UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR

Previo a la obtención del título de:

INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN

"DESARROLLO E IMPLEMENTACIÓN DE ALGORITMOS TIPO INSECTO PARA LA PLANEACIÓN DE RUTAS DE UN ROBOT MÓVIL CON RUEDAS"

AUTORES:

CHOEZ REYES JULISSA ARELYS

ECHAIZ FLOREANO RODRIGO BRYAN

DOCENTE TUTOR:

ING. JUNIOR FIGUEROA OLMEDO, MSc.

LA LIBERTAD – ECUADOR

2024

AGRADECIMIENTO

Agradezco en primer lugar a Dios por ser el autor de mi existencia y concederme el privilegio de ingresar a esta maravillosa carrera llamada Electrónica y Automatización, gracias por permitirme llegar hasta este día tan importante para mí, su fortaleza y guía han sido fundamentales para culminar exitosamente este proyecto de investigación.

Agradezco a mi madre, Paula Reyes por su apoyo constante e incondicional, por inculcarme principios y valores esenciales en mi vida, su dedicación ha sido un pilar fundamental en esta travesía.

A mi padre, Bernaldo Choez por sus sabios consejos y constante motivación desde el inicio de mi carrera, su confianza en mí ha sido un impulso fundamental para alcanzar este sueño.

Agradezco a mi colega, Rodrigo Echaiz, por su apoyo incondicional y por acompañarme en cada paso de este proceso, su colaboración ha sido invaluable.

A mis hermanos por su constante apoyo y por creer en mí., sus palabras de aliento han sido esenciales y comparto mis logros con ustedes con gran satisfacción.

A mi tutor, Ing. Junior Figueroa expreso mi más sincero agradecimiento por su paciencia, dedicación y guía durante todo mi proceso de titulación, su apoyo y conocimiento han sido esencial para mi desarrollo académico.

Agradezco a mis amigos ingenieros, especialmente a Kevin Villao y Carmen Flores, por su apoyo y amistad en los momentos buenos y difíciles, su compañía y motivación han sido clave para lograr este objetivo.

A la Universidad Estatal Península de Santa Elena por permitirme lograr dar un paso más hacia el éxito, por convertirme en una profesional competitiva, llena de conocimientos y

expectativas. También quiero agradecer de manera muy especial a todos los docentes que me dieron clases por darme una formación profesional con calidad, muchísimas gracias por todo.

Julissa Arelys Choez Reyes

AGRADECIMIENTO

Agradezco a Dios por darme la fortaleza y el carácter necesarios para perseverar en esta etapa universitaria, por brindarme salud y energía para continuar cada día, y por poner en mi camino a las personas adecuadas para alcanzar y culminar este objetivo.

A mi madre, Paola Floreano, le agradezco profundamente por su constante apoyo y por no permitir que me rinda en este proceso de adquisición de conocimientos. Sus consejos siempre me motivaron a seguir adelante, recordándome que cada etapa de la vida es crucial y tiene su propio tiempo para empezar y terminar.

A mi padre, Rodrigo Echaiz, le debo mi interés y curiosidad por la electricidad, lo que me llevó a buscar una carrera relacionada con esta rama de la ingeniería. Además, su ejemplo me ha enseñado a ser una persona honrada y un profesional ejemplar.

Agradezco profundamente a mi compañera, Julissa Choez, por su constante apoyo durante todo el proceso de titulación, su dedicación y colaboración han sido esenciales.

A mis amigos y compañeros, les agradezco por su apoyo incondicional y por contribuir de alguna manera a mi progreso.

A los docentes y a mi tutor, Junior Figueroa, les expreso mi más sincero agradecimiento por compartir sus conocimientos en cada una de las materias en las que se especializan. Su dedicación ha sido invaluable para mi formación.

Rodrigo Bryan Echaiz Floreano

DEDICATORIA

Dedico este proyecto de titulación a mis padres, Bernaldo Choez y Paula Reyes, por su constante apoyo, su sacrificio y esfuerzo han sido fundamentales para el logro de mis objetivos. Su ejemplo de determinación y compromiso ha sido mi mayor inspiración, y este logro no habría sido posible sin su guía inquebrantable.

También deseo dedicar este proyecto a Rodrigo Echaiz por su apoyo y compañía durante este proceso académico, su comprensión y respaldo han sido esenciales para superar desafíos y alcanzar este logro.

Julissa Arelys Choez Reyes

DEDICATORIA

Dedico este trabajo de titulación a mis padres, Rodrigo Echaiz y Paola Floreano. Con abnegación, esfuerzo y una entrega incondicional, han dedicado su amor y apoyo para formarme como una persona íntegra. Cada logro y meta alcanzada, así como las que vendrán, son gracias a ustedes. Agradezco a Dios por haberme bendecido con padres como ustedes.

A mi compañera Julissa Choez, quien ha sido una guía y apoyo constante, aprecio enormemente tu paciencia, especialmente en los momentos en que quería rendirme y me motivaste a seguir adelante.

A mis queridos hermanos, quienes han sido una fuente constante de motivación y apoyo su orgullo y cariño me han impulsado a esforzarme y superar cada desafío. Gracias por inspirarme y recordarme siempre la importancia de la familia.

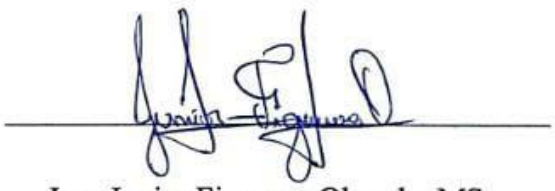
A mis familiares y amigos, quienes siempre creyeron en mí y me alentaron a seguir adelante. Gracias por su constante apoyo, sus palabras de aliento, además, por recordarme que sí podía lograrlo. Su fe en mí ha sido una fuente inagotable de motivación y fuerza a lo largo de este viaje.

Rodrigo Bryan Echaiz Floreano

APROBACIÓN DEL TUTOR

En mi calidad de Tutor del trabajo de integración curricular denominado: "DESARROLLO E IMPLEMENTACIÓN DE ALGORITMOS TIPO INSECTO PARA LA PLANEACIÓN DE RUTAS DE UN ROBOT MÓVIL CON RUEDAS", elaborado por los estudiantes Choez Reyes Julissa Arelys y Echaiz Floreano Rodrigo Bryan, de la Carrera de Electrónica y Automatización de la Facultad de Sistemas y Telecomunicaciones de la Universidad Estatal Península de Santa Elena, me permito declarar que luego de haber orientado, estudiado y revisado, lo apruebo en todas sus partes y autorizo a los estudiantes para que inicien los trámites legales correspondientes.

La libertad, 01 julio del 2024



Ing. Junior Figueroa Olmedo, MSc.

DOCENTE TUTOR

TRIBUNAL DE SUSTENTACIÓN

Ing. Ronald Rovira Jurado, Ph. D.
DIRECTOR DE LA CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

Ing. Carlos Saldaña Enderica, MSc.
DOCENTE ESPECIALISTA

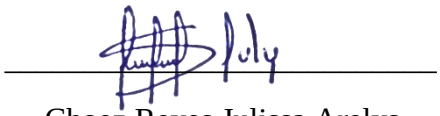
Ing. Junior Figueroa Olmedo, MSc.
DOCENTE TUTOR

Ing. Luis Chuquimarca Jiménez, Mgt.
DOCENTE GUÍA UIC

Ing. Corina Gonzabay De La A, Mgt.
SECRETARIA

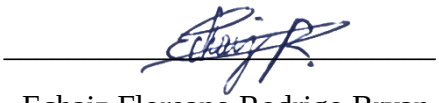
DECLARACIÓN

El contenido del presente trabajo de titulación es de nuestra entera responsabilidad, el patrimonio intelectual del mismo le pertenece a la Universidad Estatal Península de Santa Elena.



Choez Reyes Julissa Arelys

Autor



Echaiz Floreano Rodrigo Bryan

Autor

RESUMEN

El presente trabajo de titulación se enfoca en el diseño y la implementación de tres algoritmos de navegación autónoma, denominados "Insecto 0", "Insecto 1" e "Insecto 2", para la planificación de rutas de un robot móvil con ruedas, específicamente el mBot Neo.

El proyecto tiene como objetivo evaluar el rendimiento de estos algoritmos, para ello se realizarán pruebas en entornos virtuales y reales, proporcionando una evaluación exhaustiva bajo diversas condiciones operativas.

La creación e implementación de estos algoritmos se basa en sólidos principios matemáticos y en el uso de software especializado para su programación. El sistema de percepción avanzado del mBot Neo permitirá analizar su capacidad de navegación autónoma y su adaptación a cambios en el entorno físico.

El objetivo principal es identificar cuál de los tres algoritmos ofrece el mejor rendimiento ante tres escenarios de navegación distintos, identificando así el que ofrece la mejor eficiencia y precisión en la planificación de rutas. Este proyecto representa un avance significativo en la aplicación de algoritmos inspirados en la naturaleza para la robótica autónoma, contribuyendo al desarrollo y conocimiento de tecnologías avanzadas en el campo de la planificación de rutas para robots móviles.

Palabras claves: insecto, robot móvil, planificación de rutas, navegación autónoma, mBot Neo.

ABSTRACT

This degree work focuses on the design and implementation of three autonomous navigation algorithms, named "Insect 0", "Insect 1" and "Insect 2", for the path planning of a wheeled mobile robot, specifically the mBot Neo.

The project aims to evaluate the performance of these algorithms, for this purpose tests will be performed in virtual and real environments, providing a comprehensive evaluation under various operating conditions.

The creation and implementation of these algorithms is based on solid mathematical principles and the use of specialized software for their programming. The advanced perception system of the mBot Neo will allow the analysis of its autonomous navigation capability and its adaptation to changes in the physical environment.

The main objective is to identify which of the three algorithms offers the best performance under three different navigation scenarios, thus identifying the one that offers the best efficiency and accuracy in route planning. This project represents a significant advance in the application of nature-inspired algorithms for autonomous robotics, contributing to the development and understanding of advanced technologies in the field of route planning for mobile robots.

Keywords: bug, mobile robot, path planning, autonomous navigation, mBot Neo.

ÍNDICE GENERAL

ÍNDICE DE FIGURA

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

La robótica autónoma ha experimentado un crecimiento notable en las últimas décadas, impulsada por avances en tecnologías de percepción, procesamiento y control. Entre las diversas estrategias de navegación y planificación de rutas, los algoritmos inspirados en el comportamiento de insectos han surgido como herramientas innovadoras y eficientes. Estos algoritmos, que imitan las capacidades adaptativas y de orientación de los insectos, ofrecen ventajas significativas en la gestión autónoma de trayectorias en robots móviles.

El presente proyecto, titulado “Desarrollo e Implementación de Algoritmos Tipo Insecto para la Planeación de Rutas de un Robot Móvil con Ruedas”, se enfoca en la implementación de tres algoritmos de navegación autónoma denominados "Insecto 0", "Insecto 1" e "Insecto 2", utilizando el robot móvil mBot Neo, equipado con un sistema de percepción avanzado. Se busca determinar cuál de estos algoritmos ofrece las mejores prestaciones, en un análisis comparativo que se llevará a cabo tanto en entornos virtuales como reales, proporcionando una evaluación integral de cada algoritmo bajo diversas condiciones operativas.

En términos de resultados esperados, se busca que el mBot Neo, utilizando cualquiera de los algoritmos tipo insecto, sea capaz de desplazarse eficientemente desde un punto inicial hasta un punto final en un entorno con obstáculos. Este proyecto requerirá la integración de conocimientos avanzados en matemáticas, robótica y programación, aplicados de manera concreta para desarrollar y evaluar los algoritmos. Este enfoque interdisciplinario no solo contribuirá al avance del conocimiento en la planificación autónoma de rutas, sino que también demostrará la viabilidad y ventajas de los algoritmos inspirados en el comportamiento de insectos en la robótica moderna.

CAPÍTULO I

1.1 ANTECEDENTES

En el transcurso de la última década la planificación de rutas es una de las áreas de investigación más importante en robótica desde el punto de vista del ingeniero de control. Muchos de los problemas que se presentan para la interacción de los robots móviles con el entorno se resuelven proponiendo la planificación de rutas, desde algoritmos de control muy simples hasta la elección de la secuencia correcta de acciones; esto se ha aplicado para guiar a los robots a que logren un objetivo específico. La navegación de un robot móvil es un proceso complejo porque depende de la información del entorno y de varios factores influyentes que representan una amenaza para el rendimiento del robot, al sugerir algoritmos adecuados, la planificación de rutas se puede aplicar ampliamente en entornos desconocidos y parcialmente estructurados [1].

Según [2], se han ejecutado varios algoritmos de control para la planeación de rutas, entre ellos están los de tipo insecto de los cuales se ha desarrollado el tipo 0, tipo 1 y tipo 2. En [3], se desarrolló un algoritmo de enjambre para la planeación de rutas inspirado en el comportamiento de las abejas que van en la búsqueda de miel; este algoritmo desde su desarrollo ha sido utilizado para resolver diversos problemas del mundo real.

El algoritmo de optimización metaheurístico es un método el cual fue sugerido para simular el comportamiento inteligente de las abejas artificiales, que van en la búsqueda de fuentes de alimento en función de su propia experiencia y la de otras abejas. Algunas de estas vuelan y eligen fuentes de alimentos al azar, sin experiencia, si la cantidad de néctar de una nueva fuente es mayor en su memoria que la anterior, recordarán el nuevo lugar y olvidarán el anterior. Por lo tanto, el sistema ABC combina los métodos locales de búsqueda de alimento de las abejas empleadas y presentan métodos globales de

búsqueda de alimento controlados por observadores e investigadores que intentan equilibrar el proceso de búsqueda de alimento y explotación. Este algoritmo fue empleado exitosamente para resolver problemas de complejidad de optimización numérica sin restricción extendido a problemas de optimización con restricción [4].

En [5], hace referencia a un nuevo método llamado MBPP (Multi Bug Path Planning), esta metodología la usaron para la planeación de rutas de robots móviles en entornos con obstáculos estáticos conocidos, este método es similar al algoritmo tipo insecto que evalúa las mejores rutas posibles para un entorno dado. Sin embargo, el trabajo propuesto en este artículo fue un intento de combinar las funciones de los métodos offline y online de tal manera que se pueda utilizar el mismo algoritmo en ambos casos, y el resultado de la simulación de estos algoritmos muestran que MBPP encuentran las rutas con tiempos de ejecución más cortos.

En [6], se hizo un análisis comparativo entre dos algoritmos de planeación de rutas, el tipo A*(A-Star) y el algoritmo insecto tipo 0, los cuales fueron ejecutados sobre un robot móvil con ruedas. Este trabajo tuvo como finalidad analizar e investigar los fundamentos matemáticos de los dos algoritmos ya mencionados que luego fueron implementados mediante programación en el software Labview y almacenados en la memoria del controlador del robot móvil. Para verificar la efectividad y el desempeño de los dos algoritmos, se realizaron varias pruebas de funcionamiento para determinar la mejor ruta que debía generar el robot móvil en el proceso de búsqueda de la salida dentro de un laberinto de competencia.

El aporte de esta tesis se centrará en proporcionar una evaluación exhaustiva y comparativa de los algoritmos tipo insecto para la planeación de rutas, destacando sus

ventajas y limitaciones ofreciendo una base sólida para futuras investigaciones y aplicaciones en la robótica móvil.

1.2 DESCRIPCIÓN DEL PROYECTO

Este proyecto tiene como propuesta desarrollar, implementar e investigar los tres tipos de algoritmos tipo insecto conocidos como tipo 0, tipo 1 y tipo 2, que se utilizarán para la planeación de rutas de un robot móvil con ruedas. En esta propuesta se va a realizar un análisis matemático y analítico de dichos algoritmos, con la finalidad de comprender el comportamiento de cada uno de ellos. Posteriormente, estos algoritmos serán traducidos al lenguaje de programación mediante un software especializado (MATLAB y Python) el cual permitirá verificar la validez de estos mediante entornos simulados en donde se plantearán diferentes ambientes con obstáculos considerando la misma postura inicial y final.

El robot mBot Neo con ruedas que se utilizará para la implementación práctica de los tres algoritmos tipo insecto tendrá una configuración diferencial y será programado mediante lenguaje de alto nivel o gráfico. Una vez llevado a cabo las diferentes pruebas en software, se procederá a la ejecución de los tres algoritmos sobre el robot móvil con ruedas en un mismo entorno de movilización con obstáculos similares a los de la simulación. Luego se registrarán un conjunto de datos que permitirán analizar los comportamientos de los tres algoritmos tipo insecto y corroborar la veracidad de las rutas identificadas en las simulaciones, para posteriormente realizar un análisis comparativo con la finalidad de determinar cuál de los algoritmos ofrece las mejores prestaciones. Cabe mencionar que el robot móvil contará con distintos sensores acoplados a su estructura, que ayudarán a percibir el entorno y reconocer los diferentes obstáculos en el proceso de identificación de las rutas; además estos sensores permitirán determinar la dirección que llevará el dispositivo y la velocidad (angular y lineal) de sus ruedas.

1.3 OBJETIVOS DEL PROYECTO

1.3.1 Objetivo general

Desarrollar e implementar tres algoritmos tipo insectos mediante un software especializado y de modo real para procesos de planeación de rutas de un robot móvil con ruedas, con la finalidad de determinar cuál de ellos ofrece las mejores prestaciones.

1.3.2 Objetivos específicos

- Investigar sobre los fundamentos matemáticos de los algoritmos tipo insecto y su empleo en la planeación de rutas, para posteriormente interpretarlos mediante programación utilizando software especializado.
- Analizar los recursos técnicos del robot móvil con ruedas, así como su sistema de percepción para los procesos de navegación con comportamiento autónomo.
- Realizar un análisis comparativo de los posibles softwares especializados que se podrían utilizar en la programación del algoritmo de control tanto en entornos virtuales como físicos para el proceso de planeación de rutas.
- Simular el comportamiento de los algoritmos tipo insecto conocidos como tipo 0, tipo 1 y tipo 2 sobre un entorno virtual para determinar la validez de los mismos.
- Analizar el comportamiento del robot móvil con ruedas en relación al tiempo requerido para realizar el desplazamiento desde un punto de partida hasta un punto de llegada, determinando también las distancias recorridas al emplear cada uno de los algoritmos tipo insecto.
- Verificar la funcionalidad de los algoritmos implementados ejecutándolos sobre el robot móvil con ruedas el cual se movilizará en un entorno real, con la finalidad de determinar una ruta por donde pueda desplazarse y llegar a su postura de destino.

- Evaluar el comportamiento del robot móvil con ruedas en caso de que exista alguna alteración en el entorno físico para verificar cuál de los tres tipos de algoritmos tiene mejor desempeño en su implementación.
- Realizar un análisis comparativo entre los tres tipos de algoritmos para valorar su desempeño en el proceso de planeación de trayectorias y emitir criterios de funcionalidad.

