló a un entorno más complejo, en este caso la División de Estudios de Posgrado, demostrando que el modelo de RL es lo suficientemente robusto para adaptarse a condiciones diferentes a las del entorno de entrenamiento.

Posteriormente, se integraron el algoritmo SLAM y los algoritmos de navegación, para así poder navegar de forma *online*. En esta integración, permite la obtención continua del mapa mientras el robot navegaba de manera autónoma, ya sea con el control cinemático o el modelo de RL, el robot recibió una meta como objetivo final, permitiendo que se desarrollara un sistema de navegación completamente autónomo, sin intervención humana.

El algoritmo PRM-SLAM, que utiliza control cinemático para navegar, generó mapas y tra-

yectorias satisfactorias, pero fue probado únicamente en un entorno sin obstáculos, debido a que la dilatación de los objetos para evitar colisiones limitaba el acceso entre los escritorios. Por otro lado, el algoritmo SLAM-RL, que permite que el robot navegue mientras obtiene el mapa, demostró un excelente desempeño en entornos con obstáculos. Esto se debe a que el modelo de RL permite evadir obstáculos; al reducir la dilatación de los objetos, se facilita el paso entre los escritorios y el modelo se encarga de evitar las colisiones. Asimismo, uno de los problemas al generar las trayectorias con PRM al descubrir el mapa es que, si no hay conectividad entre los puntos en el grafo, no se puede generar una trayectoria, lo que provoca que el robot se detenga hasta que logre encontrar una solución. Para evitar esta situación, se prescindió del PRM y se consideró generar una submeta para llegar a la meta final; sin embargo, esta se restringió a una distancia de 2 a 3 metros debido a las condiciones en las que se entrenó el modelo de RL.

La combinación del algoritmo SLAM para la generación de mapas y el uso de RL para la navegación permitió que el robot operara sin conocimiento previo del entorno, adaptándose de manera eficiente a un escenario desconocido.

Los resultados obtenidos con ambos algoritmos fueron satisfactorios, ya que permitieron que el robot navegara de manera autónoma en un entorno complejo sin conocimiento previo del mapa y en un entorno diferente al que fue entrenado, demostrando así su versatilidad. La capacidad de obtener el mapa y los puntos de destino a medida que el robot descubre el entorno posibilita que ambos algoritmos enfrenten entornos desconocidos y complejos, lo que abre la puerta a su uso en espacios que no son seguros para los humanos, como en situaciones de rescate o exploración.

6.3. Trabajos Futuros

A partir de la evaluación del modelo de RL, se observó que, a medida que aumenta la distancia de navegación, el robot tiende a colisionar con mayor frecuencia. Por lo tanto, es necesario mejorar dicho modelo para aumentar su efectividad, especialmente en distancias mayores, ajustando los hiperparámetros durante el entrenamiento, con el objetivo de lograr que el robot navegue distancias más largas sin colisionar. Las pruebas realizadas *online* se llevaron a cabo a distancias superiores a los 10 metros, por lo que, al mejorar el modelo de RL, sería posible que el robot realice recorridos aún más largos.

En el caso de que se necesite explorar un entorno de manera completa y sin una meta definida, sería útil desarrollar un modelo de RL que permita al robot explorar áreas completamente desconocidas, optimizando la navegación en estos escenarios. Esto sería especialmente relevante en situaciones donde el entorno sea incierto o cambiante, como en entornos de rescate, inspección o exploración.

Referencias

[1] M. R. T. García *et al.*, "Robótica móvil," 2017.

[2] L. I. G. Calandín, "Modelado cinemático y control de robots móviles con ruedas," Ph.D. dissertation, Universidad Politécnica de Valencia, 2006.

[3] I. Bambino, "Una introducción a los robots móviles," *Recuperado http://www. aadeca. org/pdf/CP_monografias/monografia_robot_movil. pdf*, 2008.

[4] E. Valadez Campos, E. VALADEZ CAMPOS *et al.*, "Simulación del movimiento de un vehículo terrestre para trayectorias programadas aplicando control difuso," B.S. thesis, Benemérita Universidad Autónoma de Puebla, 2018.

[5] M. N. I. Bonilla, "Navegación autónoma de un robot con técnicas de localización y ruteo," Ph.D. dissertation, Instituto Nacional de Astrofísica, Óptica y Electrónica, 2009.

[6] H. Moreno Avalos, R. J. Saltaren Pazmiño, L. J. Puglisi, I. G. Carrera Calderon, P. F. Cardenas Herrera, and C. Álvarez, "Robótica submarina: Conceptos, elementos, modelado y control," *Revista Iberoamericana de Automática e Informática industrial*, vol. 11, no. 1, pp. 3–19, 2014.

[7] G. Alvarez and O. Flor, "Desempeño en métodos de navegación autónoma para robots móviles," *Minerva*, vol. 1, no. 2, pp. 19–29, 2020.

[8] C. A. V. Carvajal, J. A. G. Luna, and I. D. T. Pardo, "Evaluación de las técnicas de planificación de movimientos, descomposición exacta trapezoidal y descomposición adaptativa de celdas a través de mallas," *Revista Facultad de Ingeniería*, vol. 21, no. 32, pp. 41–53, 2012.

[9] F. M. Santa, F. H. M. Sarmiento, E. J. Gómez *et al.*, "Using the delaunay triangulation and voronoi diagrams for navigation in observable environments," *Tecnura*, vol. 18, pp. 81–88, 2014.

[10] G. Qing, Z. Zheng, and X. Yue, "Path-planning of automated guided vehicle based on improved dijkstra algorithm," in *2017 29th Chinese control and decision conference (CCDC)*. IEEE, 2017, pp. 7138–7143.