1.4 RESULTADOS ESPERADOS

El presente proyecto busca que mediante cualquiera de los algoritmos tipo insecto analizados e implementados sobre el robot móvil con ruedas, en este caso el mBot Neo, pueda desplazarse en un entorno conocido desde un punto inicial hasta un punto final. Una vez que el robot alcance su meta con los diferentes tipos de algoritmos en las rutas establecidas, se realizara un análisis comparativo teniendo en cuenta los siguientes pasos:

1. Define objetivos e indicadores clave.
2. Identifica los elementos que vas a comparar.
3. Recopilación de los datos a lo largo de las pruebas simuladas y reales en un entorno con obstáculos.
4. Analiza y estudia las diferencias que existen entre los tres tipos de algoritmo tipo insecto.
5. Identificar los comportamientos consistentes que emergen durante la ejecución de los algoritmos tipo insecto (tipo 0, tipo 1 y tipo 2).
6. Interpretación de los resultados en cuanto a eficiencia costos de ejecución para determinar cuál de los tres algoritmos es el que posee mejor desempeño y prestaciones para cumplir el objetivo establecido.

En este proyecto se requiere involucrar las distintas áreas de aprendizaje de la carrera de electrónica y automatización como lo es matemáticas avanzada, para realizar el análisis de las fórmulas de los diferentes algoritmos tipo insecto, robótica para poder reunir la información de los componentes del mBot Neo y como realiza su enlace con los diferentes sensores que posee para poder ejecutar un correcto funcionamiento en sincronía y precisión, además conocimientos medios sobre programación que ayudará a realizar el proceso de crear un conjunto de instrucciones que le dicen al microordenador del mbot (CyberPi), como realizar algún tipo de tarea, el cual se puede programar arrastrando bloques con el software gratuito mBlock basado en Scratch o de forma más avanzada escribiendo código mediante el lenguaje de programación Python.

1.5 JUSTIFICACIÓN

La robótica móvil se considera actualmente un campo tecnológico avanzado para la resolución de problemas muy complejos, como el transporte y la logística en almacenes automatizados, donde los robots móviles gestionan la clasificación, el almacenamiento y la distribución de mercancías de manera eficiente y sin intervención humana.

Existen aplicaciones que constituyen sus productos como en el área de control, programación, inteligencia artificial, percepción e instrumentación y sirven como base para el progreso en todas las industrias, brindando soluciones tecnológicas innovadoras que apuntan a desarrollar mejores robots y extender su alcance en aplicaciones disponibles, sin embargo en ocasiones se presentan limitaciones que surgen de factores relacionados con las imprecisiones de los algoritmos de control, así como las imprecisiones de los elementos relacionados con la instrumentación.

Uno de los aspectos más importante de la robótica móvil es la navegación, un robot móvil debe responder constantemente a las preguntas ¿Dónde estoy?, ¿Hacia dónde me dirijo?,

¿Cómo llego? y ¿Cómo evito obstáculos?, los principales estudios de los últimos años se han relacionado con el progreso tecnológico que surge de desarrollar microcontroladores más rápidos y sistemas de sensores más precisos. Son diferentes los algoritmos que se presentan como una solución que requiere una fase de percepción y otra de adaptación al sistema de control, para especificar y determinar la posición del robot dentro de un entorno que podría ser estructurado o no estructurado [7].

En el campo de la inteligencia artificial es muy común utilizar en el proceso de planificación de rutas funciones heurísticas, las cuales permiten encontrar una ruta que conecte el punto de inicio y el punto final, pero determinar una ruta óptima no está garantizado. Un grupo de algoritmos que se basan en este tipo de funciones son los conocidos como algoritmos de error tipo insecto (bug algorims), los cuales son utilizados en los procesos de navegación de un robot móvil con ruedas en un entorno real, en los cuales la mayoría de los entornos interiores presentan configuraciones con obstáculos más complejas que un entorno abierto con unos pocos obstáculos convexos.

Con esta premisa, este proyecto tiene como finalidad analizar en detalle todos los fundamentos matemáticos que determinan el principio de funcionamiento de los algoritmos tipo insecto aplicados en procesos de planeación de rutas de robots móviles con ruedas, para posteriormente traducir estos modelos matemáticos a un lenguaje de programación de alto nivel con la finalidad de que en un inicio estos algoritmos puedan ser simulados en un entorno virtual y posteriormente ser ejecutados sobre un robot móvil con ruedas que se estará movilizando en un entorno conocido.

Estos algoritmos no forman parte del contenido académico de las asignaturas en la carrera de Electrónica y Automatización. Sin embargo, pueden ser analizados, comprendidos e implementados en aplicaciones específicas, utilizando los conocimientos previos

adquiridos durante la formación del estudiante en materias como software de simulación, matemáticas avanzadas, control continuo y discreto, microcontroladores y robótica industrial, entre otras. Este enfoque permite una comprensión matemática, analítica y práctica, demostrando lo aprendido a lo largo de la carrera universitaria.

METODOLOGÍA

Para el desarrollo de este proyecto, se debe comenzar con la investigación de una variedad de conocimientos relacionados con la navegación de robots móviles con ruedas y algoritmos tipo insecto. Además, se debe investigar sobre las últimas tecnologías y comprender bajo que contexto se realizara el trabajo, posteriormente buscar información relacionada con sistemas, programas y funcionamiento del robot móvil con ruedas, ya que es importante para integrar el código con el cual se va a trabajar con otros sistemas.

Para este trabajo se utilizarán los siguientes tipos de investigaciones.

Investigación diagnóstica

Este tipo de investigación se utilizará para identificar qué factores intervienen en el escenario dado, cuáles serán sus características e implicaciones para poder generar una idea global del contexto del objeto de estudio y así permitir tomar decisiones en función de esa información recopilada y posteriormente analizada.

En este tipo de investigación se identificará los las variables que afectan al algoritmo al momento en el que el robot móvil con ruedas realice el recorrido del punto inicial al punto final a través del entorno conocido, cuáles serán sus características e implicaciones para poder generar una idea global del comportamiento con cada uno de los algoritmos implementados en el robot móvil con ruedas y así permitir formar un criterio y llegar a una conclusión en función de esa información recopilada y posteriormente analizada.

Es decir, la investigación se centrará en primordialmente en el análisis de situaciones, como cambios en el entorno, variación de obstáculos y cambio en el tipo de algoritmo, posterior a este análisis se podrá generar una base adecuada de datos para la evaluación de cada algoritmo para la posterior toma de decisiones sobre su desempeño

Investigación comparativa

Este tipo de investigación consiste en efectuar una comparación lo más exhaustiva posible tanto descriptiva como funcional. A través de estudios comparativos, es posible determinar si las diversas características de las variables son diferentes o similares, esto permite describir los elementos que explican la presencia de un fenómeno en una situación dada.

En la presente forma de investigación se efectuará una comparación detallada tanto descriptiva como funcional de los algoritmos tipo insecto evaluando aspectos como; los

cambios en cada uno de los algoritmos, las diferentes formas del entorno, tanto real como simulado, las variaciones en el código de cada uno de los algoritmos tipo insecto además los cambios en el comportamiento del robot al momento de realizar el desplazamiento del punto inicial al punto final. Para así poder describir que factores influyen en los resultados del desplazamiento del robot desde un punto inicial a un punto final.

CAPÍTULO II

2.1. MARCO CONTEXTUAL

Este proyecto está orientado para su aplicación en el campo de la robótica móvil, que actualmente ha despertado un creciente interés a nivel tanto personal, social e industrial, debido al gran número de aplicaciones donde pueden ser utilizados y en particular los robots móviles con ruedas son de los más empleados.

Decenas de científicos y catedráticos trabajan en proyectos de investigación que involucran el uso de robots móviles, considerando para ello aplicaciones como la planeación de rutas donde la adquisición y procesamiento de información, así como el cálculo matemático juegan un rol de suma importancia.

El proyecto planteado en este documento tiene por finalidad implementar varios algoritmos tipo insectos ejecutados sobre un robot móvil con ruedas con configuración diferencial y esta dirigido a los profesionales y estudiantes que están interesados en el desarrollo e implementación de algoritmos de planeación de rutas sobre robots móviles con ruedas. Los algoritmos tipo insectos que se presentarán en este proyecto se consideran como paradigmas de alto potencial para la planificación de trayectorias, debido a su simplicidad y bajo consumo de memoria [8].

Una vez documentado este proyecto los estudiantes de la carrera de Electrónica y Automatización de la Universidad Estatal Península de Santa Elena (UPSE), tendrán a su disposición documentación detallada respecto al algoritmo matemático que ya ha sido desarrollado y comprobado, y a partir de esta información podrán realizar las modificaciones que consideren pertinentes para generar nuevos proyectos enmarcados en la planeación de rutas de robots móviles con ruedas, además pueden hacer uso de esta información como referencia para futuras publicaciones en revistas. La culminación de

este proyecto también servirá de apoyo para los docentes de la carrera, los cuales tendrán a su disposición esta información que podrán ser utilizadas en algunas de las asignaturas dentro del campo de la robótica o en cursos de capacitación donde puedan tomar la redacción de los algoritmos desarrollados como fuentes bibliográficas para complementar dichas actividades.

Como ya se ha mencionado este trabajo tiene por finalidad investigar e implementar algoritmos tipo insectos, como método computacionalmente eficiente para el proceso de planeación de rutas de robot móviles con ruedas. Sin embargo, la idea general que subyace a los métodos parece ideal para la implementación de robots ligeros, muchas de sus variantes dependen de un sistema de localización global o de sensores perfectos a bordo.

2.2 MARCO CONCEPTUAL

2.2.1 Introducción a los robots móviles

Un robot móvil se define como un sistema electromecánico capaz de moverse según la programación que se le incluye y recibir información sin estar conectado físicamente a un punto. Tiene sensores que monitorean constantemente su posición en relación con los puntos de inicio y final. Por lo general, su control es de lazo cerrado y su movimiento lo proporcionan dispositivos móviles, tales como: ruedas, patas, orugas, etc [9].

En cuanto a los robots móviles no existe un solo enfoque y aquí se presentará una clasificación según el tipo de locomoción: robots terrestres, aéreos y acuáticos [9]:

- **Robots terrestres:** Los robots terrestres son fáciles de implementar y se basan en conceptos mecánicos ya probados, el sistema de locomoción puede ser con patas, orugas o ruedas como se observa en la figura 1 de los cuales se subdividen a continuación [10].

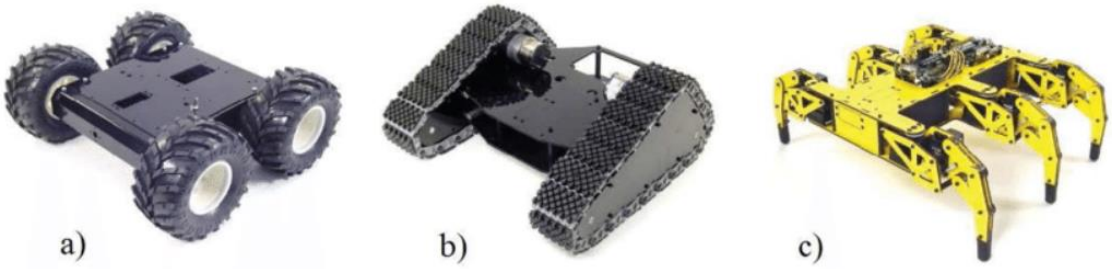


Figura 1. Robots móviles con diversos sistemas de locomoción: a) ruedas, b) orugas, c) con patas [11].

- **Robot con ruedas:** Se utilizan sobre superficies lisas para transportar materiales, mercancías y personas. Dependiendo de la configuración del robot se puede utilizar para diferentes propósitos, las ruedas instaladas son muy maniobrables y eficientes en términos de distancia recorrida y suministro de energía [12].
- **Robot oruga:** Se emplean sobre superficies irregulares y su sistema de locomoción es muy eficaz para el movimiento recto, pero presenta una desventaja al realizar giros ya que su eficiencia es muy baja debido al excesivo rozamiento con la superficie [12].
- **Robot con patas:** Se usan para superficies muy irregulares. Aunque son ineficientemente maniobrables, son excelentes para terrenos irregulares [12].

➢ **Robots aéreos:** Se trata de vehículos aéreos no tripulados que combinan diferentes tipos de aeronaves como se muestra en la figura 2, entre los cuales se pueden mencionar a los drones, aviones y helicópteros [12].

Figura 2. Robot aéreo [13].

- **Robots acuáticos**: como se muestra en la figura 3, estos robots operan en la superficie o por debajo de ella impulsados comúnmente por hélices o turbinas. Estos prototipos robóticos están diseñados para operar en el agua, y muchos de ellos a gran profundidad [10].

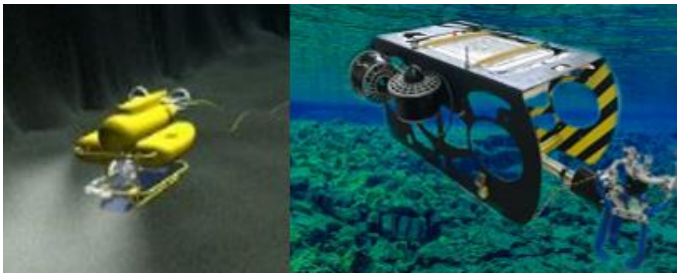


Figura 3. Robots acuáticos [14]**.**

2.2.2 Robots móviles con ruedas

Los robots móviles con ruedas se consideran los más populares porque son sencillos y fáciles de construir, la carga que pueden llevar los robots móviles con ruedas es mayor en comparación a los robots con cadenas y patas que son más complejos y pesados que estos robots. "Los RMR (Robots Móviles con Ruedas) necesitan de la fricción y del contacto con una superficie dura para moverse, una de las técnicas más usadas para estimar el movimiento en los robots móviles es la odometría la cual consiste en la estimación de la distancia recorrida por la cantidad de giros que da la rueda. Respecto a las partes que componen el RMR, existe un arreglo y un sistema cinemático de actuadores que proporcionan movimiento a la estructura cinemática. Entre las limitaciones de este tipo de robot son los deslizamientos y vibraciones, provocados por terrenos blandos e irregulares" [9].

A continuación, se presenta una pequeña clasificación de los robots móviles con ruedas considerando la disposición de las ruedas y sus actuadores sobre la estructura de la plataforma móvil.

2.2.2.1 Robot omnidireccional

El sistema de tracción se basa en el uso de tres ruedas y tracción omnidireccional, como se muestra en la figura 4. Esta configuración tiene tres grados de libertad, por lo que puede realizar cualquier tipo de movimiento y colocarse en cualquier posición u orientación, y no tiene restricciones cinemáticas [15].

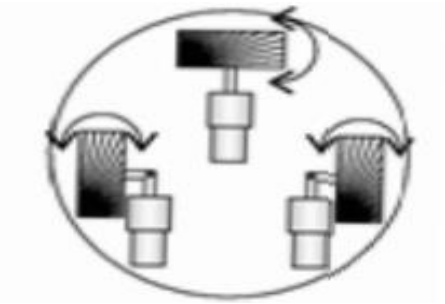


Figura 4. Robot omnidireccional [15]**.**

2.2.2.2 Robot diferencial

Este tipo de robot móvil consta de dos ruedas fijas convencionales coaxiales que se pueden dirigir de forma independiente y una rueda libre que proporciona estabilidad, como se muestra en la figura 5. Debido a su cinemática simple, estructuras mecánicas y electrónicas, estos robots se utilizan ampliamente en experimentos de laboratorio [15].

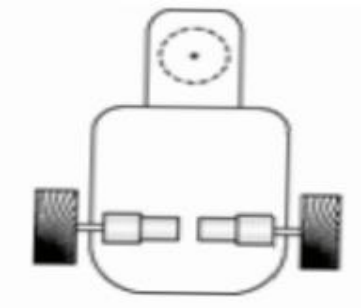


Figura 5. Robot diferencial [15].

2.2.2.3 Robot triciclo clásico

Este tipo de robot consta de una sola rueda en la parte delantera que se utiliza tanto para la tracción como para la dirección. El eje de la parte trasera con dos ruedas laterales es pasivo y cada una de ellas pueden moverse libremente como se muestra en la figura 6. Su

maniobrabilidad es superior al sistema Ackerman porque solo tiene una rueda, pero presenta problemas de estabilidad en terrenos difíciles [15].

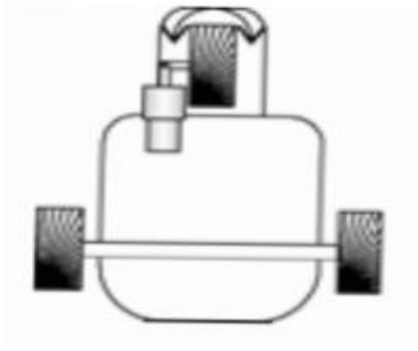


Figura 6. Robot triciclo [15].

2.2.2.4 Robot ackerman

Como se muestra en la figura 7, el modelo Ackerman es utilizado en un robot móvil con cuatro ruedas convencionales de las cuales tiene dos ruedas para la tracción y se montan paralelas a la parte trasera del chasis principal del robot móvil, y dos ruedas delanteras que se utilizan para la dirección, cada rueda de la parte delantera tiene dos ángulos de dirección diferentes, lo que provoca problemas de direccionamiento. En muchos casos el ángulo de direccionamiento se combina con uno solo, sin embargo, los dos caminos no serán paralelos por lo que las ruedas girarán en las esquinas y su mayor problema es la movilidad limitada [15].

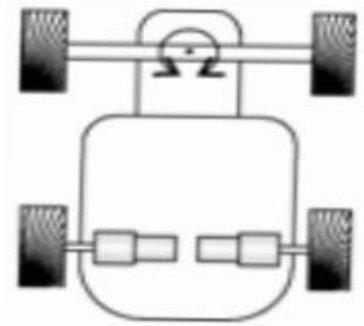


Figura 7. Robot Ackerman [15].

2.2.2.5 Robot skid steer

Como se observa en la figura 8, este tipo de robot móvil tiene varias ruedas en cada lado las cuales se mueven simultáneamente. El movimiento que genera el robot es resultado de combinar la velocidad de las ruedas de izquierda y derecha. Un ejemplo son los

vehículos tipo oruga los cuales se usan en pistas con deslizamiento como medio de locomoción para ver la impulsión como el direccionamiento que consigue el robot por medio de esta pista [15].

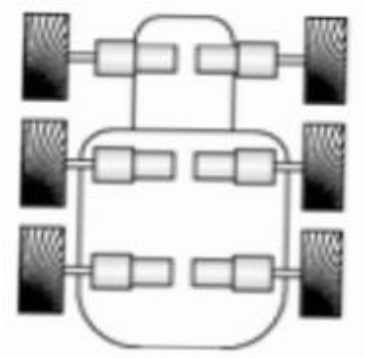


Figura 8. Robot Skid steer [15].

2.2.2.6 Robot síncrono

Consiste en el funcionamiento simultáneo de todas las ruedas que giran sincrónicamente, la conexión se realiza mediante coronas de engranajes. Este tipo de robot móvil consta de tres ruedas las cuales se conectan en los vértices de un triángulo equilátero debajo de una plataforma cilíndrica, como se muestra en la figura 9 [15].

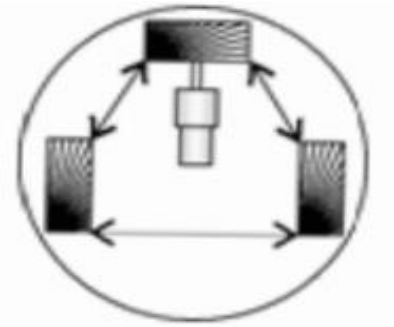


Figura 9. Robot Síncrono [15].

2.2.3 Tipos de ruedas para locomoción

Como definición se puede decir que las ruedas son una de las partes más importantes que tiene un robot móvil, ya que proporcionan la tracción necesaria para el robot. Entre los tipos de ruedas que se utilizan en RMR destacan: ruedas convencionales, tipo castor, tipo bola y omnidireccionales. En cuanto al sistema del actuador, tiene la tarea de generar en

los elementos el movimiento necesario para el robot basados en el orden especificado en la unidad de control.

2.2.3.1 Ruedas convencionales

Las ruedas convencionales posibilitan obtener desplazamientos siempre y cuando se empleen configuraciones de posición apropiadas para el robot. Estas ruedas pueden ser clasificadas de acuerdo con la posición del eje de rotación respecto de la rueda. A su vez entre las ruedas convencionales se distinguen tres tipos como se aprecia en la figura 10 *[16]*.

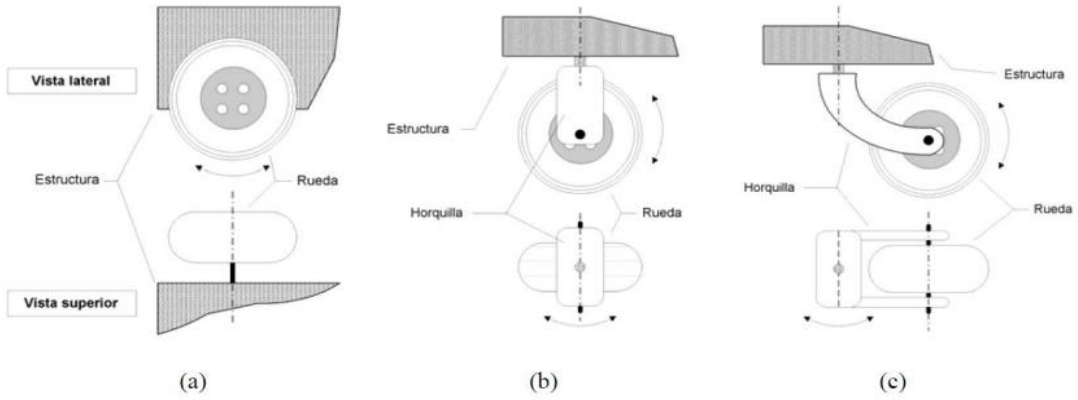


Figura 10. Tipos de ruedas: (a) Rueda fija, (b) Rueda orientable centrada, (c) Rueda loca [16].

- **Rueda fija:** los ejes se fijan a la estructura que tiene el robot. Por lo general, está relacionado con el sistema de tracción del robot móvil [16].
- **Rueda orientable centrada:** Es aquella en la que el movimiento del plano de la rueda con respecto a la estructura es una rotación alrededor de un eje vertical que pasa a través del centro de la rueda. Suele cumplir funciones como rueda de dirección o como rueda de tracción-dirección [16].
- **Rueda orientable descentrada:** También conocidas como rueda castor (*castor wheel*), esta es un tipo de rueda diseñada para que el plano de la rueda gire alrededor de un eje vertical y no cruce el centro de la rueda, y su función principal es dar

estabilidad a la estructura mecánica que tiene el robot como una rueda de direccionamiento [16].

2.2.3.2 Ruedas tipo bola

Las ruedas tipo bola consiste en una rueda de forma esférica con un tipo de mecanismo de accionamiento. Funcionalmente, la rueda esférica corresponde a la rueda sueca, como se muestra en la figura 11. Las limitaciones de movimiento vienen determinadas por la posible acción de ruedas y/o rodillos durante la rotación en contacto con el suelo. Generalmente, el movimiento está limitado por ruedas de bolas y pasadores en una sola dirección o en cualquier dirección si se consideran ruedas libres [17].

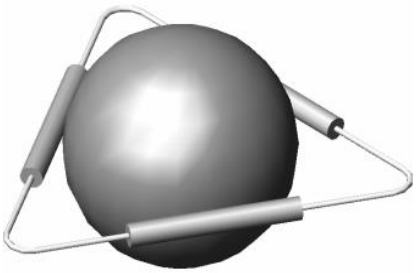


Figura 11. Rueda tipo bola con tres rodillos formando un rectángulo horizontal [17].

2.2.3.3 Ruedas omnidireccionales o suecas

Como se observa en la figura 12 este tipo de ruedas tienen rodillos en su circunferencia la cual se la define como rueda estándar, es dotada de una corona de rodillos, cuyos ejes de giros son perpendicular al sentido normal de la marcha, de forma que cuando se aplica una fuerza lateral, el rodillo gira sobre sí mismo y permite que la componente de velocidad del vector X sea cero, eliminando la restricción de no holomicidad, este tipo de rodamiento se utiliza en robots omnidireccionales [18].

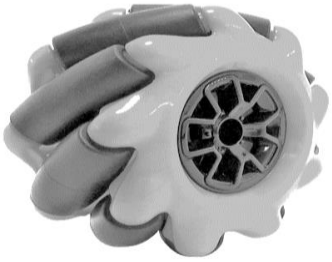


Figura 12. Rueda Omnidireccional [18].

2.2.4 Cinemática de un robot móvil diferencial

Existen muchos tipos de modelos cinemáticos a continuación se mencionan los más comunes:

- La cinemática interna explica la relación entre las variables internas del sistema (por ejemplo, la rotación de la rueda y el movimiento del robot) [19].
- La cinemática externa describe la posición y la orientación del robot según algún marco de coordenadas de referencia [19].
- Cinemática directa y cinemática inversa. Una cinemática directa describe estados del robot en función de sus entradas (velocidades de las ruedas, movimiento de las articulaciones, volante, etc.). A partir de la cinemática inversa se puede diseñar un movimiento planificado, lo que significa que las entradas del robot se pueden calcular para una secuencia deseada del estado del robot [19].
- Las restricciones de movimiento aparecen cuando un sistema tiene menos variables de entrada que grados de libertad (DOF). Las restricciones holonómicas permiten generar diferentes posturas del robot sin que el chasis tenga que reorientarse, mientras que una restricción no holonómica prohíbe ciertas posturas del robot (el robot puede conducir solo en la dirección rotación de las ruedas) [19].

A continuación, se va a determinar el modelo cinemático interno para un robot móvil diferencial. Para cualquier robot móvil con ruedas la postura del mismo en un plano dado se representa mediante el siguiente vector de estado.

$$q(t) = \begin{pmatrix} x(t) \\ y(t) \\ \varphi(t) \end{pmatrix}$$

En un sistema de coordenadas global (X_g, Y_g), como se ilustra en la figura 13 un movimiento marco (X_m, Y_m) está unido al robot. La relación entre el mundo y el marco

móvil (cinemática externa) está definido por el vector de traducción $[x, y]^T$ y matriz de rotación [19]:

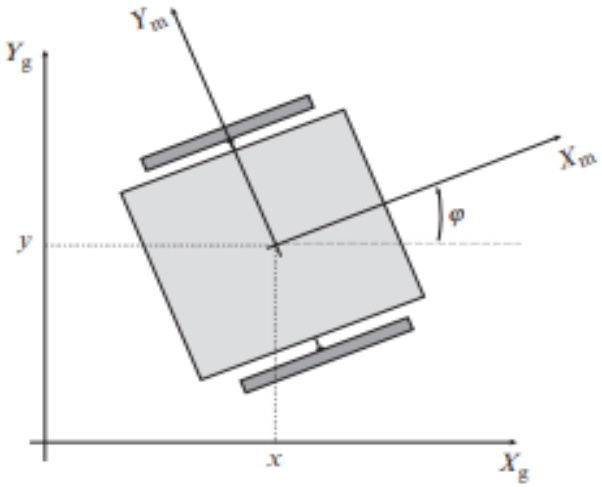


Figura 13. Robot en el Plano [19].

$$R(\varphi) = \begin{pmatrix} cos(\varphi) & sin(\varphi) & 0 \\ -sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2.2.4.1 Accionamiento diferencial

El accionamiento diferencial es un mecanismo de accionamiento muy simple que, con bastante frecuencia utilizado en la práctica, especialmente para robots móviles más pequeños. Robots con esta unidad generalmente tienen una o más ruedas giratorias para sostener el vehículo y evitar inclinación. Ambas ruedas principales están colocadas sobre un eje común. la velocidad de cada rueda está controlada por un motor separado. De acuerdo con la figura 14 [19].

De acuerdo con la figura 14, las variables de entrada (control) son la velocidad de la rueda derecha v_d y la velocidad de la rueda izquierdav_i, en donde para el análisis se asume que $v_d > v_i$. R es el radio instantáneo de la trayectoria de conducción del robot y que corresponde a la distancia entre el centro del chasis (punto medio entre las ruedas activas)

y el punto ICR. En cada instancia de tiempo, ambas ruedas tienen la misma velocidad angular ω alrededor del ICR *[20]*.

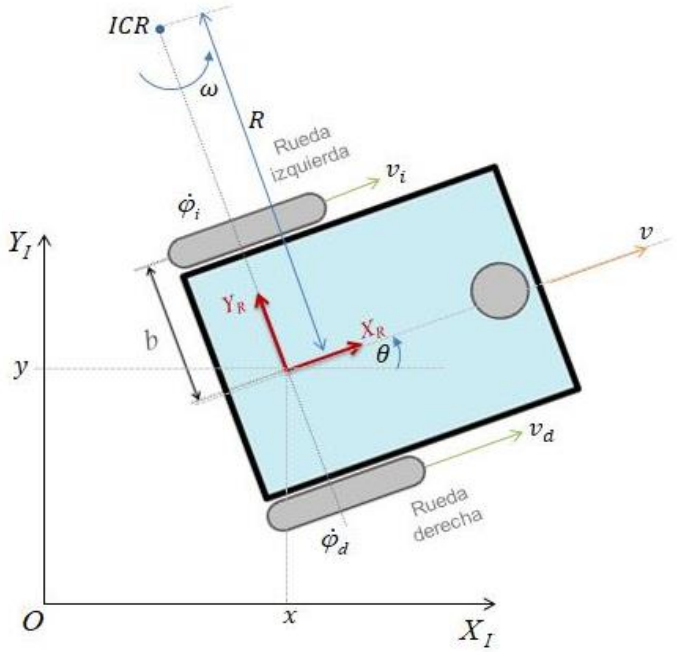


Figura 14. Geometría de un robot con accionamiento diferencial [20].

Las velocidades líneas de cada rueda se expresan de la siguiente manera [20].:

$$v_d = \omega(R + b) \qquad v_i = \omega(R - b) \qquad \text{Ec (2.1)}$$

Resolviendo las ecuaciones anteriores podemos determinar ω y R [20].

$$\omega = \frac{v_d - v_i}{2b} \qquad \text{Ec (2.2)}$$

$$R = \frac{b(v_d + v_i)}{v_d - v_i} \qquad \text{Ec (2.3)}$$

La velocidad tangencial para el robot se la calcula como [20].:

$$v = \omega R = \frac{v_d + v_i}{2} \qquad \text{Ec (2.4)}$$

Las velocidades tangenciales de las ruedas son $v_i = r\dot{\varphi}_i(t)$ y $v_d = r\dot{\varphi}_d$, donde $\dot{\varphi}_i$ y $\dot{\varphi}_d$son las velocidades angulares izquierda y derecha de las ruedas alrededor de sus ejes, respectivamente [20].

$$v = \frac{r\dot{\varphi}_d + r\dot{\varphi}_i}{2} = \frac{r}{2}\dot{\varphi}_d + \frac{r}{2}\dot{\varphi}_i \qquad \text{Ec (2.5)}$$

$$\omega = \frac{r\dot{\varphi}_d - r\dot{\varphi}_i}{2b} = \frac{r}{2b}\dot{\varphi}_d - \frac{r}{2b}\dot{\varphi}_i \qquad \text{Ec (2.6)}$$

Teniendo en cuenta las relaciones anteriores, la cinemática del robot (en coordenadas locales) se puede expresar como [20]:

$$\begin{bmatrix} \dot{R}_{\mathrm{m}} \\ \dot{R}_{\mathrm{m}} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v_{\mathrm{x}} \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2b} & -\frac{r}{2b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_d \\ \dot{\varphi}_i \end{bmatrix} \qquad \text{Ec (2.7)}$$

2.2.4.2 Cinemática directa e inversa

La postura del robot en algún momento (t) se obtiene por integración de la cinemática modelo, que se conoce como odometría o navegación a estima. Determinación de la postura del robot para las variables de control dadas se llama directa cinemática (también hacia adelante) [19]:

$$x(t) = \int_0^t v(t)\cos\left(\varphi(t)\right)dt \qquad \text{Ec (2.8)}$$

$$y(t) = \int_0^t v(t)\sin\left(\varphi(t)\right)dt \qquad \text{Ec (2.9)}$$

$$\varphi(t) = \int_0^t \omega(t)dt \qquad \text{Ec (2.10)}$$

Si se suponen velocidades constantes v y ω durante el tiempo de muestra, la integración de las ecuaciones, (2.8), (2.9) y (2.10) se puede hacer numéricamente usando el método de Euler. La cinemática directa viene dada entonces por [19]:

$$x(k+1) = x(k) + v(k)T_S\cos(\varphi(k)) \quad \text{Ec (2.11)}$$

$$y(k+1) = y(k) + v(k)T_S\sin(\varphi(k)) \quad \text{Ec (2.12)}$$

$$\varphi(k+1) = \varphi(k) + \omega(k)T_S \quad \text{Ec (2.13)}$$

Si se aplica la integración exacta usando las ecuaciones (2.11), (2.12) y (2.13), la cinemática directa es [19]:

$$x(k+1) = x(k) + \frac{v(k)}{\omega(k)}\ sin(\varphi(k) + \omega(k)T_S) - \ sin(\varphi(k))) \quad \text{Ec (2.14)}$$

$$y(k+1) = y(k) - \frac{v(k)}{\omega(k)}\ cos(\varphi(k) + \omega(k)T_S) - \ cos(\varphi(k))) \quad \text{Ec (2.15)}$$

$$\varphi(k+1) = \varphi(k) + \omega(k)T_S \quad \text{Ec (2.16)}$$

Donde la integración de las ecuaciones (2.14), (2.15) y (2.16) se realizan dentro del intervalo de tiempo de muestreo donde se suponen velocidades constantes v y w para obtener incrementos [19]:

$$\Delta x(k) = v(k)\int_{kT_S}^{(k+1)T_S} \cos(\varphi(t))dt$$

$$= v(k)\int_{kT_S}^{(k+1)T_S} \cos(\varphi(k) + \omega(k)(t - kT_S))dt \quad \text{Ec (2.17)}$$

$$\Delta y(k) = v(k)\int_{kT_S}^{(k+1)T_S} \sin(\varphi(t))dt$$

$$= v(k)\int_{kT_S}^{(k+1)T_S} \sin(\varphi(k) + \omega(k)(t - kT_S))dt \quad \text{Ec (2.18)}$$

El desarrollo de la cinemática inversa es una tarea más desafiante que los casos anteriores de cinemática directa. Usamos la cinemática inversa para determinar las variables de control para conducir el robot a la posición deseada del robot o la trayectoria de la ruta. Los robots suelen estar sujetos a restricciones no holonómicas, lo que significa que no todas las direcciones de conducción son posibles. también hay muchas soluciones posibles para llegar a la postura deseada [19].

Una solución simple al problema de la cinemática inversa sería si permitiéramos que un robot diferencial se moviera solo hacia adelante($v_R(t) = v_L(t) = v_R \Rightarrow \omega(t) = 0, v(t) = v_R$) o girará en el lugar ($v_R(t) = -v_L(t) = v_R \Rightarrow \omega(t) = \frac{2v_R(t)}{L}, v(t) = 0$) a velocidades constantes. Para el movimiento de rotación, las ecuaciones (2.11), (2.12) y (2.13) se simplifican a [19]:

$$x(t) = x(0) \qquad \text{Ec (2.19)}$$

$$y(t) = y(0) \qquad \text{Ec (2.20)}$$

$$\varphi(t) = \varphi(0) + \frac{2v_R t}{L} \qquad \text{Ec (2.21)}$$

y para el movimiento rectilíneo se simplifican a:

$$x(t) = x(0) + v_R \cos\big(\varphi(0)\big)t \qquad \text{Ec (2.20)}$$

$$y(t) = y(0) + v_R \sin\big(\varphi(0)\big)t \qquad \text{Ec (2.21)}$$

$$\varphi(t) = \varphi(0) \qquad \text{Ec (2.22)}$$

La estrategia de movimiento podría entonces ser orientar el robot a la posición de destino mediante rotación y luego conducir el robot a la posición de destino mediante un movimiento recto y finalmente alinear (con rotación) la orientación del robot con la orientación deseada en la postura del robot deseada. Las variables de control requeridas

para cada fase (rotación, movimiento rectilíneo, rotación) se pueden calcular fácilmente a partir de las ecuaciones (2.12) y (2.13) [19].

Si consideramos una notación de tiempo discreto donde las velocidades de control $v_R(k), v_L(k)$ son constantes durante el intervalo de tiempo T_s y los cambios de las velocidades de control solo son posibles en instantes de tiempo $t = kT_s$ entonces podemos escribir ecuaciones para el movimiento del robot. Para el movimiento de rotación $v_R(k) = -v_L(k)$ siguen las ecuaciones (2.23), (2.24) y (2.25) [19].

$$x(k+1) = x(k) \qquad \text{Ec (2.23)}$$

$$y(k+1) = y(k) \qquad \text{Ec (2.24)}$$

$$\varphi(k+1) = \varphi(k) + \frac{2v_R(k)T_s}{L} \qquad \text{Ec (2.25)}$$

y las ecuaciones de movimiento rectilíneo $v_R(k) = v_L(k)$ es:

$$x(k+1) = x(k) + v_R\,(\mathrm{k})\cos\big(\varphi(k)\big)t \qquad \text{Ec (2.26)}$$

$$y(k+1) = y(k) + v_R\,(\mathrm{k})\sin\big(\varphi(k)\big)t \qquad \text{Ec (2.27)}$$

$$\varphi(k+1) = \varphi(k) \qquad \text{Ec (2.28)}$$

Entonces, para el movimiento del robot deseado dentro del intervalo de tiempo $t \in (\mathrm{k}T_s, (k+1)T_s$ la cinemática inversa se puede calcular para cada tiempo de muestra expresando variables de control de las ecuaciones (2.23), (2.24) y (2.25) [19].

Como ya se ha dicho, hay muchas otras soluciones para conducir el robot a la postura deseada usando trayectorias suaves y cambiantes. El problema cinemático inverso es más fácil para la trayectoria suave deseada del objetivo $(x(t), y(t))$ que debe seguir el robot para que su orientación sea siempre tangente a la trayectoria. La trayectoria se define en el intervalo de tiempo $t \in [0, T]$. Suponiendo que la postura inicial del robot está en la

trayectoria y hay un modelo cinemático perfecto y sin perturbaciones, podemos calcular las variables de control requeridas v de la siguiente manera [19]:

$$v(t) = \pm\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \qquad \text{Ec (2.29)}$$

2.2.5 Planeación de rutas

Una ruta o trayectoria en definición es una secuencia de puntos o configuraciones de las cuales las podemos determinar algorítmicamente en un mapa del entorno, que luego se procesa para que el robot pueda rastrearlos usando expresiones matemáticas para alcanzar una meta determinada. “En robótica, la planificación de rutas se refiere al problema de cómo mover un robot de un punto a otro. En inteligencia artificial, la planificación de trayectorias significa encontrar una secuencia lógica de acciones que transforme la postura inicial del robot a una postura de destino deseada. En la teoría de control, la planificación de rutas se ocupa de cuestiones de estabilidad, retroalimentación y optimización. Por lo tanto, la planificación de rutas para un robot móvil es un problema amplio y existen muchos métodos y enfoques” [21].