[11] N. Alpkiray, Y. Torun, and O. KAYNAR, "Probabilistic roadmap and artificial bee colony algorithm cooperation for path planning," in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. IEEE, 2018, pp. 1–6.

[12] J. Chen, Y. Zhou, J. Gong, and Y. Deng, "An improved probabilistic roadmap algorithm with potential field function for path planning of quadrotor," in *2019 Chinese Control Conference (CCC)*. IEEE, 2019, pp. 3248–3253.

[13] L. G. Véras, F. L. Medeiros, and L. N. Guimaraes, "Rapidly exploring random tree* with a sampling method based on sukharev grids and convex vertices of safety hulls of obstacles," *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, p. 1729881419825941, 2019.

[14] M. R. H. Al-Dahhan and K. W. Schmidt, "Path planning based on voronoi diagram and prm for omnidirectional mobile robots," in *DTSS2019 international conference. Ankara, Turkey*, 2019.

[15] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang, and Z. Dawei, "An improved a-star based path planning algorithm for autonomous land vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 5, p. 1729881420962263, 2020.

[16] J. M. A. Bernal, J. A. B. Ruíz, J. C. A. Díaz, and V. A. T. Monteza, "Planificador de rutas para recojo de desechos sólidos utilizando el algoritmo de dijkstra," *INGENIERÍA: Ciencia, Tecnología e Innovación*, vol. 8, no. 2, pp. 92–99, 2021.

[17] I. Palacios Serrano, C. Cruz Ulloa, and M. Barraza Rodríguez, "Análisis de los algoritmos de planificación de trayectorias rrt, prm y voronoi en la solución de un laberinto modular controlado por una plataforma de dos gdl," *Ingeniare. Revista chilena de ingeniería*, vol. 30, no. 1, pp. 157–170, 2022.

[18] H. E. Espitia Cuchango and J. I. Sofrony Esmeral, "Algoritmo para planear trayectorias de robots móviles, empleando campos potenciales y enjambres de partículas activas brownianas," *Ciencia e Ingeniería Neogranadina*, vol. 22, no. 2, pp. 75–96, 2012.

[19] R. Osorio-Comparán, I. López-Juárez, A. Reyes-Acosta, M. Pena-Cabrera, M. Bustamante, and G. Lefranc, "Mobile robot navigation using potential fields and lma," in *2016 IEEE International Conference on Automatica (ICA-ACCA)*. IEEE, 2016, pp. 1–7.

[20] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[21] H. F. Durrant-Whyte, "Uncertain geometry in robotics," *IEEE Journal on Robotics and Automation*, vol. 4, no. 1, pp. 23–31, 1988.

[22] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[23] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot." in *IROS*, vol. 3, 1991, pp. 1442–1447.

[24] F. A. A. Cheein, C. De la Cruz, R. Carelli, and T. F. Bastos Filho, "Navegación autónoma asistida basada en slam para una silla de ruedas robotizada en entornos restringidos," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 8, no. 2, pp. 81–92, 2011.

[25] E. Martínez Sánchez, "Navegación autónoma de un robot móvil en un ambiente desconocido usando una red de neuronas electrónicas," Master's thesis, Tesis (MC)–Centro de Investigación y de Estudios Avanzados del IPN, 2022.

[26] N. D. Toan and K. Gon-Woo, "Environment exploration for mapless navigation based on deep reinforcement learning," in *2021 21st International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2021, pp. 17–20.

[27] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, "A comparative analysis of lidar slam-based indoor navigation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[28] J. Xin, H. Zhao, D. Liu, and M. Li, "Application of deep reinforcement learning in mobile robot path planning," in *2017 Chinese Automation Congress (CAC)*. IEEE, 2017, pp. 7112–7116.

[29] J. Rufo Paris *et al.*, "Técnicas de aprendizaje por refuerzo aplicadas a la navegación de un robot móvil," 2022.

[30] M. Choi, R. Sakthivel, and W. K. Chung, "Neural network-aided extended kalman filter for slam problem," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 1686–1690.

[31] S. S. Haykin and S. S. Haykin, *Kalman filtering and neural networks*. Wiley Online Library, 2001, vol. 284.

[32] A. E. Rodas Córdova, "Diseño e implementación de un sistema de generación de trayectoria para el control de un robot móvil, utilizando inteligencia artificial," B.S. thesis, Universidad del Azuay, 2021.

[33] "Autopilot." [Online]. Available: https://www.tesla.com/es_MX/autopilot

[34] T. Greicius, "Nasa's self-driving perseverance mars rover 'takes the wheel'," Jul 2021. [Online]. Available: https://www.nasa.gov/feature/jpl/nasa-s-self-driving-perseverance-mars-rover-takes-the-wheel

[35] G. Ramírez, "Método de aprendizaje simple para navegación de minirobots móviles rodantes," *Dyna*, vol. 70, no. 138, pp. 59–66, 2003.

[36] "Roomba serie j." [Online]. Available: https://www.irobot.lat/roomba/j-series

[37] G. Grisetti, R. Kúmmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[38] J. Sierra-García and M. Santos, "Redes neuronales y aprendizaje por refuerzo en el control de turbinas eólicas," *Revista Iberoamericana de Automática e Informática industrial*, vol. 18, no. 4, pp. 327–335, 2021.

[39] C. A. V. Hernández, J. J. C. Chávez, and E. C. Nieto, "Implementación de sistema de navegación autónomo en robot móvil experimental para reconstrucción y exploración de entornos desconocidos," *Memorias*, 2015.