La planificación de rutas para robots móviles puede ocurrir en los siguientes tipos de entornos estáticos y dinámicos. En el caso estático, el obstáculo no se mueve y se conoce completamente el área de trabajo, mientras que, en el caso dinámico se considera que el obstáculo se está moviendo por lo que no se conoce completamente el área de trabajo. Cuando el entorno es estático, se realiza una planificación fuera de línea porque el espacio de trabajo no cambia a lo largo de la trayectoria; Si bien el entorno es dinámico, el espacio de trabajo cambia constantemente, lo que obliga al algoritmo a generar nuevas rutas a través de la planificación en línea [21]. El objetivo de un algoritmo de planificación de rutas es explorar un espacio con poca o ninguna información para encontrar una ruta que pueda conectar dos puntos de coordenadas; el diseño considerará

un espacio bidimensional ubicado en el plano XY para simplificar la tarea de búsqueda [22].

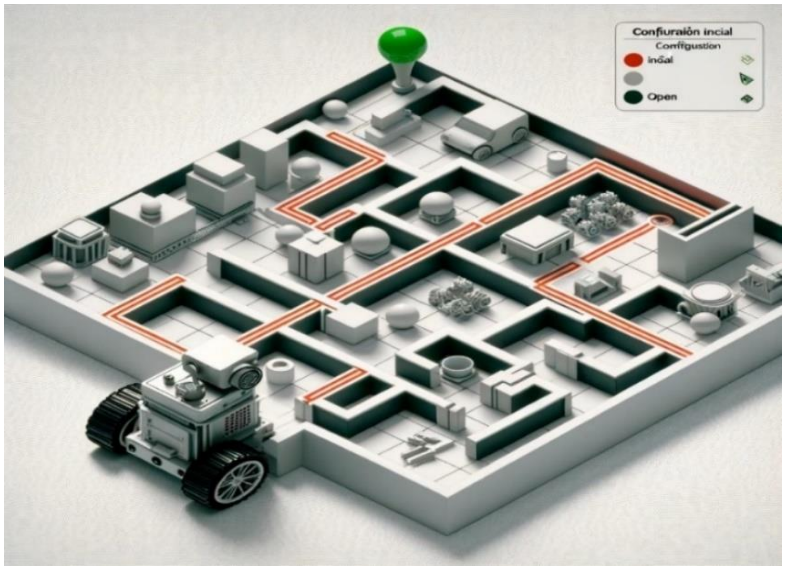


Figura 15. Planeación de Rutas.

2.2.6 Algoritmos tipo insecto

Los algoritmos tipo insecto son los algoritmos de planeación de rutas más simples que asumen solo el conocimiento local y no necesitan un mapa del entorno. Por lo tanto, son apropiados en situaciones en las que se desconoce un mapa del entorno o está cambiando rápidamente y también cuando el controlador del robot móvil tiene una potencia computacional muy limitada. Estos algoritmos utilizan la información local suministrada por sus sensores y la información global del punto objetivo. Su funcionamiento consta de dos comportamientos simples: movimiento en línea recta hacia la meta y seguimiento del límite de obstáculos [8].

“Los robots móviles que utilizan estos algoritmos pueden evitar obstáculos y avanzar hacia la meta. Estos algoritmos requieren poco uso de memoria.” Existen 3 versiones básicas del algoritmo tipo Insecto de las cuales se describe a continuación [8].

Tipo Insecto 0

Un algoritmo tipo insecto 0 tiene dos comportamientos de los cuales detallamos a continuación:

1. Se mueve hacia la meta hasta que detecta un obstáculo o alcanza la meta [8].
2. Si se detecta un obstáculo, girar a la izquierda (o a la derecha, pero siempre en la misma dirección) y seguir el contorno del obstáculo hasta que se mueva en la misma dirección. misma dirección) y seguir el contorno del obstáculo hasta que el movimiento en línea recta hacia la meta [8].

Un ejemplo del funcionamiento del algoritmo tipo insecto 0 se presenta en a continuación la figura 16.

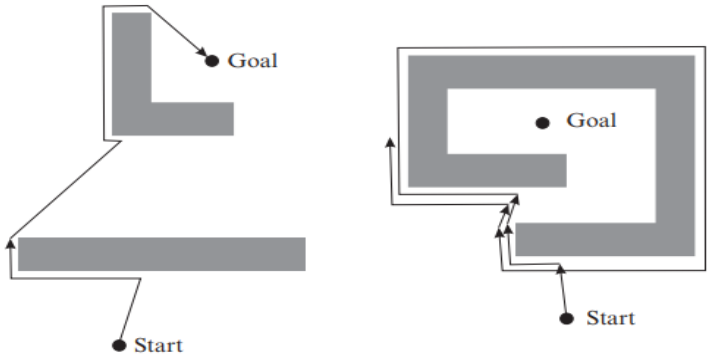


Figura 16. Demostración del algoritmo tipo insecto 0 [19].

Tipo Insecto 1

Un algoritmo insecto 1 en comparación con el insecto 0 utiliza más memoria y requiere cálculos adicionales. En cada iteración necesita calcular la distancia euclidiana euclídea a la meta y recordar el punto de la circunferencia del obstáculo más cercano a la meta. Su funcionamiento se describe de la siguiente manera [19].

1. Se mueve en línea recta hacia la meta hasta que choca con un obstáculo o alcanza la meta [19].

2. Si se detecta un obstáculo, entonces se gira a la izquierda y se sigue todo el contorno del obstáculo y se mide la distancia euclidiana a la meta. Cuando el punto en el que se detectó inicialmente el obstáculo siga el contorno del obstáculo en la dirección más corta hasta el punto del contorno más cercano a la meta. A continuación, se reanuda el movimiento hacia la meta en línea recta [19].

Un ejemplo de funcionamiento del algoritmo tipo insecto 1 se aprecia en la figura 17.

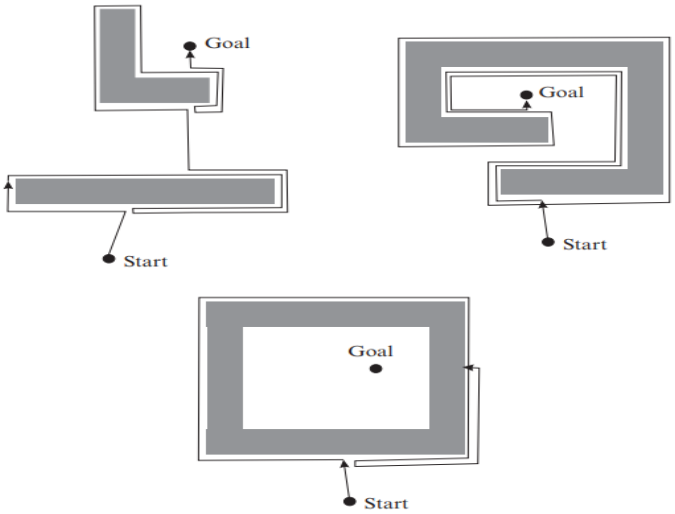


Figura 17. Demostración del algoritmo tipo insecto1 el cual puede identificar la meta inalcanzable [19].

La trayectoria obtenida no es óptima y es, en el peor de los casos, para 3/2 de la longitud de todos los contornos de obstáculos más larga que la distancia euclidiana desde el inicio hasta la configuración de la meta. Para cada obstáculo detectado en su trayectoria desde el inicio hasta la meta, el algoritmo sólo encuentra un punto de entrada y un punto de salida del contorno del obstáculo. Por lo tanto, nunca detecta el obstáculo más de una vez y, por lo tanto, nunca circula entre los mismos obstáculos. Cuando el algoritmo detecta el mismo obstáculo más de una vez, sabe que el punto de partida o el punto de llegada se encuentran dentro del obstáculo y puede finalizar la búsqueda del camino, ya que no existe ningún camino factible hacia el objetivo [19].

Tipo Insecto 2

El algoritmo tipo insecto 2 siempre intenta moverse por la línea principal que se define como una línea recta que une el punto de partida y el punto de llegada. Funciona repitiendo los siguientes pasos [19]:

1. Desplazarse por la línea principal hasta chocar con un obstáculo o alcanzar el punto de meta. En este último caso finaliza la búsqueda de la trayectoria [19].
2. Si se detecta un obstáculo, siga el contorno del obstáculo hasta que se alcance la línea principal en la que la distancia euclidiana al punto de destino sea menor, principal en la que la distancia euclidiana al punto de destino sea menor que la distancia euclidiana al punto de destino que la distancia euclídea desde el punto en que se detectó el obstáculo por primera vez detectado [19].

Aunque el algoritmo tipo insecto 2 parece en general mucho más eficaz que tipo insecto 1, no garantiza que el robot detecte determinados obstáculos sólo una vez. En algunas configuraciones de obstáculos, el robot con tipo insecto 2 podría repetir innecesariamente los círculos de obstáculos hasta alcanzar el punto de meta, como se ilustra en la parte derecha de la figura 18 [19].

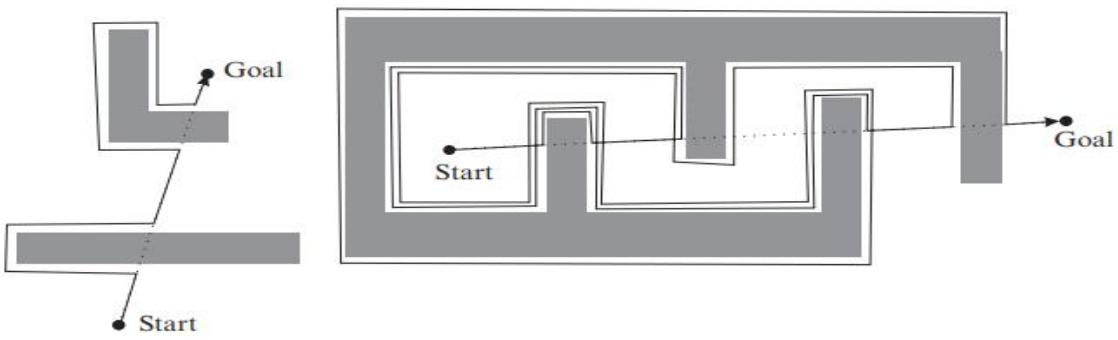


Figura 18. El algoritmo tipo insecto 2 puede identificar un objetivo inalcanzable [19].

2.2.7 Software

Como lenguaje de programación se ha decidido utilizar MATLAB, Scratch, y Python para el desarrollo de este proyecto, considerando las diferentes ventajas que presentan

estos softwares tanto para la simulación del proyecto y la implementación física del mismo. A continuación, se mencionan varias características de los mismos.

MATLAB: Es un lenguaje de alto nivel diseñado para su uso en entornos informáticos técnicos de alto rendimiento para computación, visualización y análisis numérico, cálculos matriciales, procesamiento de señales e integración de gráficos. Proporciona un entorno amigable donde los problemas y las soluciones se pueden expresar matemáticamente sin estar en la programación tradicional [23].

Una parte importante de la robótica en MATLAB es el diseño de sistemas autónomos o semiautónomos capaces de interactuar con el entorno modificando con el que está equipado de: sensores, actuadores y controladores. Programar un robot es diseñar un controlador que controle el comportamiento del robot. A medida que la robótica se vuelve más compleja, el modelado y la simulación se convierten en métodos esenciales para que los ingenieros comprendan el comportamiento de los sistemas robóticos, cómo funcionan los controladores para procesar la percepción de un robot sobre su entorno y gestionar su movilidad e interacción con dicho entorno. Investigadores e ingenieros de robótica utilizan MATLAB para diseñar, simular y verificar todos los aspectos de los sistemas autónomos, desde la percepción hasta el movimiento [24].

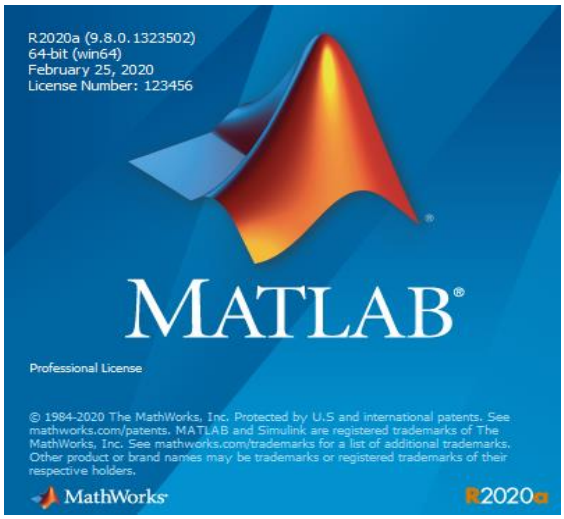


Figura 19. Icono de MATLAB.

Scratch: Es un lenguaje de programación visual - gráfico en el que se permite el uso de condiciones, eventos y métodos en forma de bloques de código para crear aplicaciones interactivas. La plataforma, que lleva el nombre del lenguaje, es donde se alojan todos los recursos y entornos de desarrollo [25].

Figura 20. Icono del programa Scratch [25].

2.3 MARCO TEÓRICO

En esta sección se presentara resumenes de tesis de grados que estan relacionadas con la presente propuesta de titulación, de las cuales fueron analizadas y que sirvieron de guia para este proyecto.

El trabajo de titulación "NAVEGACIÓN DE ROBOTS MÓVILES UTILIZANDO LOS ALGORITMOS INSECTOS EXTENDIDOS" realizado en Benemérita Universidad Autónoma de Puebla, en el año 2018, por Mario Serna Hernández, se realizó un estudio de los algoritmos tipo insecto 0, insecto 1, insecto 2, Dist-insecto, Intelligent-insecto, Intensity-Based-insecto y Tangent-insecto, para dar una solución al problema de planificación de trayectorias, estos algoritmos fueron implementados y probados tanto en un entorno simulado como en uno real, de los cuales se utilizó el robot Pioneer 3-DX con el objetivo de que las pruebas realizadas en el entorno virtual sean muy similares a las reales. Para la detección de obstáculos se utilizó un sensor láser y para realizar la prueba se construyeron varios mapas con obstáculos de diversas formas teniendo en cuenta las debilidades y fortalezas de los algoritmos, con el objetivo de que sea más sencillo apreciar las diferencias entre los comportamientos y las trayectorias generadas. En este proyecto

se realizarón comparaciones del rendimiento de los algoritmos tomando como referencia el tiempo de ejecución y la longitud total de la trayectoria [26].

En un artículo publicado por Andres Vaca en la Universidad Técnica del Norte, Ibarra, Ecuador en el año 2018, con el tema ROBOT MÓVIL PARA INVESTIGACIÓN EN ALGORITMOS DE PLANEAMIENTO DE RUTAS SISTEMA DE ODOMETRÍA, el cual consiste en desarrollar un robot móvil con hardware de gran capacidad y software propietario existentes en los laboratorios de los cuales la plataforma servirá para desarrollar investigaciones relacionadas con robótica, programación, visión artificial, control y automatización. En la realización del robot móvil utilizaron motores acoplados a llantas para su locomoción, colocados en configuración diferencial para estimar el desplazamiento y la rotación sobre su eje además, este cuenta con un sistema de medición basados en codificadores incrementales utilizados para la odometría, situados en cada uno de los motores. Dichas señales son procesadas por un sistema embebido sbRIO usado para el control y adquisición de datos en tiempo real. Existe una interfaz gráfica humano – robot desarrollado en LabVIEW que permite manipular al robot y observar las señales de los sensores a través de indicadores fáciles de interpretar [27].

En el año 2022, en la Universidad Politécnica Salesiana Sede Quito, José Vicente Arias Ganchala y Alex Javier García Patiño mediante su tema de titulación “ANÁLISIS COMPARATIVO ENTRE EL ALGORITMO A* Y EL ALGORITMO TIPO INSECTO CERO PARA LA PLANEACIÓN DE RUTAS APLICADOS SOBRE UN ROBOT MÓVIL CON RUEDAS” realizarón un análisis comparativo entre dos algoritmos de planeación de rutas, el tipo A*(A-Star) y el algoritmo insecto tipo 0, los cuales fueron ejecutados sobre un robot móvil con ruedas. Este trabajo tuvo como finalidad analizar e investigar los fundamentos matemáticos de los dos algoritmos ya mencionados que luego fueron implementados mediante programación en el software Labview y almacenados en

la memoria del controlador del robot móvil. Para verificar la efectividad y el desempeño de los dos algoritmos, se realizaron varias pruebas de funcionamiento para determinar la mejor ruta que debía generar el robot móvil en el proceso de búsqueda de la salida dentro de un laberinto de competencia [6].

CAPITULO III

3.1 COMPONENTES DE LA PROPUESTA

3.1.1 Componentes físicos

Los componentes físicos son responsables de ejecutar la implementación práctica. A continuación, se detallan todos los componentes utilizados para llevar a cabo esta propuesta de investigación.

3.1.1.1 Descripción general del kit mBot Neo

El mBot Neo es un kit educativo de robótica desarrollado y producido por la empresa Makeblock. Como se observa en la figura 21 este kit ofrece una amplia gama de componentes específicos de robótica, piezas estructurales y un entorno de programación amigable que lo hace ideal para la educación. A pesar de que a veces se le clasifica como un juego, sus capacidades en el ámbito educativo son notables y permiten la creación de proyectos prácticos y educativos.

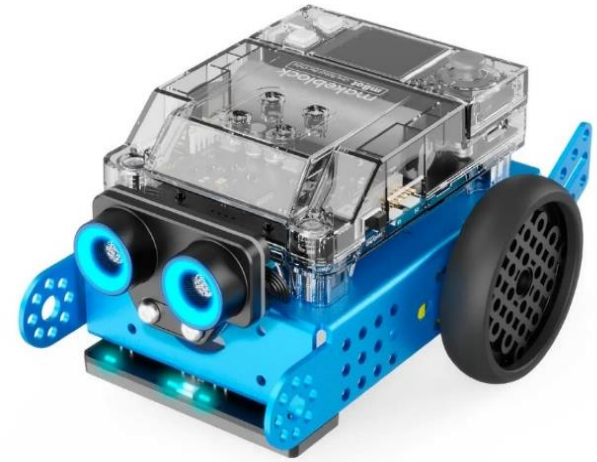


Figura 21. Robot móvil mBot Neo [28].

El mBot Neo, al igual que el sistema sensorial en los seres vivos, recopila información a través de sus sensores y utiliza esta información para planificar y ejecutar movimientos dentro del espacio en el que se encuentra. Las aplicaciones creadas en esta plataforma pueden recibir datos de los sensores y controlar los actuadores para interactuar con el

entorno, lo que lo convierte en una herramienta poderosa para proyectos principalmente orientados a la robótica y la automatización. El kit básico de mBot Neo (ver figura 22) incluye un módulo controlador programable CyberPi ESP32, dos motores DC con codificadores, varios sensores (ultrasónico, Sharp, acelerómetro, giroscópico, luz y de color), una batería recargable, cables y tornillos para conectar los diferentes sensores y actuadores, un cable USB, un chasis metálico, dos llantas lisas, una rueda loca, un destornillador y una pista para seguimiento de línea.