[40] A. P. de Contenidos de OBS Business School, "Metodología de programación: Definición, tipos y aplicación." [Online]. Available: https://www.obsbusiness.school/blog/metodologia-de-programacion-definicion-tipos-y-aplicacion

[41] L. A. Góngora Velandia *et al.*, "Localización y mapeo simultáneo basado en la librería de nube de puntos [pcl]: etapa de registro," B.S. thesis, Universidad Piloto de Colombia, 2015.

[42] L. Fernández, L. Payá, M. Ballesta, F. Amorós, and O. Reinoso, "Odometría visual y construcción de un mapa topológico a partir de la apariencia global de imágenes omnidireccionales," 2011.

[43] F. Werner, F. Maire, and J. Sitte, "Topological slam using fast vision techniques," in *FIRA RoboWorld Congress*. Springer, 2009, pp. 187–196.

[44] V. M. Jáquez Leal, "Construcción de mapas y localización simultánea con robots móviles." 2005.

[45] F. Cheein, R. Carelli, A. F. Neto, T. Bastos Filho, and W. Celeste, "Mapeo probabilístico y localización simultáneas em um robot móvil gobernado por una icc," *IV Jornadas Argentinas de Robótica, Córdoba*, 2006.

[46] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 3, pp. 249–265, 1987.

[47] J. Minguez, F. Lamiraux, and J.-P. Laumond, "Motion planning and obstacle avoidance," in *Springer handbook of robotics*. Springer, 2016, pp. 1177–1202.

[48] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[49] V. Vonásek, J. Faigl, T. Krajník, and L. Přeučil, "Rrt-path–a guided rapidly exploring random tree," in *Robot motion and control 2009*. Springer, 2009, pp. 307–316.

[50] B. Boots and N. Shiode, "Recursive voronoi diagrams," *Environment and Planning B: Planning and Design*, vol. 30, no. 1, pp. 113–124, 2003.

[51] S. Martínez and R. Sisto, "Campos potenciales," 2009.

[52] A. Yufka and O. Parlaktuna, "Performance comparison of bug algorithms for mobile robots," in *Proceedings of the 5th international advanced technologies symposium, Karabuk, Turkey*, 2009, pp. 13–15.

[53] L. Rodríguez Castañeda *et al.*, "Algoritmos para calcular la ruta más corta en la malla vial de la ciudad de bogotá," 2005.

[54] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[55] A. Milstein, "Occupancy grid maps for localization and mapping," *Motion planning*, pp. 381–408, 2008.

[56] D. Simon, "Kalman filtering," *Embedded systems programming*, vol. 14, no. 6, pp. 72–79, 2001.

[57] A. Díaz Comeche, "Diseño de una metodología para incorporar simulación monte carlo en planificaciones de braquiterapia," 2023.

[58] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle filters for mobile robot localization," in *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 401–428.

[59] J. D. C. Cartagena and V. H. J. Velásquez, "Estimación de posición en robots móviles usando filtros de partículas," *Revista Politécnica*, vol. 13, no. 25, pp. 103–113, 2017.

[60] J. Yang, X. Cui, J. Li, S. Li, J. Liu, and H. Chen, "Particle filter algorithm optimized by genetic algorithm combined with particle swarm optimization," *Procedia Computer Science*, vol. 187, pp. 206–211, 2021.

[61] "Robot operating system." [Online]. Available: https://www.ros.org/

[62] R. Suáreza, J. Rosella, M. Vinagreb, F. Cortesb, A. Ansuateguic, I. Maurtuac, D. Martind, A. Guashd, J. Azpiazuf, D. Serranog *et al.*, "Robot operating system (ros)," *Este trabajo se ha llevado a cabo por iniciativa y en el seno del Grupo de Trabajo de Innovación de la Asociación Española de Robótica y Automatización (AER).. Accessed: Nov*, vol. 28, 2022.

[63] L. Joseph, *ROS Robotics projects*. Packt Publishing Ltd, 2017.

[64] Robotnik, "Aplicaciones del nuevo robot turtlebot2: Robotnik®," Aug 2021. [Online]. Available: https://robotnik.eu/es/aplicaciones-del-nuevo-robot-kobuki-turtlebot-ii-2/

[65] A. T. Norman, *Aprendizaje automático en acción*. Litres, 2019.

[66] X. B. Olabe, "Redes neuronales artificiales y sus aplicaciones," *Publicaciones de la Escuela de Ingenieros*, 1998.

[67] D. J. Matich, "Redes neuronales: Conceptos básicos y aplicaciones," *Universidad Tecnológica Nacional, México*, vol. 41, pp. 12–16, 2001.

[68] J. C. Ponce Gallegos, A. Torres Soto, F. S. Quezada Aguilera, A. Silva Sprock, E. U. Martínez Flor, A. Casali, E. Scheihing, Y. J. Túpac Valdivia, M. D. Torres Soto, F. J. Ornelas Zapata *et al.*, *Inteligencia artificial*. Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn), 2014.

[69] R. Zambrano, "Modelos de machine learning," Apr 2023. [Online]. Available: https://openwebinars.net/blog/modelos-de-machine-learning/

[70] J. A. Villamil Torres and J. A. Delgado Rivera, "Entrenamiento de una red neuronal multicapa para la tasa de cambio euro-dólar (eur/usd)," *Ingeniería e investigación*, vol. 27, no. 3, pp. 106–117, 2007.

[71] Na8, "Aprendizaje por refuerzo," Dec 2020. [Online]. Available: https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/

[72] R. Candela Arias, "Navegación de una apiladora industrial mediante aprendizaje por refuerzo," 2020.