Figura 22. Kit de Robótica Makeblock mBot Neo [28].

3.1.1.2 Sensores

Los sensores en los robots desempeñan un papel fundamental al permitirles percibir y comprender su entorno, de manera similar a cómo los sentidos en los seres vivos les proporcionan información sobre el mundo que les rodea.

Un sensor es un dispositivo que detecta una magnitud física, como la temperatura, la luz, el sonido, la proximidad o cualquier otra variable, y luego convierte esta información en una señal eléctrica que puede ser procesada y utilizada por el robot. Esta señal eléctrica puede ser empleada directamente en el control del robot o pasar por etapas previas de acondicionamiento, como amplificación o filtrado, para garantizar que la información sea precisa y adecuada para la toma de decisiones y la interacción con el entorno.

Sensor Ultrasónico

El sensor ultrasónico en el mBot Neo (ver figura 23) es un componente que se utiliza para medir la distancia entre el robot y los objetos que se encuentran frente a él.

Figura 23. Sensor Ultrasónico mBot Neo [28].

Como se observa en la figura 24 este sensor funciona emitiendo ondas ultrasónicas de alta frecuencia y luego detectando el tiempo que tarda en recibir el eco de estas ondas después de rebotar en un objeto cercano. A partir de este tiempo de viaje, el sensor puede calcular la distancia entre el robot y el objeto.

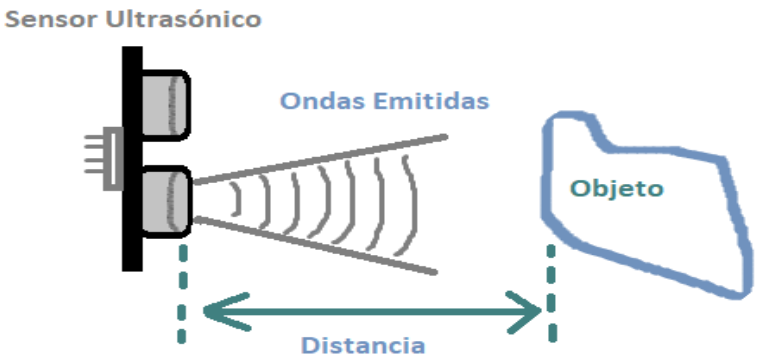


Figura 24. Sensor Ultrasónico. Elaborado por: Julissa Choez y Rodrigo Echaiz

El sensor ultrasónico es muy útil en aplicaciones de robótica porque permite al mBot Neo evitar obstáculos o interactuar de manera efectiva con su entorno; por ejemplo, puede ser empleado en las siguientes aplicaciones:

- **Detección de obstáculos:** El robot puede utilizar el sensor ultrasónico para detectar objetos o paredes en su camino y tomar decisiones para evitar colisiones.

- **Seguimiento de líneas:** En aplicaciones de seguimiento de líneas, el sensor ultrasónico puede usarse para mantener una distancia constante entre el robot y una línea en el suelo.

- **Seguimiento de líneas:** En aplicaciones de seguimiento de líneas, el sensor ultrasónico puede usarse para mantener una distancia constante entre el robot y una línea en el suelo.

- **Mapeo de entorno:** Puede ayudar al robot a mapear su entorno midiendo distancias a diferentes puntos y creando un mapa de obstáculos.
- **Estacionamiento autónomo:** En aplicaciones de estacionamiento de robots, el sensor ultrasónico puede utilizarse para medir la distancia entre el robot y una pared, lo que permite un estacionamiento preciso.

Sensor Giroscópico

El sensor giroscópico se encuentra localizado dentro del módulo CyberPi del robot y es un componente que se utiliza para medir la velocidad angular o la tasa de rotación del robot en torno a un eje específico. En otras palabras, detecta cuánto y en qué dirección está girando el robot.

Este sensor giroscópico es útil en diversas aplicaciones en robótica, especialmente en el control de movimiento y orientación del robot. Algunas de las funciones y aplicaciones que puede realizar el sensor giroscópico en el mBot Neo incluyen:

- **Estabilización y equilibrio:** El sensor giroscópico ayuda al robot a mantener su equilibrio y estabilidad al detectar cambios en su posición o inclinación y realizar ajustes para mantenerse erguido.

39

- **Seguimiento de orientación:** Puede utilizarse para determinar la orientación del robot en relación con un punto de referencia, lo que es útil en aplicaciones de navegación y seguimiento de rutas.
- **Control de movimiento:** El sensor giroscópico permite al robot medir la velocidad de rotación y ajustar su movimiento en función de esta información, lo que es esencial para maniobras precisas y movimientos controlados.
- **Detección de giros bruscos:** Puede detectar giros bruscos o cambios repentinos en la dirección del robot, lo que puede ser útil en aplicaciones de evasión de obstáculos.

Sensor Cuádruple RGB

El sensor cuádruple RGB en el mBot Neo como se muestra en la figura 25, es un componente que combina la capacidad de detección de colores y luz ambiental. Este sensor tiene cuatro canales RGB (rojo, verde, azul) y uno adicional para la luz ambiental. Su función principal es detectar y medir colores y niveles de luz en su entorno.

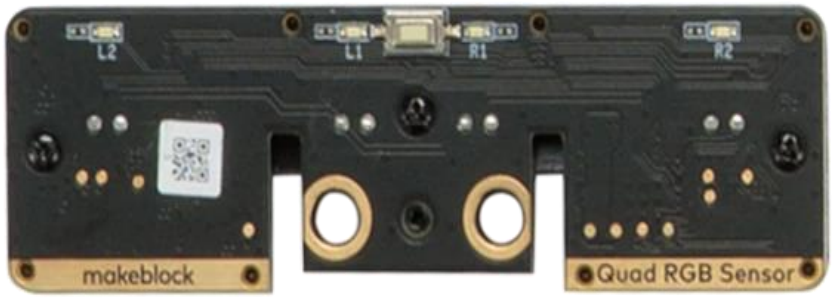


Figura 25. Sensor cuádruple RGB [28].

3.1.1.3 Actuadores

Los actuadores son componentes fundamentales que se incorporan a la estructura del robot para posibilitar y controlar sus movimientos. Gracias a estos elementos, el robot puede llevar a cabo diversas acciones, como desplazarse, girar, evitar obstáculos y más.

En el kit del mBot Neo, se incluyen dos motores DC que pueden ser integrados en el mecanismo del robot para darle la capacidad de movimiento. Estos motores son esenciales

para la funcionalidad del robot, permitiéndole ejecutar una amplia gama de acciones y tareas, desde desplazarse por su entorno hasta evitar obstáculos de manera autónoma.

Motor codificador

El Motor Codificador como se muestra en la figura 26, está diseñado con un codificador óptico que permite un control altamente preciso, con dos orificios roscados M4 en cada uno de sus tres lados, es fácil de conectar a las piezas mecánicas de Makeblock, lo que permite su versátil utilización en combinación con otros componentes. Además, utiliza materiales personalizados óptico para el rendimiento eléctrico los cuales reducen el nivel de ruido y garantizan un alto par de salida estas características la podemos observar en la Tabla 1 del rendimiento del motor codificador óptico. Este motor es compatible con múltiples controladores de motor y placas de control principales, como Orion, MegaPi, MegaPi Pro, Me Auriga y mBot2 Shield, lo que lo hace adecuado para una amplia gama de aplicaciones en robótica y proyectos de automatización.

Figura 26. Motor Codificador [29].

Rendimiento Eléctrico

Tensión de conducción	5V
Rango de velocidad de rotación	1–207 (RPM)
Precisión rotacional	≤ 5°
Precisión de detección	1°
Par rotacional	1500 g·cm
Material del eje de salida	Metal

Tabla 1. Características del rendimiento del motor codificador [29].

Microprocesador

Los robots Makeblock, como el mBot, generalmente están equipados con microcontroladores que actúan como el "microprocesador" del robot. Estos microcontroladores permiten que el robot procese información y ejecute programas para realizar diversas tareas y movimientos.

CyberPi

CyberPi es un versátil tablero de control principal desarrollado de forma independiente por Makeblock, como se muestra en la figura 27 tiene un diseño compacto y una variedad de puertos integrados, este dispositivo se presta fácilmente a la expansión y personalización. CyberPi es compatible con las plataformas de programación mBlock5 y mBlock-Python Editor, lo que lo hace ideal para una amplia gama de escenarios educativos. Este dispositivo cubre una amplia variedad de campos de enseñanza, que incluyen programación, creación y robótica, capaz de satisfacer diversas necesidades educativas, desde la programación de inteligencia artificial (IA) hasta el Internet de las cosas (IoT), la ciencia de datos y el diseño de interfaces de usuario. Las especificaciones técnicas de este dispositivo la podemos observar en la Tabla 2 donde nos muestra la capacidad que tiene cada uno de sus componentes integrados.

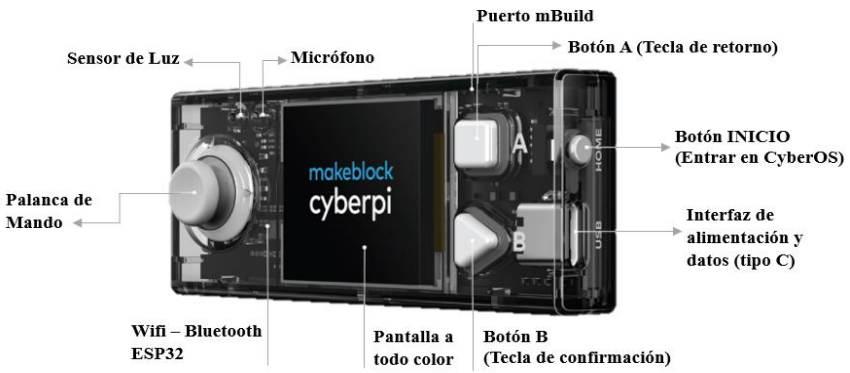


Figura 27. Componentes del Cyber [30].

Especificaciones Técnicas

Nombre		**CyberPi**
Chip		ESP32-WROVER-B
Procesador	Procesador principal	Xtensa® LX6 de 32 bits de doble núcleo
	Frecuencia de reloj	240MHz
Memoria integrada	ROM	448KB
	SRAM	520KB
Memoria extendida	Flash SPI	8 megas
	PSRAM	8 megas
Sistema operativo		CyberOS, desarrollado independientemente por Makeblock
Comunicación inalámbrica		Wifi, Bluetooth de modo dual
Puertos físicos		Puerto micro USB (Tipo C), Puerto para conectar placas de extensión, Puerto para conexión de módulos electrónicos (comunicación serie)
Versión del hardware		V1.0
Dimensiones		84 mm × 35 mm × 13 mm (alto × ancho × profundidad)
Peso		36 gramos

Tabla 2. Especificaciones técnicas del CyberPi [30].

3.1.2 Componentes lógicos

A continuación, se enumeran los componentes lógicos que se han utilizado para programar los componentes físicos de esta propuesta de investigación. Estos componentes son fundamentales para el desarrollo y funcionamiento adecuado de la implementación, ya que permiten la coordinación y control de las actividades llevadas a cabo por los elementos físicos.

3.1.2.1 Lenguajes de Programación en el Entorno de mBlock

mBlock es una plataforma de programación diseñada para ser accesible y educativa, especialmente para niños y principiantes en el mundo de la programación. Desarrollado

por Makeblock, mBlock se basa en Scratch, un popular lenguaje de programación visual, y está orientado a la educación STEAM (Ciencia, Tecnología, Ingeniería, Arte y Matemáticas).

3.1.2.2 Lenguaje Visual Scratch

mBlock utiliza un lenguaje de programación visual basado en bloques, similar a Scratch. En este entorno, los usuarios pueden arrastrar y soltar bloques de código para crear programas, lo que facilita el aprendizaje de conceptos de programación sin la necesidad de escribir código complejo. Esta metodología es especialmente útil para enseñar lógica de programación y pensamiento computacional, scratch es un lenguaje ampliamente utilizado en educación y creado por el Grupo Lifelong Kindergarten del MIT Media Lab.

Una de las características de scratch es que los bloques de código están codificados por colores y representan diferentes tipos de comandos, como movimiento, control, sensores y operadores. Esta representación visual ayuda a los usuarios a comprender rápidamente la lógica detrás de sus programas. En la figura 28 se muestran las diferentes áreas de trabajo y configuración de la pantalla principal del software mBlock.

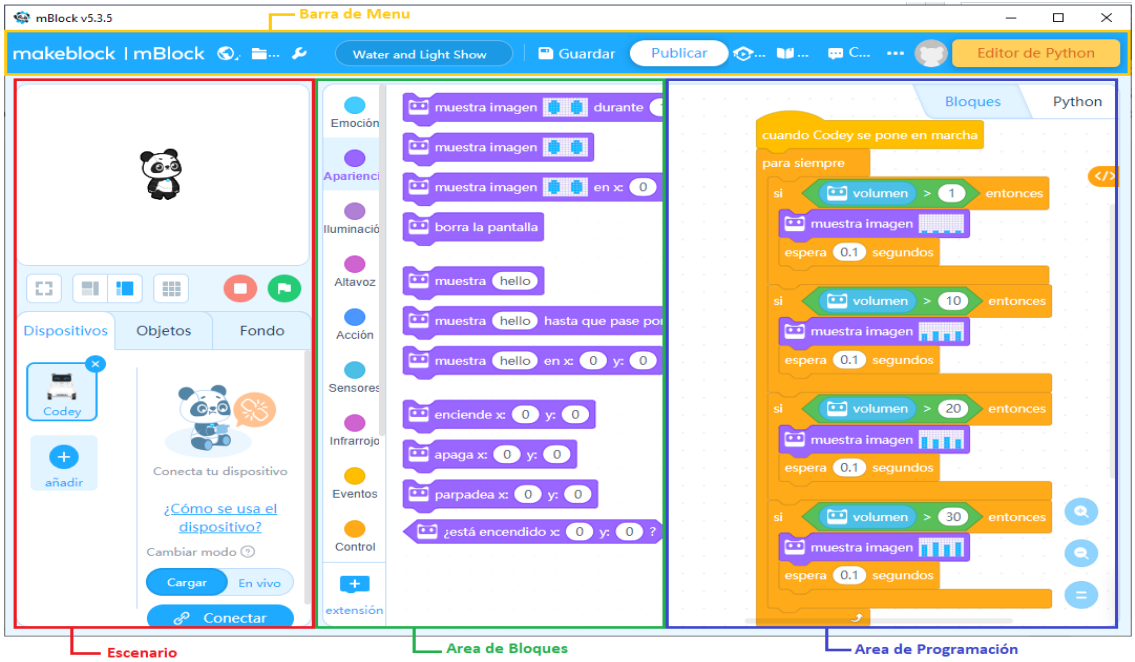


Figura 28. Lenguaje de programación visual mBlock.

3.1.2.3 Python

Python es un lenguaje de programación de alto nivel conocido por su sintaxis simple y legible, lo que lo convierte en una excelente opción tanto para principiantes como para programadores experimentados. En el entorno de mBlock como se observa en la figura 29, además del lenguaje de bloques, también se soporta Python. Este lenguaje de programación textual es ampliamente utilizado en la educación y la industria. La sintaxis clara y sencilla de Python facilita a los estudiantes la transición desde la programación basada en bloques a un lenguaje textual. En mBlock, los usuarios pueden ver el código Python correspondiente a sus programas de bloques, lo que ayuda a comprender mejor la lógica detrás del código y facilita el aprendizaje progresivo de ambos paradigmas de programación.

Esta integración permite a los usuarios desarrollar habilidades avanzadas de programación y aplicar sus conocimientos en una variedad de contextos, preparando a los estudiantes para futuros estudios y carreras en tecnología y ciencia.

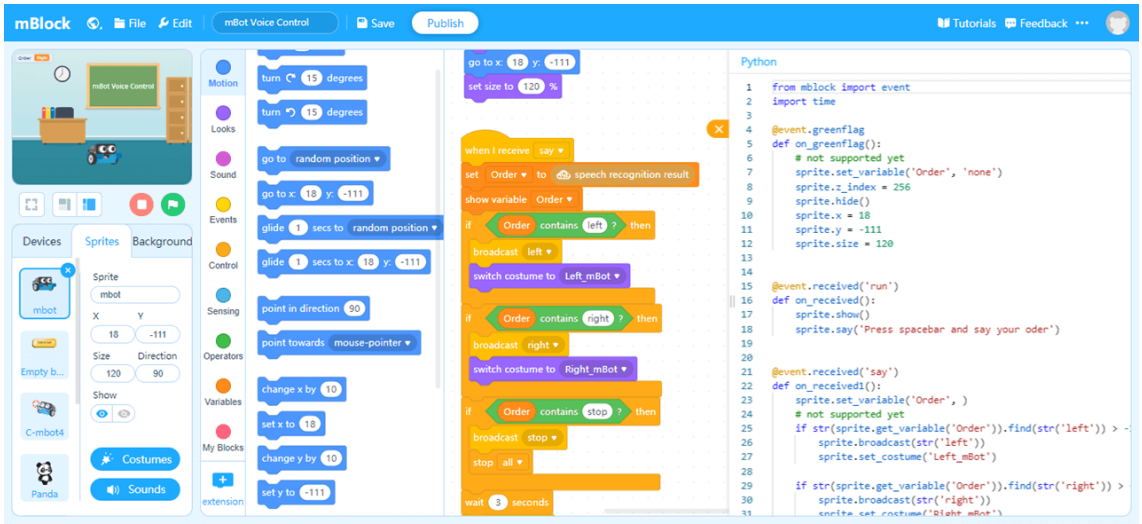


Figura 29. Lenguaje de programación visual mBlock con Python [31].

3.1.2.3 MATLAB

MATLAB es una potente herramienta de programación y análisis numérico ampliamente utilizada en la investigación, la ingeniería y la ciencia. Su capacidad para realizar cálculos

complejos, visualizar datos y su amplia variedad de aplicaciones lo convierten en una herramienta esencial en numerosos campos técnicos y científicos, utiliza un lenguaje de programación de alto nivel que permite a los usuarios escribir programas y funciones con una sintaxis clara y concisa. Esto facilita el desarrollo rápido de algoritmos y la realización de cálculos avanzados, una de las fortalezas de MATLAB es su capacidad para crear gráficos y visualizaciones de datos. Los usuarios pueden generar desde gráficos simples hasta visualizaciones tridimensionales complejas, lo que ayuda a interpretar y comunicar los resultados de los análisis de manera efectiva, también está optimizado para realizar cálculos numéricos intensivos, lo que lo hace ideal para tareas como la resolución de ecuaciones diferenciales, el análisis de series temporales y la optimización.

MATLAB se integra fácilmente con otros lenguajes de programación como C, C++, Java y Python. Esto permite a los usuarios combinar la potencia de MATLAB con otras herramientas y bibliotecas, aumentando la flexibilidad y el alcance de sus aplicaciones, es utilizado en diversas ramas de la ingeniería, como la ingeniería eléctrica, mecánica, civil y aeroespacial. Se emplea para el diseño y análisis de sistemas, el procesamiento de señales, la dinámica de sistemas y el control automático.

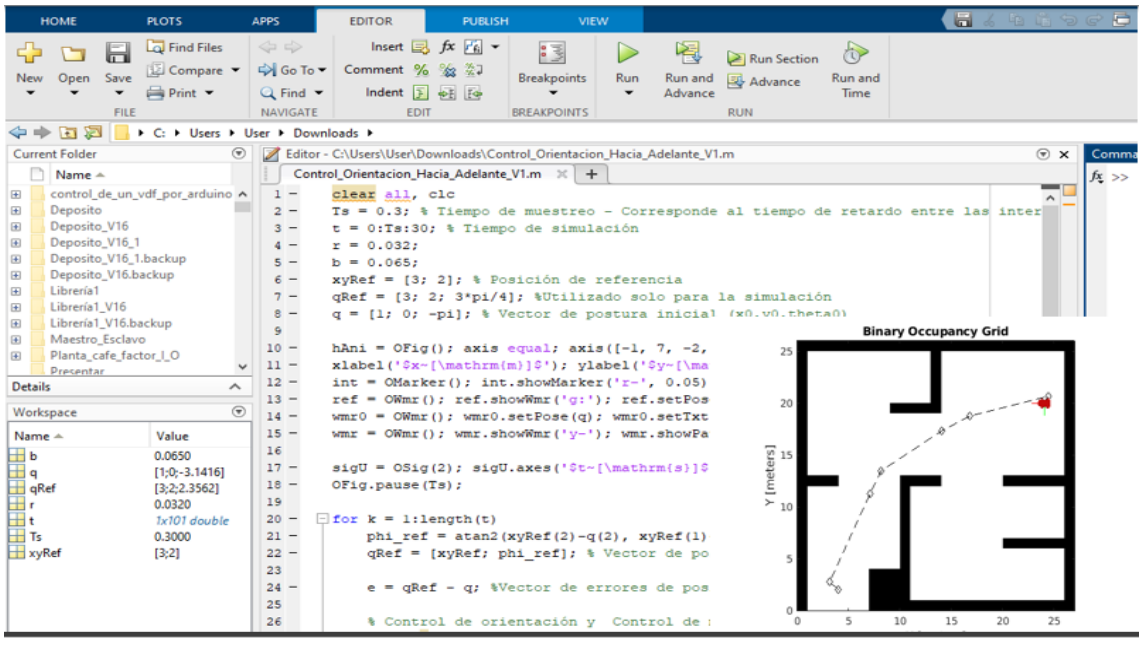


Figura 30. Herramienta de programación MATLAB.

3.2 IMPLEMENTACIÓN DE LOS ALGORITMOS TIPO INSECTO

Los algoritmos tipo insecto, inspirados en los comportamientos colectivos de insectos, utilizan modelos matemáticos para simular estos comportamientos y resolver problemas complejos de optimización. A continuación, se presenta los tres algoritmos tipo insecto que utilizan en esta propuesta, acompañada de sus formulaciones matemáticas.

3.2.1 Algoritmo insecto 0

El algoritmo insecto 0 implica varias operaciones matemáticas fundamentales para la navegación del robot desde un punto inicial a un punto final, evitando obstáculos, el cual funciona de la siguiente manera:

- **Movimiento hacia el Objetivo**: El robot se mueve en línea recta hacia el objetivo hasta que detecta un obstáculo.
- **Detección de Obstáculos**: Al encontrar un obstáculo, el robot cambia su comportamiento para seguir el contorno del obstáculo.
- **Seguimiento del Contorno del Obstáculo**: El robot sigue el contorno del obstáculo hasta que puede reanudar el movimiento hacia el objetivo.

Variables

x_{goal}: Coordenada x del objetivo

y_{goal}: Coordenada y del objetivo

x_{robot}: Coordenada x del robot

y_{robot}: Coordenada y del robot

θ_{goal}: Ángulo al objetivo calculado usando la función arctangente de las diferencias de coordenadas y y x.

$\theta_{referncia}$: Ángulo de referencia.

θ_{error}: Error de ángulo calculado como la diferencia entre el ángulo de referencia y el ángulo actual.

v: Velocidad lineal del robot.

v_x: Componente de la velocidad en el eje x

v_y: Componente de la velocidad en el eje y.

ω_d: Velocidad angular de la rueda derecha.

ω_i: Velocidad angular de la rueda izquierda.

r: Radio de las ruedas del robot.

L: Distancia entre las ruedas del robot.

$\omega_{d(\mathrm{RPM})}$: Velocidad angular de la rueda derecha en revoluciones por minuto (RPM).

$\omega_{i(\mathrm{RPM})}$: Velocidad angular de la rueda izquierda en revoluciones por minuto (RPM).

Para encontrar las velocidades, obtener los valores de ω_d y ω_i para posteriormente utilizarlo en la programación del mblock se utilizan las siguientes ecuaciones.

Ángulo al objetivo

$$[\theta_{goal} = \mathrm{atan2}(y_{goal} - y_{robot}, x_{goal} - x_{robot})]$$

Velocidades lineales y angulares.

$$v_x = v\cos(\theta) \qquad v_y = v\sin(\theta)$$

$$\omega_d = \frac{v}{r} + \frac{L\omega}{2r} \qquad \omega_i = \frac{v}{r} - \frac{L\omega}{2r}$$

Conversión a RPM

$$\omega_{d(\text{RPM})} = \frac{\omega_d \times 60}{2\pi} \qquad \omega_{i(\text{RPM})} = \frac{\omega_i \times 60}{2\pi}$$

Error de ángulo con respecto al obstáculo.

$$\theta_{\text{error}} = \theta_{\text{referncia}} - \theta_{\text{error}}$$

Estas mismas ecuaciones se utilizan para el control del mBot en los demás algoritmos.

Flujograma algoritmo insecto 0

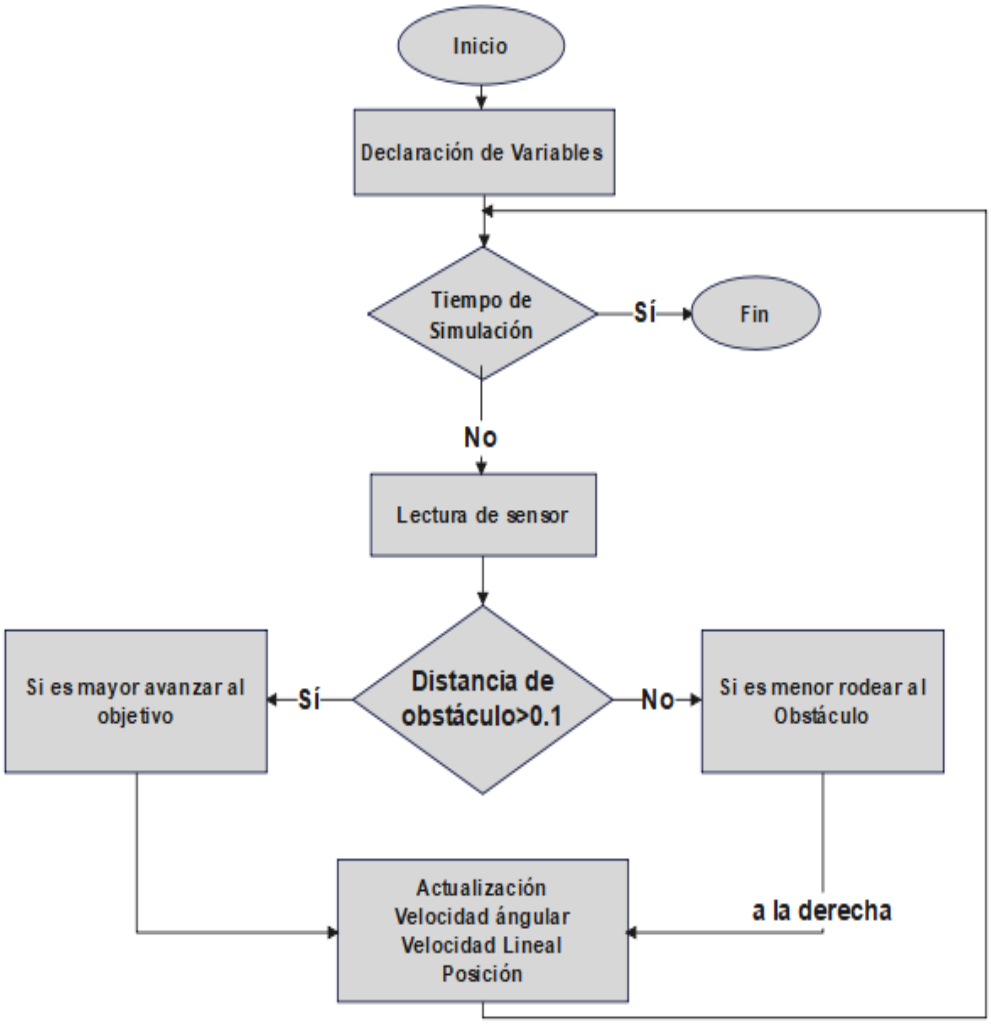


Figura 31. Flujograma tipo insecto 0

El flujo de diagrama en la figura 31, describe el algoritmo insecto 0 que controla un robot móvil para moverse hacia un objetivo mientras evita obstáculos. El robot inicialmente se

mueve hacia el objetivo, si detecta un obstáculo, calcula un nuevo ángulo de referencia para rodear el obstáculo, ajusta sus velocidades lineales y angulares basándose en la distancia y el ángulo al objetivo, y continua este proceso hasta que alcanza el objetivo o el tiempo límite se agota.

3.2.2 Algoritmo insecto 1

Para el algoritmo insecto 1 se utilizan las mismas funciones mencionadas en el insecto 0, tales como el cálculo de las posiciones iniciales y finales, la determinación de la distancia, el ángulo, la detección de obstáculos y el seguimiento del contorno. Estas funciones son esenciales no solo para determinar si el robot ha llegado al objetivo, sino también para manejar la detección y evitación de obstáculos durante la navegación.

El algoritmo insecto 1 extiende las capacidades del insecto 0 al introducir la capacidad de medir y recordar la distancia más cercana al objetivo durante la circunnavegación de un obstáculo. Esto se realiza midiendo la distancia euclidiana desde varios puntos del contorno del obstáculo hasta el objetivo y recordando el punto más cercano.

$$qm = d_{\text{obst}} = \left(\sqrt{\left(x_{\text{goal}} - x_{\text{actual}}\right)^2 + \left(y_{\text{goal}} - y_{\text{actual}}\right)^2}\right)$$

Cuando el robot vuelve al punto donde inicialmente encontró el obstáculo, sigue la dirección que lo lleva al punto más cercano previamente registrado. Este enfoque permite que el robot navegue de manera más eficiente alrededor de obstáculos grandes o complejos, asegurando que siempre se mueve hacia el objetivo más cercano posible.

Además, el algoritmo insecto 1 utiliza un sistema de control proporcional para ajustar las velocidades lineales y angulares del robot en función del error angular y la distancia al objetivo. Esto garantiza un movimiento suave y preciso, mejorando la capacidad del robot para navegar en entornos dinámicos y complejos.

Flujograma algoritmo insecto 1

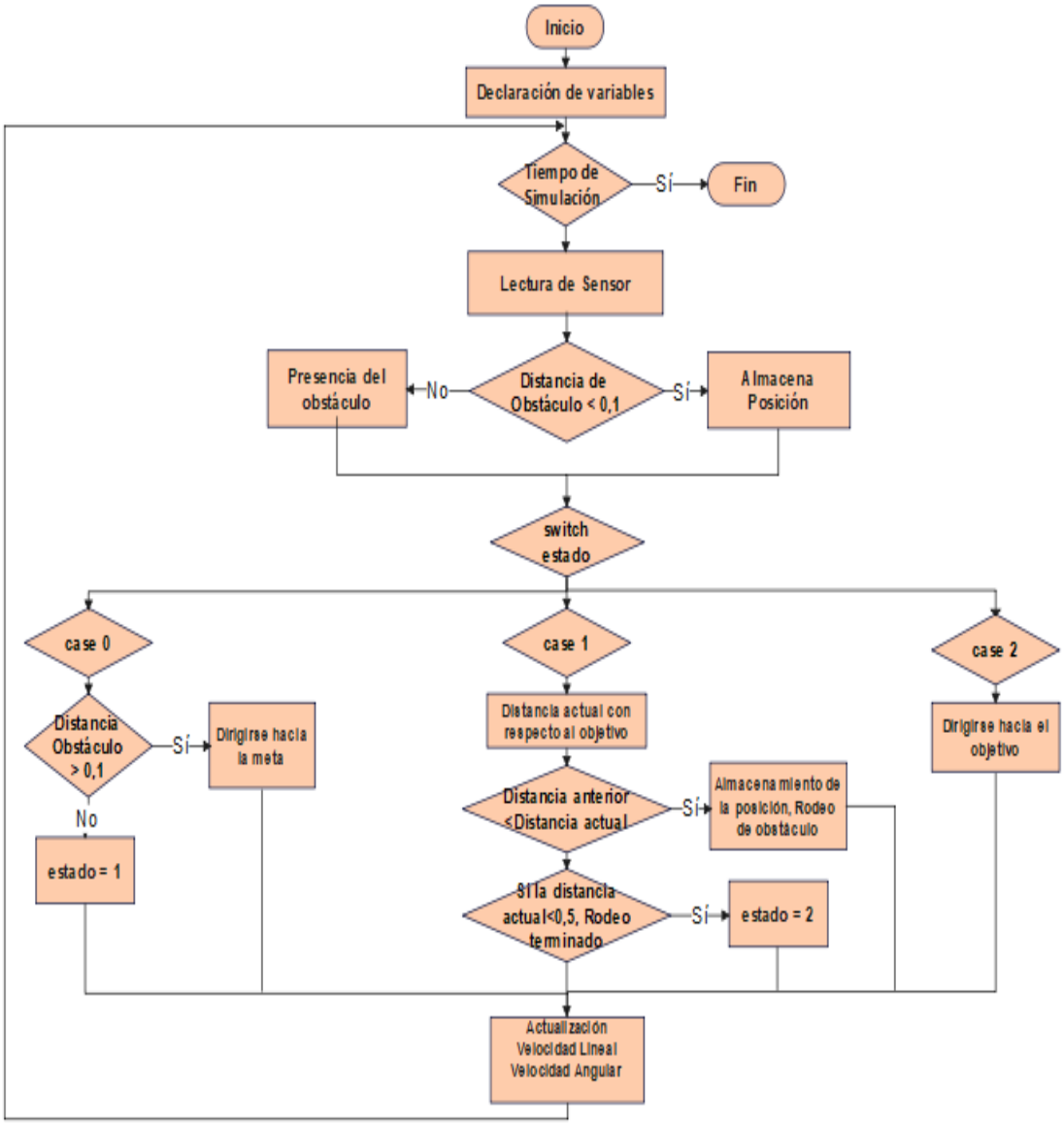


Figura 32. Flujograma tipo insecto 1.

El algoritmo insecto 1 se inicia con los datos de salida y llegada proporcionados. Este algoritmo se distingue por su capacidad para rodear completamente un obstáculo al detectarlo, y luego, basándose en los datos capturados, determina la trayectoria más cercana para seguir. Para lograr esto, utiliza las mismas funciones que el algoritmo insecto 0, como atan para calcular el ángulo de la trayectoria principal, y las funciones rodeo para rodear el obstáculo y seguir la trayectoria más cercana respectivamente.

Cuando el algoritmo detecta un obstáculo, rodea completamente dicho obstáculo y luego regresa al punto inicial. Posteriormente, utiliza la función qm para encontrar la trayectoria más cercana para continuar sin el obstáculo. Este proceso se repite para cada obstáculo detectado en el camino hasta que el robot alcance su objetivo. Este enfoque permite al robot navegar de manera efectiva alrededor de los obstáculos en su camino hacia la meta, optimizando así su ruta y evitando colisiones.

3.2.3 Algoritmo insecto 2

Para el algoritmo insecto 2, se utilizan funciones similares a las mencionadas en insecto 0 e insecto 1, este algoritmo mejora las capacidades del insecto 1 al considerar tanto la distancia directa desde el inicio hasta el objetivo como la distancia desde la posición actual al objetivo durante la circunnavegación de un obstáculo. Esto permite decidir el momento óptimo para abandonar la circunnavegación y reanudar el movimiento directo hacia el objetivo. El algoritmo insecto 2 evalúa continuamente si la línea directa hacia el objetivo está libre de obstáculos. Cuando se encuentra un punto despejado, el robot interrumpe la circunnavegación y se dirige directamente hacia el objetivo, minimizando el tiempo y la distancia recorrida. Además, el insecto 2 utiliza un control más refinado de las velocidades lineales y angulares del robot, permitiendo una navegación más eficiente en entornos dinámicos y complejos, mejorando la precisión en la llegada al objetivo y reduciendo el tiempo total de navegación.

En el algoritmo insecto 2 se utilizan dos fórmulas para determinar distancias clave en la planificación de rutas.

$$d_{punto\_línea} = \frac{|(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

La primera fórmula, distancia_punto_linea, calcula la distancia entre un punto (x_0, y_0) y una línea definida por dos puntos (x_1, y_1) y (x_2, y_2). Esta distancia es para determinar si el robot se encuentra dentro de un obstáculo o si necesita desviarse para evitarlo.

$$d_{Linea\_m} = \frac{|(G_2 - S_1)(S_Y - X_Y) - (S_X - X_X)(G_Y - S_Y)|}{\sqrt{(G_X - S_X)^2 + (G_Y - S_Y)^2}}$$

La segunda fórmula, **distancia_linea_meta**, calcula la distancia entre un punto (S_X, S_Y) que representa el inicio de la línea de llegada y un punto (G_X, G_Y) que representa el final de la línea de llegada. Además, se utiliza un punto intermedio (X_Y) en la coordenada Y de la línea de llegada para calcular esta distancia. Esta distancia es importante para determinar cuándo el robot ha llegado a la línea de llegada y puede continuar hacia su objetivo final.