[73] R. Cimurs, I. H. Suh, and J. H. Lee, "Goal-driven autonomous exploration through deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 730–737, 2022.

[74] L. H. R. González, J. A. Valencia, and A. Montoya, "Modelo cinemático de un robot móvil tipo diferencial y navegación a partir de la estimación odométrica." *Scientia et Technica*, vol. 1, no. 41, pp. 191–196, 2009.

[75] P. A. J. Israel, T. A. Saúl, U. G. V. Carrillo, G. H. Efrén, P. O. J. Carlos, V. S. J. Emilio, R. A. J. Manuel, and S. O. Artemio, "Robot móvil de tracción diferencial con plataforma de control modular para investigación y desarrollo ágil de proyectos," in *10 Congreso Nacional de Mecatrónica-Puerto Vallarta*, vol. 4, 2011.

[76] L. E. S. Guzmán, M. A. M. Villa, and E. L. R. Vásquez, "Seguimiento de trayectorias con un robot móvil de configuración diferencial," *Ingenierías USBMed*, vol. 5, no. 1, pp. 26–34, 2014.

[77] G. Hernández Millán, L. H. Ríos Gonzales, and M. Bueno López, "Implementación de un controlador de posición y movimiento de un robot móvil diferencial," *Tecnura*, vol. 20, no. 48, pp. 123–136, 2016.

[78] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Force control*. Springer, 2009.

[79] G. M. Andaluz Ortiz, "Modelación y control predictivo de un robot móvil con centro de masa desplazado." 2018.

[80] R. L. Villarreal Onofre *et al.*, "Un framework basado en ros para la navegación de robots móviles autónomos," B.S. thesis, Benemérita Universidad Autónoma de Puebla, 2021.

Anexo A

Plano

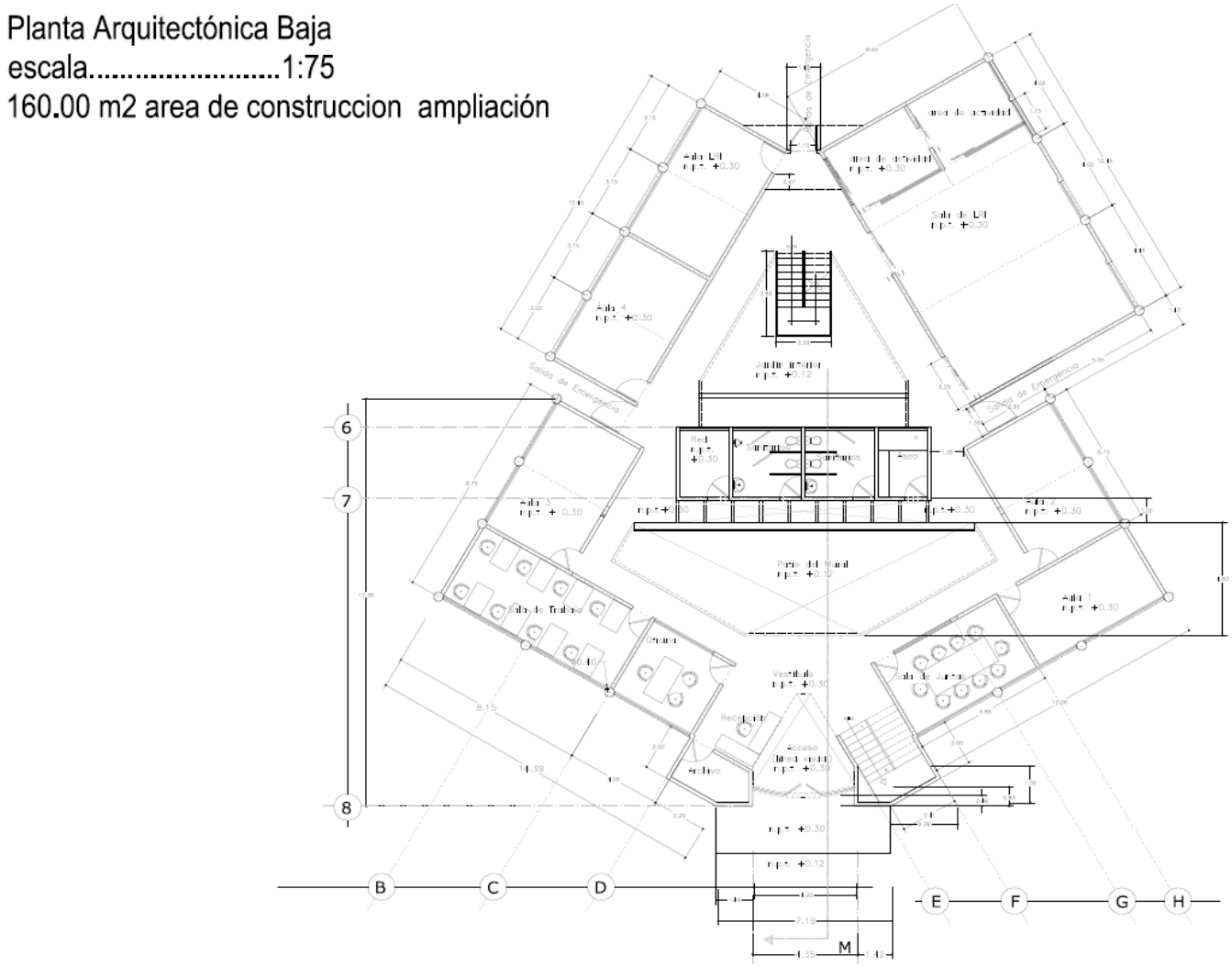


Figura A.1: Plano del edificio de la "*División de estudios de posgrado*" de la UTM.

Anexo B

Entorno

B.1. Archivos Plano

B.1.1. Archivo Launch del plano

```
<launch>
  <arg name="model" default="burger"/>
  <arg name="x_pos" default="0"/>
  <arg name="y_pos" default="0"/>
  <arg name="z_pos" default="0.15"/>
  <arg name="roll" default="0"/>
  <arg name="pitch" default="0"/>
  <arg name="yaw" default="0"/>
  <!-- Plano -->
   <arg name="x2" default="8.927857"/>
  <arg name="y2" default="32.771286"/>
  <arg name="z2" default="0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/plano.world
    "/>
    <arg name="paused" value="false"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
 </include>
  <node
    name="tf_footprint_base"
    pkg="tf"
    type="static_transform_publisher"
    args="0 0 0 0 0 0 base_link base_footprint 40" />
  <node
    name="spawn_model"
    pkg="gazebo_ros"
```

```
28      type="spawn_model"
29      args="-file $(find Plano_completo)/urdf/Plano_completo.urdf -urdf -model
        Plano_completo -x $(arg x2) -y $(arg y2) -z $(arg z2)""
30      output="screen" />
31    <node
32      name="fake_joint_calibration"
33      pkg="rostopic"
34      type="rostopic"
35      args="pub /calibrated std_msgs/Bool true" />
36      <param name="robot_description" command="$(find xacro)/xacro $(find
        turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
37
38    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -
        model turtlebot3 -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -R $(arg
        roll) -P $(arg pitch) -Y $(arg yaw) -param robot_description" />
39
40  </launch>
```

Anexo C

Archivos SLAM-PRM

C.1. Archivo launch del SLAM

```
<launch>

  <!-- Arguments -->


  <!--<arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [
    burger, waffle, waffle_pi]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>-->
  <arg name="model" default="burger"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="set_base_frame" default="base_link"/>
  <arg name="set_odom_frame" default="odom"/>
  <arg name="set_map_frame"  default="map"/>
  <arg name="scan_topic"  default="scan" />


  <!-- Gmapping -->
  <node pkg="gmapping" type="slam_gmapping" name="turtlebot3_slam_gmapping"
    output="screen">
    <param name="base_frame" value="$(arg set_base_frame)"/>
    <param name="odom_frame" value="$(arg set_odom_frame)"/>
    <param name="map_frame"  value="$(arg set_map_frame)"/>
    <rosparam command="load" file="$(find turtlebot3_slam)/config/
    gmapping_params.yaml" />
  </node>
</launch>
```

C.2. Archivo launch del plano y SLAM

```
<launch>
  <launch>
    <arg name="model" default="burger"/>
  <arg name="x_pos" default="0"/>
  <arg name="y_pos" default="0"/>
  <arg name="z_pos" default="0.15"/>

  <!-- Plano 1.6-->
      <arg name="x2" default="32.700508"/>
  <arg name="y2" default="-8.362269"/>
  <arg name="z2" default="0"/>
  <arg name="roll" default="0"/>
  <arg name="pitch" default="0"/>
  <arg name="yaw" default="-1.561545"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty.world
    "/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="false"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
<node
    name="spawn_model"
    pkg="gazebo_ros"
    type="spawn_model"
    args="-file $(find Plano_completo)/urdf/Plano_completo.urdf -urdf -model
    Plano_completo -x $(arg x2) -y $(arg y2) -z $(arg z2) -R $(arg roll) -P $(
    arg pitch) -Y $(arg yaw) "
    output="screen" />
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(
    find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

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

  <arg name="open_rviz" default="true"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB
```

```
    -Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_gmapping.launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="turtlebot3_lds_2d.lua"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_gmapping.rviz"/>
  </group>

  <node name="odometry_initializer" pkg="turtlebot3_gazebo" type="init_topic.
    py" output="screen" />


</launch>
```