Flujograma algoritmo insecto 2

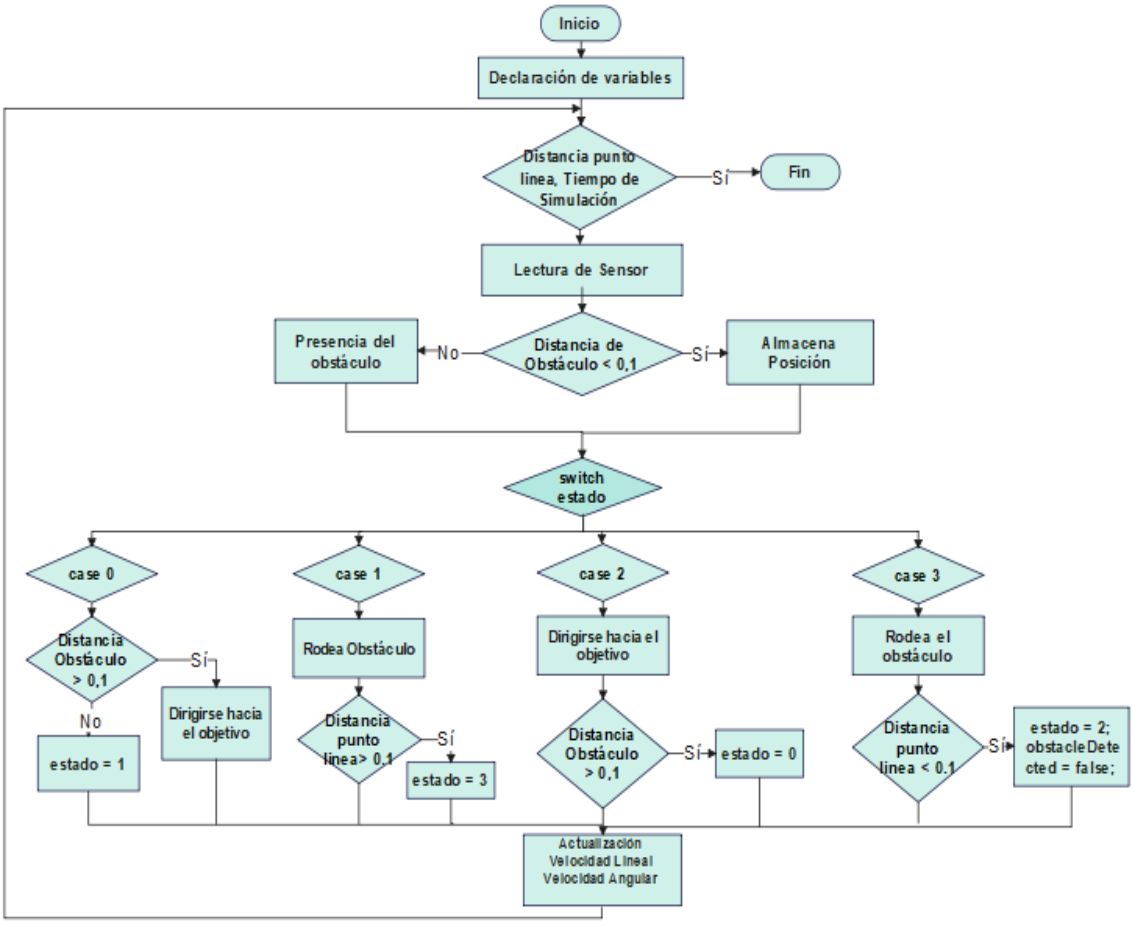


Figura 33. Flujograma tipo insecto 2.

En la figura 33, se presenta el algoritmo utilizado para implementar el tipo de movimiento denominado 'insecto 2'. El algoritmo insecto 2 permite que el robot navegue hacia un objetivo en presencia de obstáculos. La estrategia consiste en moverse directamente hacia el objetivo y circunnavegar los obstáculos cuando se detectan. El robot mide continuamente la distancia a la línea objetivo y la distancia al obstáculo, ajustando su trayectoria según sea necesario. El algoritmo cambia entre diferentes estados (estado 0, estado 1, estado 2, estado 3) para gestionar el movimiento directo y la circunnavegación, optimizando el tiempo y la distancia recorrida alrededor de los obstáculos.

3.3 DESARROLLO DE LA PROPUESTA

La implementación de esta propuesta comienza con la configuración inicial del robot mBot Neo, donde se seleccionan los elementos necesarios para la ejecución de los algoritmos tipo insecto. Este proceso implica la identificación y disposición de los motores suministrados en el paquete del robot, junto con la integración de dos sensores y un adaptador Bluetooth adicionales para mejorar su capacidad de percepción y comunicación. Una vez establecidos estos componentes, se procede a diseñar la estructura de las pistas, proporcionando al robot móvil un entorno adecuado para desarrollar sus trayectorias y llevar a cabo el análisis comparativo de los algoritmos.

Además, se desarrolla un entorno de simulación en MATLAB para demostrar el funcionamiento de los diferentes algoritmos. Esta simulación se configura como un entorno conocido para el robot, donde los obstáculos se agregan previamente al escenario. En contraste, las pruebas reales se realizan en un entorno físico, donde el robot se enfrenta a situaciones reales con obstáculos de diferentes tamaños y disposiciones, lo que permite una validación más precisa de los algoritmos.

Posteriormente, se lleva a cabo un análisis comparativo entre la simulación realizada en MATLAB y las pruebas reales del sistema. El objetivo principal es evaluar el rendimiento

de los tres algoritmos tipo insecto en una variedad de situaciones. Se consideran diferentes casos en el sistema, incluyendo la presencia de obstáculos con diversas características.

3.3.1 Diseño del sistema

En el diseño del sistema, se priorizó la ubicación estratégica de los sensores para optimizar el seguimiento y la trayectoria del robot móvil. Se colocó el sensor uno en la parte frontal para detectar obstáculos y realizar ajustes anticipados en la ruta, otro en la parte diagonal izquierda del robot, mientras que el sensor tres se posicionó en la parte trasera del lado izquierdo para brindar información adicional sobre posibles obstáculos en esa dirección. Este enfoque permite una navegación más efectiva y segura del robot, además, la programación se realizó en mBlock, una plataforma intuitiva que facilita la creación de proyectos de robótica y programación para principiantes y avanzados. mBlock es compatible con el lenguaje de programación Python, lo que permite a los usuarios beneficiarse de la simplicidad de mBlock y la potencia de Python. Esta integración permite a los usuarios implementar algoritmos complejos y realizar ajustes finos en el comportamiento del robot. En la programación del robot utilizando Python en mBlock, es común trabajar con las velocidades angulares de las ruedas denotada de la siguiente forma:

wdRMP = (wd * 60) / (2 * pi)	wiRPM = (wi * 60) / (2 * pi)

Estas líneas convierten las velocidades angulares de las ruedas derecha (wd) e izquierda (wi) de radianes por segundo (rad/s) a revoluciones por minuto (RPM). La fórmula utilizada es:

$$\mathrm{RPM} = 60/2 * \pi$$

Donde 60 es el número de segundos por minuto y 2π es la cantidad de radianes en una revolución completa.

```
atan2_N_N((listaPosRef[1] − listaPos[1]), (listaPosRef[0] − listaPos[0]))
```

Esta línea llama a la función atan2_N_N para calcular el ángulo entre la posición actual del robot (listaPos) y la posición de referencia (listaPosRef). La función atan2 calcula el arco tangente del cociente de sus argumentos, teniendo en cuenta los signos de ambos para determinar el cuadrante correcto del ángulo. Esto es útil para obtener la dirección que debe seguir el robot para dirigirse hacia la posición de referencia.

```
sensor1 = cyberpi.ultrasonic2.get (1) / 100
sensor2 = cyberpi.ultrasonic2.get(2) / 100
sensor3 = cyberpi.ultrasonic2.get(3) / 100
```

Estas líneas leen las distancias medidas por tres sensores ultrasónicos conectados al robot CyberPi. Los sensores están identificados como 1, 2 y 3. Las distancias medidas se dividen por 100 para convertirlas de centímetros a metros. Estas medidas se utilizan para detectar obstáculos alrededor del robot.

```
mbot2.drive_speed(wiRPM, −1 ∗ wdRMP)
```

Esta línea establece las velocidades de las ruedas del mBot2. La rueda izquierda (wiRPM) se configura para girar a wiRPM revoluciones por minuto, mientras que la rueda derecha (wdRMP) se configura para girar a - wdRMP revoluciones por minuto (el negativo indica que la rueda derecha gira en sentido opuesto). Esto permite controlar la dirección y velocidad del robot para su navegación.

Preparación de trayectoria.

En la preparación de la trayectoria del robot, se consideran varios factores que afectan su dinámica y comportamiento. Entre ellos, se destacan las tres pistas que se implementarán: la pista para evaluar la velocidad y precisión del robot, la pista con obstáculos que desafiará su capacidad de evitar colisiones.

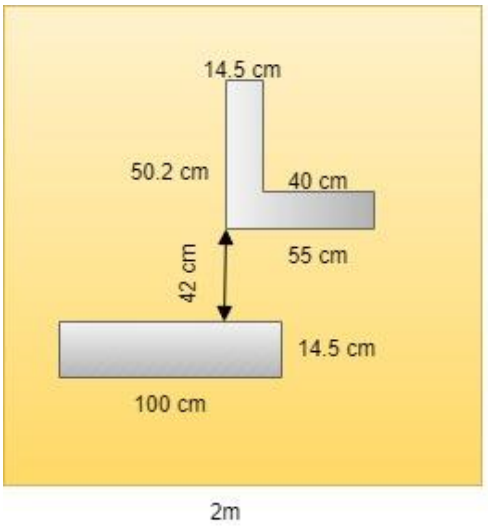


Figura 34. Pista 1. Fuente. Autor

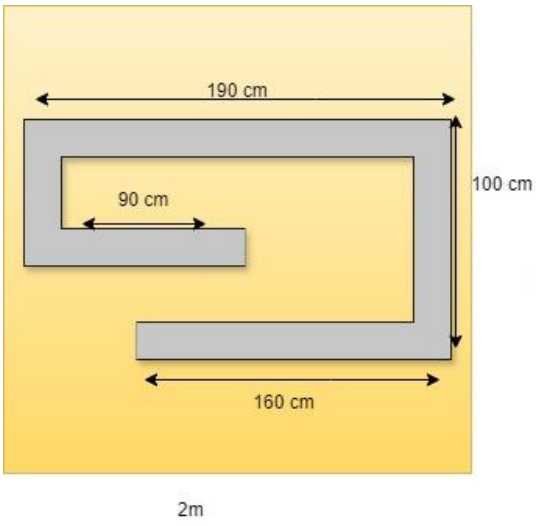


Figura 35. Pista 2.

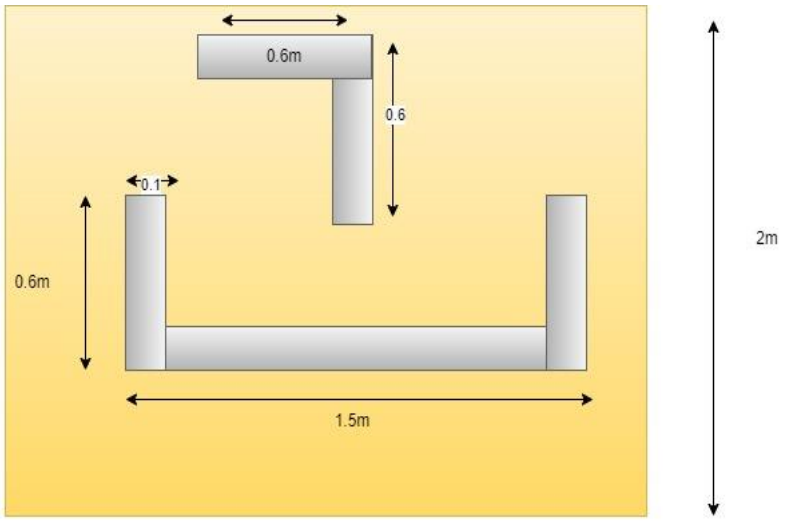


Figura 36. Pista 3.

Las figuras mostradas representan las pistas utilizadas para demostrar los tres algoritmos: Tipo insecto 0, Tipo insecto 1 y Tipo insecto 2.

Seguimiento de la frontera

El comportamiento principal de los algoritmos es seguir la frontera de los obstáculos, lo que implica una serie de movimientos específicos para llevar a cabo esta tarea.

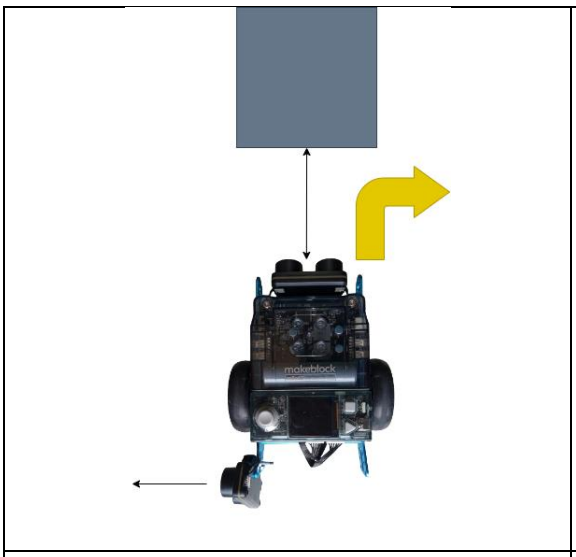


Figura 37. Movimiento 1.

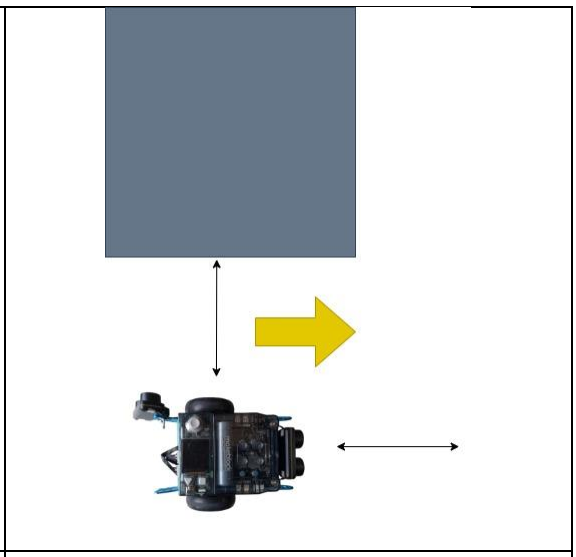


Figura 38. Movimiento 2.

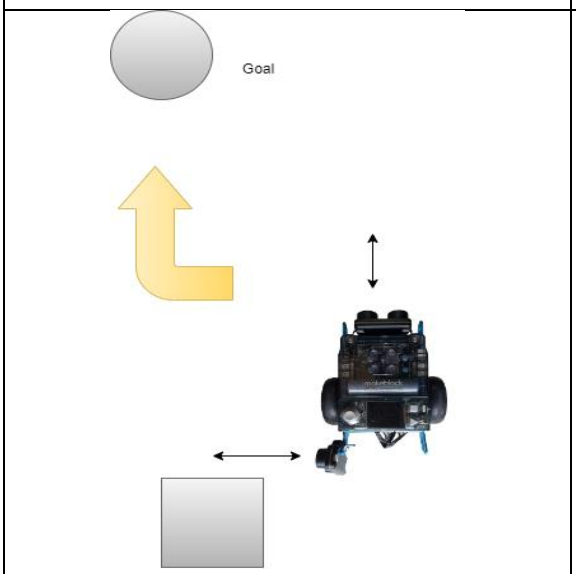


Figura 39. Movimiento 3.

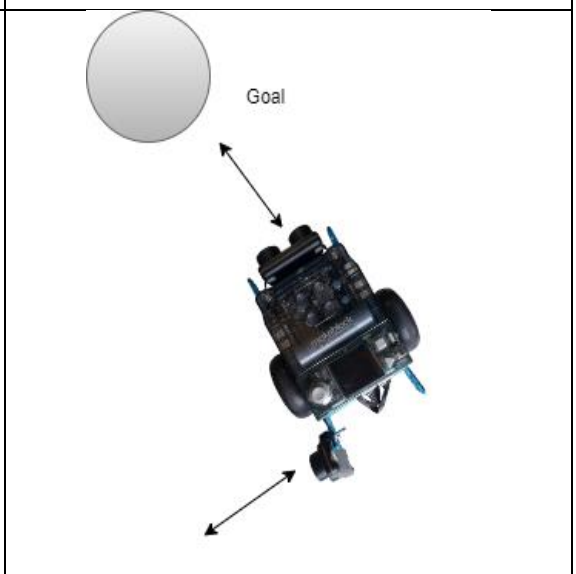


Figura 40. Movimiento 4.

En la figura 37, se representa el primer movimiento del algoritmo al detectar un obstáculo. Consiste en girar hacia la derecha y avanzar, con el objetivo de evitar una colisión directa y comenzar a seguir el contorno del obstáculo. La figura 38 muestra cómo avanza mientras detecta un obstáculo con un sensor. En la figura 39 el movimiento representa la

búsqueda de la trayectoria principal, y la figura 40 muestra el movimiento en línea recta siguiendo la trayectoria deseada.

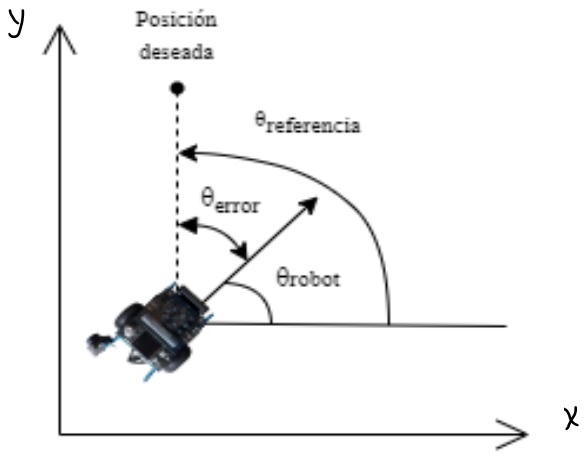


Figura 41. Ángulo del mBot.

En la figura 41 se muestra el ángulo del robot, la referencia y la dirección de la posición deseada. Para calcular el error, se utiliza la siguiente fórmula:

$$\theta_{referencia} = arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

Donde (x_1, x_2) es la posición actual del robot y (y_1, y_2)) es la posición de llegada en las coordenadas x y y, respectivamente. En el lenguaje de programación, comúnmente se utiliza la función "arctan2" para determinar el ángulo en los 4 cuadrantes, ya que la función "arctan" solo devuelve resultados entre -90 y 90 grados. Sin embargo, MBlock no admite directamente la función "arctan2", por lo que es necesario implementar un algoritmo adicional para determinar el cuadrante en el que está el robot.

Para determinar el cuadrante, se puede utilizar la siguiente lógica:

- Si ambos diferenciales (x y y) son positivos, el ángulo está en el primer cuadrante.
- Si x es negativo y y es positivo, el ángulo está en el segundo cuadrante.
- Si ambos diferenciales son negativos, el ángulo está en el tercer cuadrante.

- Si x es positivo y y es negativo, el ángulo está en el cuarto cuadrante.

3.3.2 Implementación de los algoritmos.

Implementación algoritmo insecto 0

La implementación del algoritmo Insecto 0 se lleva a cabo mediante un código que emplea un tiempo de muestreo de 0.03 segundos. Este código se encarga de la navegación de un robot desde un punto de partida, representado por "q", hasta un objetivo, denominado "goal". El código completo se encuentra en los ANEXOS A, tanto en MATLAB como en Python utilizando mBlock.

El código tiene una lógica principal basada en la detección de obstáculos. Si la distancia al obstáculo (dObst) es mayor que el umbral definido, que representa la distancia entre el mBot y el obstáculo el robot realizará un giro rigurosamente a la derecha para evitar la colisión. En la simulación, se crean funciones adicionales como nearestSegment y closestSegment para calcular la proximidad y la dirección del obstáculo, respectivamente.

En MATLAB, la variable dObst representa la distancia medida frente al obstáculo. La simulación muestra cómo el robot navega hacia su objetivo mientras evita obstáculos, ajustando su trayectoria en función de la información proporcionada por los obstáculos generados mediante la simulación.

Para el cálculo de las distancias y velocidades de las ruedas, se utilizan las siguientes fórmulas: la velocidad lineal v se descompone en componentes vx y vy, y las velocidades angulares de las ruedas derecha e izquierda se calculan como wd y wi, respectivamente, como se muestra en la figura 42. Estas velocidades se convierten a RPM para controlar el movimiento del robot.