C.3. Código PRM

```
import cv2
import networkx as nx
import numpy as np
import random
import matplotlib.pyplot as plt
import csv
from math import sin, cos, pi, sqrt
from scipy.spatial import KDTree

def pixeles_a_metros(pixel_x, pixel_y):
    # Calcula las coordenadas en metros a partir de las coordenadas en pixeles
    resx=(666-98)/28.6693
    resy=(735-175)/28.4363
    centro_y = 277
    centro_x = 508
    metros_y = ((centro_y-pixel_y) / resy)
    metros_x = ((centro_x-pixel_x) / resx)
    print("Coordenadas en metros del punto:", metros_x, metros_y)
    return metros_x, metros_y

def generar_muestras_validas(imagen_binaria, numero_de_muestras):
    # Obtener las coordenadas de los puntos blancos en la imagen binaria
    puntos_validos = np.column_stack(np.where(imagen_binaria == 255)) #
    Puntos blancos
    # Seleccionar aleatoriamente un n mero espec fico de muestras
    indices_muestras = np.random.choice(len(puntos_validos),
```

```
    numero_de_muestras, replace=False)
26      muestras = puntos_validos[indices_muestras]
27      imagen_color = cv2.cvtColor(imagen_binaria, cv2.COLOR_GRAY2BGR)
28      for (y, x) in muestras:
29          imagen_color[y, x] = [0, 255, 255]
30      return muestras
31
32  def hay_obstaculo_entre_puntos(punto1, punto2, imagen_binaria):
33      linea = np.linspace(punto1, punto2, num=10).astype(int)
34      for punto in linea:
35          if imagen_binaria[punto[0], punto[1]] == 0:
36              return True
37      return False
38
39  def dibujar_grafo(imagen_color, grafo, puntos):
40      imagen_dibujo = imagen_color.copy()
41      # Dibujar nodos
42      for punto in puntos:
43          cv2.circle(imagen_dibujo, (int(punto[1]), int(punto[0])), 5, (0, 255,
    255), -1)
44      for (p1, p2, data) in grafo.edges(data=True):
45          cv2.line(imagen_dibujo, (int(p1[1]), int(p1[0])), (int(p2[1]), int(p2
    [0])), (255, 0, 0), 1)  # Rojo
46      return imagen_dibujo
47
48  def prm( imagen, numero_de_muestras, distancia_max):
49      path = None
50
51      punto_inicio = (508,277)
52      punto_destino = (240, 300)
53
54      points = []  # Lista para almacenar los puntos aleatorios
55      img_color = cv2.cvtColor(imagen, cv2.COLOR_GRAY2BGR)
56
57      # Bucle para intentar encontrar una ruta
58      while path is None:
59          print("generating while")
60          grafo = nx.Graph()
61          muestras_validas = generar_muestras_validas(imagen, numero_de_muestras
    )
62          img_color[punto_inicio] = [255, 255, 0]  # Amarillo
63          img_color[punto_destino] = [0, 255, 255]  # Cian
64          cv2.imshow('imagen',img_color)
65          cv2.waitKey(0)
66          cv2.destroyAllWindows()
67          muestras = np.vstack([muestras_validas, punto_inicio, punto_destino])
      # Agregar puntos de inicio y destino
68          kdtree = KDTree(muestras)
69          for i, punto in enumerate(muestras):
```

```
70              distancias, vecinos_indices = kdtree.query(punto, k=25,
    distance_upper_bound=distancia_max)
71              for vecino_indice, distancia in zip(vecinos_indices, distancias):
72                  if distancia < distancia_max and not
    hay_obstaculo_entre_puntos(punto, muestras[vecino_indice], imagen):
73                      grafo.add_edge(tuple(punto), tuple(muestras[vecino_indice
    ]), weight=distancia)
74
75          # Dibuja la ruta en la imagen original.
76
77          img_color = cv2.cvtColor(imagen, cv2.COLOR_GRAY2BGR)
78          imagen_con_ruta = img_color.copy()  # Copia la imagen original.
79          im=dibujar_grafo(img_color, grafo, muestras)
80          cv2.imwrite("grafo5.jpg", im)
81          if nx.has_path(grafo, tuple(punto_inicio), tuple(punto_destino)):
82              ruta_optima = nx.shortest_path(grafo, source=tuple(punto_inicio),
    target=tuple(punto_destino), weight='weight')
83              print(len(ruta_optima))
84              punto_inicio = pixeles_a_metros(punto_inicio[0], punto_inicio[1])
85              punto_destino = pixeles_a_metros(punto_destino[0], punto_destino
    [1])
86              distancia_total = sqrt((punto_inicio[0] - punto_destino[0])**2 + (
    punto_inicio[1] - punto_destino[1])**2)
87              path = ruta_optima
88              for i in range(len(path) - 1):
89                  point1 = (path[i][1], path[i][0])
90                  point2 = (path[i + 1][1], path[i + 1][0])
91                  cv2.line(imagen_con_ruta, point1, point2, (0, 0, 255), 1)
92                  points.append(point2)
93              # Dibuja los puntos aleatorios en la imagen
94              for point in points:
95                  cv2.circle(imagen_con_ruta, point, 2, (0, 255, 0), -1)
96              punto_destino=(punto_destino[1],punto_destino[0])
97              points.append(punto_destino)
98              cv2.imwrite("prueba10_2000_1000.jpg", imagen_con_ruta)
99              # Guardar los puntos aleatorios en un archivo CSV
100             with open("puntos10_2000_1000.csv", "w", newline="") as csvfile:
101                 csvwriter = csv.writer(csvfile)
102                 csvwriter.writerow(["X", "Y"])
103                 for point in points:
104                     x,y=pixeles_a_metros(point[0],point[1])
105                     print(x,y)
106                     csvwriter.writerow((x,y))
107
108             with open("prueba10_2000_1000.csv", "w", newline="") as csvfile2:
109                 csvwriter = csv.writer(csvfile2)
110                 csvwriter.writerow(["X", "Y"])
111                 for i in range(len(path)):
112                     x,y=pixeles_a_metros(path[i][0],path[i][1])
```

```
                    print(x,y)
                    csvwriter.writerow((x,y))
            return path
        else:
            return None


# Carga la imagen en escala de grises.
imagen = cv2.imread("slam_12m_2.png", cv2.IMREAD_GRAYSCALE)
imagen_invertida = cv2.bitwise_not(imagen)

kernel = np.ones((13,13),np.uint8)
dilatacion = cv2.dilate(imagen_invertida,kernel,iterations = 1)
kernel = np.ones((1,1),np.uint8)
img_dil = cv2.morphologyEx(dilatacion, cv2.MORPH_CLOSE, kernel)

cv2.imshow('imagen',img_dil)
# Establece un umbral para la binarizaci n.
umbral = 128  # Puedes ajustar este valor seg n tus necesidades.
imagen_inv = cv2.bitwise_not(img_dil)
imagen[400,107]=255
cv2.imshow('imagen',imagen_inv)

# Aplica la binarizaci n.
image = np.where(imagen_inv >= umbral, 255, 0)

num_samples = 2000
k_neighbors = 1000
path = prm(imagen_inv, num_samples, k_neighbors)
print(path)
```

Anexo D

Contenido Digital

D.1. Repositorios

Control Cinemático

https://github.com/itzchav/Control_Cinematico_Turtlebot3

Entorno de simulación

https://github.com/itzchav/Entorno-Division-Estudios-Posgrado

PRM SLAM Control Cinemático

https://github.com/itzchav/PRM_SLAM_Control_cinematico/tree/main

Offline
https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/src/prm_slam/src/prm_slam_offline.py

Online
https://github.com/itzchav/PRM_SLAM_Control_cinematico/blob/main/prm_slam_ws/src/prm_slam/src/prm_slam_online.py

SLAM-RL

https://github.com/itzchav/PRM_SLAM_Control_cinematico/tree/main

Offline
https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/plano_env.py

Online
https://github.com/itzchav/Navegacion-SLAM-RL/blob/main/TD3/plano_env_slam_sin_prm.py

D.2. Videos

Navegación *Offline*

Pruebas Navegación *offline*

https://www.youtube.com/watch?v=ReEV5yGQyYA

Evaluación del modelo de RL

https://www.youtube.com/watch?v=1TfGGCmBoH4

Navegación *Online*

https://www.youtube.com/watch?v=ReEV5yGQyYA

Anexo E

Pruebas

E.1. Prueba 1 con obstáculos

Tabla E.1: Resultados de la Prueba 1 (con obstáculos)

Pruebas	**Puntos recorridos**	**Puntos totales**	**Completo**
1	15	15	sí
2	15	15	sí
3	15	15	sí
4	15	15	sí
5	15	15	sí
6	15	15	sí
7	15	15	sí
8	15	15	sí
9	15	15	sí
10	15	15	sí
11	15	15	sí
12	4	15	no
13	15	15	sí
14	15	15	sí
15	15	15	sí
16	15	15	sí
17	15	15	sí
18	15	15	sí
19	15	15	sí
20	15	15	sí
21	15	15	sí
22	15	15	sí
23	15	15	sí
24	15	15	sí
25	15	15	sí
26	15	15	sí
27	15	15	sí
28	15	15	sí
29	15	15	sí
30	15	15	sí

E.2. Prueba 1 sin obstáculos

Tabla E.2: Resultados de la Prueba 1 (sin obstáculos)

Pruebas	Puntos recorridos	Puntos totales	Completo
1	15	15	sí
2	15	15	sí
3	15	15	sí
4	15	15	sí
5	5	15	no
6	15	15	sí
7	15	15	sí
8	15	15	sí
9	15	15	sí
10	15	15	sí
11	12	15	no
12	9	15	no
13	15	15	sí
14	15	15	sí
15	15	15	sí
16	5	15	no
17	15	15	sí
18	15	15	sí
19	15	15	sí
20	15	15	sí
21	15	15	sí
22	15	15	sí
23	3	15	no
24	15	15	sí
25	15	15	sí
26	15	15	sí
27	3	15	no
28	15	15	sí
29	15	15	sí
30	15	15	sí

E.3. Prueba 2 con obstáculos

Tabla E.3: Resultados de la Prueba 2 (con obstáculos)

Pruebas	**Puntos totales**	**Puntos objetivo**	**Completo**
1	26	26	sí
2	26	26	sí
3	16	26	no
4	6	26	no
5	17	26	no
6	26	26	sí
7	26	26	sí
8	6	26	no
9	16	26	no
10	26	26	sí
11	22	26	no
12	26	26	sí
13	6	26	no
14	22	26	no
15	22	26	no
16	26	26	sí
17	26	26	sí
18	22	26	no
19	7	26	no
20	26	26	sí
21	26	26	sí
22	17	26	no
23	11	26	no
24	26	26	sí
25	26	26	sí
26	17	26	no
27	26	26	sí
28	17	26	no
29	7	26	no
30	26	26	sí

E.4. Prueba 2 sin obstáculos

Tabla E.4: Resultados de la Prueba 2 (sin obstáculos)

Pruebas	**Puntos totales**	**Puntos objetivo**	**Completo**
1	26	26	sí
2	26	26	sí
3	22	26	no
4	26	26	sí
5	16	26	no
6	22	26	no
7	26	26	sí
8	13	26	no
9	6	26	no
10	26	26	sí
11	26	26	sí
12	16	26	no
13	26	26	sí
14	26	26	sí
15	22	26	no
16	16	26	no
17	26	26	sí
18	9	26	no
19	26	26	sí
20	26	26	sí
21	7	26	no
22	11	26	no
23	26	26	sí
24	26	26	sí
25	17	26	no
26	16	26	no
27	22	26	no
28	26	26	sí
29	16	26	no
30	26	26	sí

E.5. Prueba 3 con obstáculos

Tabla E.5: Resultados de la Prueba 3 (con obstáculos)

Pruebas	**Puntos totales**	**Puntos objetivo**	**Completo**
1	27	27	sí
2	6	27	no
3	16	27	no
4	27	27	sí
5	9	27	no
6	6	27	no
7	27	27	sí
8	27	27	sí
9	6	27	no
10	11	27	no
11	11	27	no
12	27	27	sí
13	27	27	sí
14	6	27	no
15	15	27	no
16	27	27	sí
17	11	27	no
18	27	27	sí
19	6	27	no
20	27	27	sí
21	27	27	sí
22	6	27	no
23	27	27	sí
24	27	27	sí
25	27	27	sí
26	15	27	no
27	6	27	no
28	27	27	sí
29	27	27	sí
30	27	27	sí

E.6. Prueba 3 sin obstáculos

Tabla E.6: Resultados de la Prueba 3 (sin obstáculos)

Pruebas	**Puntos totales**	**Puntos objetivo**	**Completo**
1	27	27	sí
2	27	27	sí
3	15	27	no
4	16	27	no
5	27	27	sí
6	27	27	sí
7	27	27	sí
8	9	27	no
9	27	27	sí
10	11	27	no
11	27	27	sí
12	6	27	no
13	27	27	sí
14	15	27	no
15	27	27	sí
16	27	27	sí
17	27	27	sí
18	6	27	no
19	6	27	no
20	27	27	sí
21	27	27	sí
22	27	27	sí
23	6	27	no
24	27	27	sí
25	27	27	sí
26	27	27	sí
27	27	27	sí
28	16	27	no
29	27	27	sí
30	9	27	no

E.7. Prueba 4 con obstáculos

Tabla E.7: Resultados de la Prueba 4 (con obstáculos)

Pruebas	**Puntos totales**	**Puntos objetivo**	**Completo**
1	41	41	sí
2	7	41	no
3	41	41	sí
4	9	41	no
5	21	41	no
6	40	41	no
7	41	41	sí
8	9	41	no
9	41	41	sí
10	41	41	sí
11	21	41	no
12	41	41	sí
13	41	41	sí
14	41	41	sí
15	40	41	no
16	41	41	sí
17	9	41	no
18	8	41	no
19	41	41	sí
20	12	41	no
21	8	41	no
22	18	41	no
23	41	41	sí
24	9	41	no
25	39	41	no
26	41	41	sí
27	18	41	no
28	12	41	no
29	41	41	sí
30	18	41	no

E.8. Prueba 4 sin obstáculos

Tabla E.8: Resultados de la Prueba 4 (sin obstáculos)

Pruebas	**Puntos totales**	**Puntos objetivo**	**Completo**
1	41	41	sí
2	18	41	no
3	17	41	no
4	41	41	sí
5	15	41	no
6	21	41	no
7	41	41	sí
8	41	41	sí
9	18	41	no
10	41	41	sí
11	41	41	sí
12	15	41	no
13	22	41	no
14	41	41	sí
15	41	41	sí
16	39	41	no
17	41	41	sí
18	39	41	no
19	41	41	sí
20	18	41	no
21	41	41	sí
22	41	41	sí
23	18	41	no
24	12	41	no
25	17	41	no
26	21	41	no
27	41	41	sí
28	21	41	no
29	18	41	no
30	39	41	no

Anexo F

Artículo Publicado

Implementation of potential fields in a differential mobile robot (*TurtleBot2*).

1st Karla I. Salado-Chávez
División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
itzchav3001@gmail.com

2nd Filiberto-E. Martínez-Hernández
División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
filiberto.martinez133@gmail.com

3rd Oscar D. Ramírez-Cárdenas
División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
odramirez@mixteco.utm.mx

4th José A. Arias-Aguilar
División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
anibal@mixteco.utm.mx

***Abstract*—This paper focuses on the implementation of a method based on the physical principle of "Potential Fields." A differential robot, known as *TurtleBot2* (Kobuki), was utilized to address the problem of autonomous navigation. To perceive the environment, a laser-based object detection and measurement system was used, enabling *TurtleBot2* to navigate complex environments by detecting and avoiding obstacles. Additionally, an NVIDIA *Jetson Nano* ARM architecture computer was utilized, equipped with a streamlined set of instructions for essential processing, enabling it to control the robot's angular and linear speed. The wireless connection between the computer and the *TurtleBot2* enabled efficient information transfer and data processing. This wireless connection extends robot's working range based on Wi-Fi network coverage, ensuring safe navigation in restricted and challenging spaces. Tests were conducted in two distinct environments, presenting results from both simulation and experimentation, demonstrating the efficacy of the technique.**

***Index Terms*—Potential Fields, ROS, Jetson Nano, LiDAR, TurtleBot**

I. Introduction

THE advancements about mobile robotics has generated great interest in development of autonomous systems, be able to navigating and performing tasks in dynamic and complex environments. One of fundamental challenges in autonomous navigation is algorithms design with objective that robots move safely and efficiently in an unknown environment, avoiding obstacles and reaching specific objectives. In this context, potential fields, has emerged as an effective technique for controlling mobile robots.

Recent years have been significant advancements in field of Potential Fields. Rostami [3] approached need for autonomous robots that avoid collisions and select optimal paths. Duhé et al. [4] presented four alternative formulations about repulsive field, considering dynamic and unpredictable obstacles in urban environments. Due to uncertainty in environment, it is required to use local rather than global road planning techniques.

This article presents implementation a navigation techniques using Potential Fields in Kobuki robot (TurtleBot2) for this technique was implemented LiDAR sensor and a *Jetson Nano* module were implemented, compatible with an open source system that provides libraries and tools for developing robotic applications and drivers Robotic Operating System (ROS) [5]. Using *Jetson Nano* obviates conventional installation on robot, as well as avoiding long wired connections robot to an external computer.

The primary objective is the achievement of wireless control of the robot from a computer connected to the Jetson Nano module, enabling efficient navigation in complex environments. This configuration yields a compact and flexible solution for robot control. Nodes facilitate real-time sensor data acquisition; by using Potential Fields technique, send angular and linear velocities for obstacle avoidance.

This technique can be extrapolated to robots navigating in situations where human safety is a priority, such as limited access areas or dangerous environments. It can be applied in various domains, including search and rescue, military security, space exploration, precision agriculture, urban logistics,