En Python, mediante mBlock, se sigue una lógica similar. El robot utiliza tres sensores ultrasónicos para detectar obstáculos y ajustar su dirección en consecuencia. La lógica del código Python también incluye un filtro de media móvil para suavizar las lecturas de los sensores y evitar valores falsos. El código completo se encuentra en los anexos, proporcionando una implementación del algoritmo insecto 0 en ambas plataformas.

```
if dObst > distanciaMinima
    phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
    ePhi = wrapToPi(phiRef - q(3));
    dGoal = sqrt(sum((goal - q(1:2)).^2));
    g = [Vs, 1];
    obstacleAvoided = false;
elseif dObst <= distanciaMinima
    ePhi = wrapToPi(pi + phiObst - q(3));
    dGoal = sqrt(sum((goal - q(1:2)).^2));
    g = [Vs, 10];
end

v = g(1) * abs(cos(ePhi));
w = g(2) * ePhi;
v = min([v, 0.3]);

vx = v * cos(q(3));
vy = v * sin(q(3));
wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w);
wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w);
wdRMP = wd * 60 / (2 * pi);
wiRMP = wi * 60 / (2 * pi);
```

Figura 42. Código de programación en MATLAB algoritmo tipo insecto 0.

Implementación algoritmo insecto 1

La implementación del algoritmo insecto 1 se realiza con un tiempo de muestreo de 0.03 segundos. Este algoritmo gestiona la navegación de un robot desde una posición inicial" q" hasta un objetivo final, denominado" goal". El código completo se encuentra en los ANEXOS A, tanto en MATLAB como en Python utilizando mBlock.

El algoritmo insecto 1 permite que el robot navegue alrededor de los obstáculos mientras se compara constantemente la distancia más corta hacia el objetivo. Se utilizan funciones adicionales como nearestSegment para determinar la distancia al obstáculo más cercano

y wrapToPi para ajustar los ángulos dentro del rango$[-\pi, \pi]$. En la simulación el robot inicia su movimiento hacia el objetivo y al detectar un obstáculo, almacena la posición inicial del obstáculo y comienza a rodearlo.

En MATLAB como se muestra en la figura 43, la variable dObst representa la distancia medida al obstáculo más cercano. Si la distancia dObst es menor que el umbral, el robot guarda la posición inicial del obstáculo y empieza a rodearlo. Durante este rodeo, el robot compara constantemente su distancia al objetivo y almacena el punto más cercano al objetivo que encuentra mientras rodea el obstáculo. Una vez que el robot ha rodeado el obstáculo y se encuentra en el punto más cercano al objetivo, se dirige a ese punto y luego retoma su trayectoria hacia el objetivo.

En Python, utilizando mBlock, el algoritmo sigue una lógica similar. El robot detecta obstáculos utilizando tres sensores ultrasónicos y ajusta su dirección para rodearlos. Durante el rodeo, el robot calcula y almacena constantemente el punto más cercano al objetivo que encuentra. Al completar el rodeo, el robot se dirige al punto más cercano al objetivo y luego avanza hacia el objetivo. El código completo se encuentra en los anexos, proporcionando una implementación detallada del algoritmo insecto 1 en ambas plataformas.

Para el cálculo de las distancias y velocidades de las ruedas, se utilizan las siguientes fórmulas: la velocidad lineal v se descompone en componentes vx y vy y las velocidades angulares de las ruedas derecha e izquierda se calculan como wd y wi respectivamente. Estas velocidades se convierten a RPM para controlar el movimiento del robot. Las fórmulas son esenciales para garantizar un movimiento suave y preciso del robot.

```
if dObst > umbralObstaculo  % Conducir hacia la meta
       phiRef = atan2(qm(2)-q(2), qm(1)-q(1));
       ePhi = wrapToPi(phiRef - q(3));
       dGoal = sqrt(sum((qm-q(1:2)).^2));
       g = [Vs, 5]; % Ganancias de control
   else % Conducir alrededor del obstáculo
           phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
       ePhi = wrapToPi(phiRef-q(3));
       g = [Vs, 5]; % Ganancias de control
       f=sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2);
        if sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2) < 0.1
           estado = 5 % Cambiar al estado de conducción hacia la meta

           guardar=1;
           rodeo=0;

       end
```

Figura 43. Código de programación en MATLAB algoritmo tipo insecto 1

Implementación algoritmo insecto 2

La implementación del algoritmo insecto 2 se realiza con un tiempo de muestreo de 0.03 segundos. Este algoritmo controla la navegación de un robot desde una posición inicial "q" hasta un objetivo final, denominado "goal".

El algoritmo insecto 2 permite que el robot navegue alrededor de los obstáculos hasta que se alinee con la línea recta que conecta el punto de partida y el objetivo. Se utilizan funciones adicionales como nearestSegment para determinar la distancia al obstáculo más cercano y wrapToPi para ajustar los ángulos dentro del rango$[-\pi, \pi]$. En la simulación, el robot inicia su movimiento hacia el objetivo y, al detectar un obstáculo, rodea el obstáculo hasta encontrar una posición en la línea que conecta el punto de partida con el objetivo, momento en el que retoma su trayectoria hacia el objetivo.

Así mismo como el algoritmo tipo insecto 0 e insecto 1, en este algoritmo se utiliza la variable dObst. Si la distancia dObst es menor que el umbral, el robot almacena la

posición inicial del obstáculo y comienza a rodearlo. Una vez que el robot se alinea nuevamente con la línea hacia el objetivo, continua hacia el objetivo mientras evita cualquier obstáculo en su camino.

En Python, utilizando mBlock, el algoritmo sigue una lógica similar. El robot detecta obstáculos utilizando tres sensores ultrasónicos y ajusta su dirección para rodearlos. Utiliza la fórmula d_line_m, como se muestra en la figura 44 para calcular la distancia entre la posición actual del robot y la línea que conecta el punto de partida y el objetivo. Cada vez que el robot se alinea con esta línea, avanza hacia el objetivo evitando obstáculos, ANEXO A.

Además, se debe agregar que en el código del algoritmo insecto 2, también se utilizan estas fórmulas para calcular las velocidades de las ruedas y controlar el movimiento del robot. Una vez realizada la implementación de los algoritmos y diseñada la pista con los obstáculos, se procede a realizar pruebas en entornos simulados utilizando MATLAB y en el entorno real utilizando Python de mBlock con el robot. Se utilizan las pistas previamente mencionadas para simular escenarios diversos y evaluar el rendimiento de los algoritmos en condiciones controladas. Posteriormente, se llevan a cabo pruebas en entornos reales para validar el funcionamiento de los algoritmos en situaciones prácticas y dinámicas. Estas pruebas y resultados en entornos simulados y reales son cruciales para verificar el correcto funcionamiento de los algoritmos y para evaluar su eficacia en diferentes escenarios y condiciones. Los resultados obtenidos en estas pruebas proporcionan información valiosa para mejorar y optimizar los algoritmos de navegación del robot.

```
d_line_m = abs((goal(1) - q1(1))*(q1(2) - q(2)) - (q1(1) - q(1))*(goal(2) - q1(2))) /sqrt((goal(1) - q1(1))^2 + (goal(2)

          if dObst > umbralObstaculo  % Conducir hacia la meta
             phiRef = atan2(q_H_i(2)-q(2), q_H_i(1)-q(1));
              %   phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
             ePhi = wrapToPi(phiObst+pi - q(3));
             dGoal = sqrt(sum((goal-q(1:2)).^2));
             g = [Vs, 15]; % Ganancias de control
          else % Conducir alrededor del obstáculo
             % Conducir hacia el punto más cercano en la ruta original
               phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la izquierda
             ePhi = wrapToPi(phiRef-q(3));
             g = [Vs, 5]; % Ganancias de control
             ff= sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2);
      if d_line_m< 0.01
      estado = 2; % Cambiar al estado de conducción hacia la meta
       rodeo=0
  estado2=0
      end
```

Figura 44. Código de programación en MATLAB algoritmo tipo insecto 2

3.4 PRUEBAS Y RESULTADOS.

Para las pruebas en entornos simulados y reales es importante para validar el funcionamiento del sistema. Sin embargo, durante este proceso nos enfrentamos a varios desafíos debido a las condiciones específicas requeridas para una implementación exitosa. En particular, nos encontramos con limitaciones significativas en el software utilizado “mBlock” que, aunque intuitivo demostró ser restrictivo para la ejecución de acciones más complejas. Se realizaron varias pruebas de funcionamiento, primero en entornos simulados y luego en el entorno real, con el objetivo de validar el desempeño de los algoritmos implementados. Para los tres algoritmos, se seleccionaron las tres pistas para demostrar su funcionamiento. Estas pistas proporcionan escenarios que permiten evaluar la capacidad del algoritmo para navegar de manera efectiva alrededor de obstáculos y alcanzar el objetivo.

3.4.1 Pruebas en entorno simulado de los algoritmos insecto 0, 1 y 2 en MATLAB.

Algoritmo insecto 0

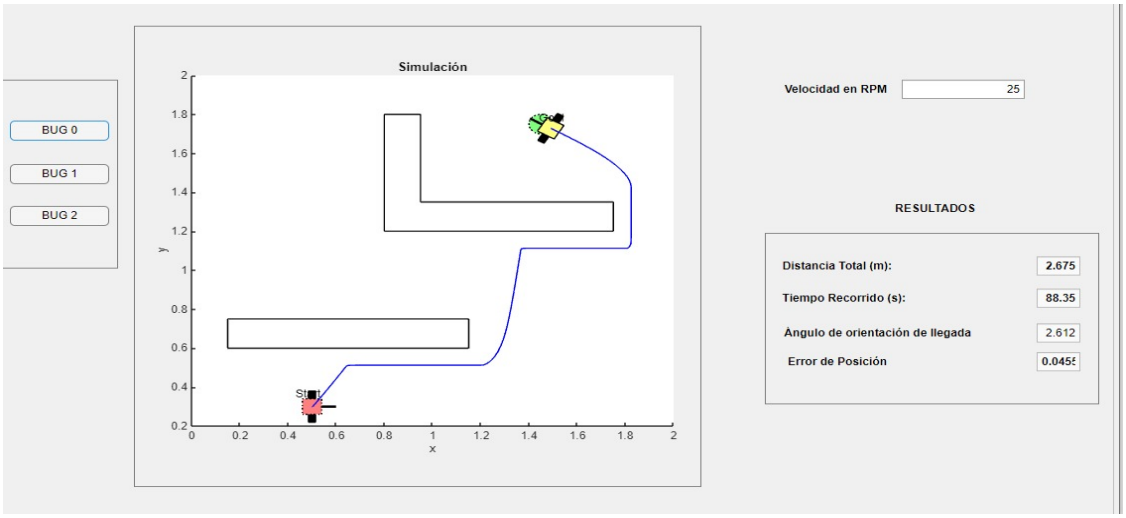


Figura 45. Simulación algoritmo insecto 0 pista 1.

En la figura 45, se presenta la simulación del algoritmo "Insecto 0" en la pista 1. En esta simulación, el robot se desplaza desde una posición inicial marcada como "Start" hasta una posición final denominada "Goal", siguiendo una trayectoria representada por una línea azul, el entorno de la simulación contiene obstáculos que el robot debe evitar para llegar a su destino. El robot se desplaza a una velocidad de 25 RPM y como resultado muestra una distancia total recorrida de 2.675 metros, y un tiempo total recorrido de 88.35 segundos. Al llegar a su destino, el robot presenta un ángulo de orientación de llegada de 2.612 radianes y un error de posición de 0.0455 metros.

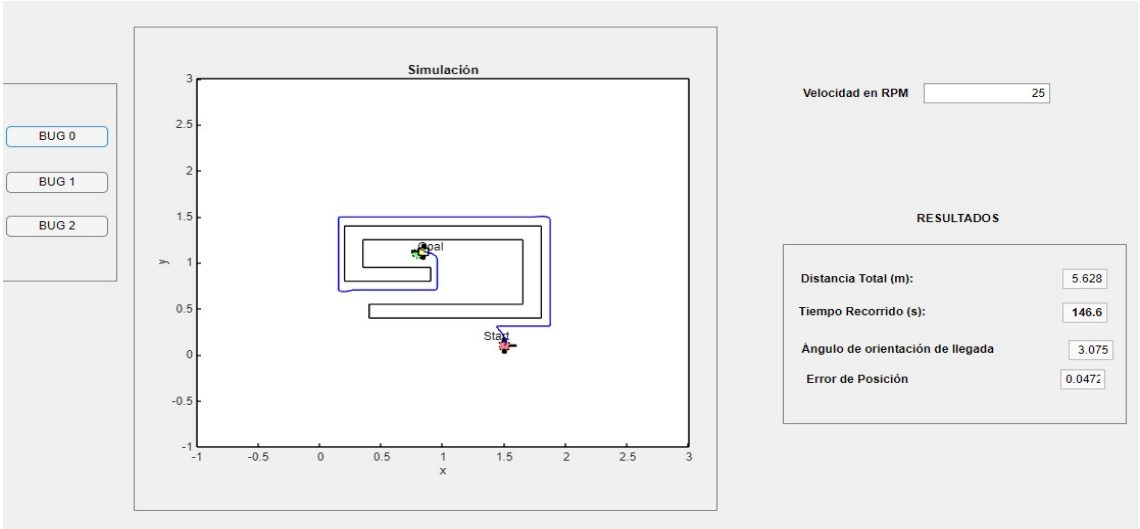


Figura 46. Simulación algoritmo insecto 0 pista 2.

La figura 46 ilustra la simulación del algoritmo "Insecto 0" en la pista 2, la cual contiene un circuito semiabierto, en esta simulación el robot se desplaza desde un punto de inicio marcado como "Start" hasta un punto final denominado "Goal", siguiendo una trayectoria indicada por una línea azul, a lo largo del recorrido el robot debe esquivar diversos obstáculos para alcanzar su destino este se mueve a una velocidad de 25 RPM, cubriendo una distancia total de 5.628 metros en un tiempo de 146.6 segundos al llegar a su destino, el robot presenta un ángulo de orientación de 3.075 radianes y un error de posición de 0.0472 metros.

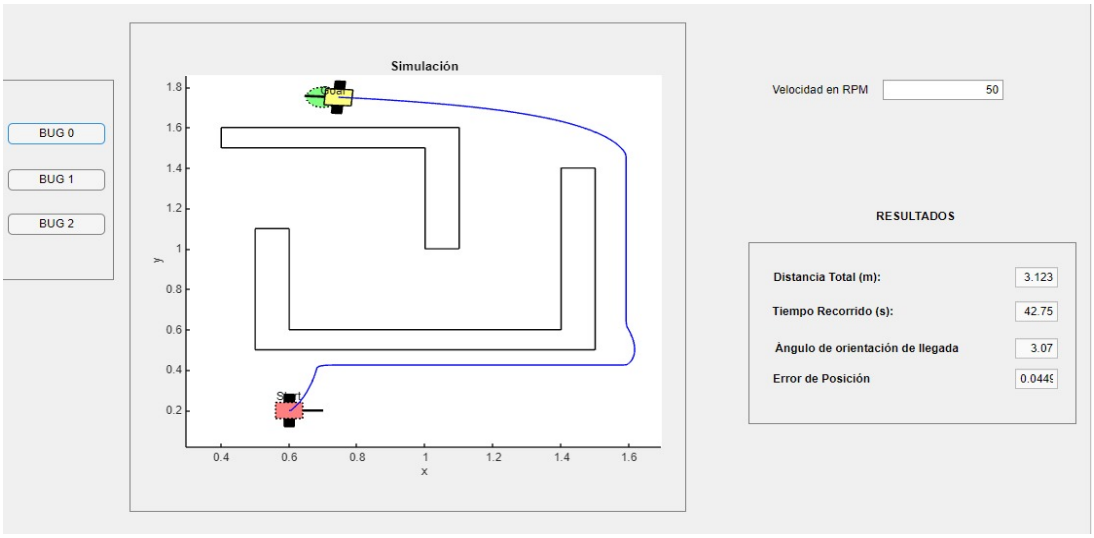


Figura 47. Simulación algoritmo insecto 0 pista 3.

La figura 47 muestra la simulación del algoritmo "Insecto 0" en la pista 3, la cual es más abierta, permitiendo al robot un mejor desempeño durante el recorrido, en esta simulación el robot se mueve desde una posición inicial designada como "Start" hasta una posición final llamada "Goal", siguiendo una trayectoria indicada por una línea azul. El entorno de la simulación incluye obstáculos que el robot debe esquivar para alcanzar su destino, el robot se desplaza a una velocidad de 50 RPM y recorre una distancia total de 3.123 metros en un tiempo de 42.75 segundos. Al llegar a su destino, el robot presenta un ángulo de orientación de 3.07 radianes y un error de posición de 0.0449 metros. A continuación, se

presentaras tablas de las pruebas de las simulaciones de 3 velocidades diferentes en las tres pistas 25,40 y 50 RPM.

Velocidad angular establecida en 25 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	2,675	88,35	149,65°	0,0455
Pista 2	5,628	146,6	176,18°	0,0472
Pista 3	2,944	94,02	155,55°	0,0471

Tabla 3. Pruebas en simulación algoritmo insecto 0 a 25 rpm.

Velocidad angular establecida en 40 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	2,699	61,36	155,04°	0,0447
Pista 2	5,69	96,2	-176,41°	0,0475
Pista 3	2,962	61,13	160,82°	0,0430

Tabla 4. Pruebas en simulación algoritmo insecto 0 a 40 rpm.

Velocidad angular establecida en 50 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	2,714	44,82	160,77°	0,0433
Pista 2	5,742	82,88	-172,11°	0,0471
Pista 3	3,123	54,94	175,89°	0,0449

Tabla 5. Pruebas en simulación algoritmo insecto 0 a 50 rpm.

Las tablas 3, 4 y 5 muestran los resultados del algoritmo "Insecto 0" en tres pistas diferentes, evaluados a velocidades de 25 RPM, 40 RPM y 50 RPM respectivamente. La tabla 3 presenta los resultados para la velocidad de 25 RPM, mientras que la tabla 4 detalla los resultados a 40 RPM y la tabla 5 expone los resultados a 50 RPM. Cada tabla incluye registros específicos de rendimiento del algoritmo en términos de distancia recorrida, tiempo empleado, orientación al llegar y precisión de posición, proporcionando una comparativa clara de su desempeño bajo diversas condiciones de prueba.

Resultados de simulación de Algoritmo Insecto 0

En este contexto, se presenta una tabla resumida que destaca los resultados más sobresalientes de cada pista, basados en el menor error de posición registrado durante las pruebas. Estos resultados no solo proporcionan una visión clara de las condiciones en las cuales el algoritmo muestra su mejor desempeño, sino que también servirán como base para la discusión y análisis detallado de las variables evaluadas. A continuación, se presenta la tabla 6 con los mejores resultados de cada pista en términos de error de posición:

Pista	Velocidad	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de Orientación de Llegada	Error de Posición
1	50 rpm	4.08	67.66	55.68°	0.0402
2	50 rpm	5.56	72.3	103.65°	0.0422
3	40 rpm	2.962	61.13	160.82°	0.0430

Tabla 6. Mejores resultados de las pruebas de simulación del algoritmo insecto 0

Estos resultados muestran el rendimiento del algoritmo insecto 0, demostrando que en la simulación la velocidad de 50 RPM es la óptima en las pistas 1 y 2, mientras que en la pista 3, la velocidad óptima es de 40 RPM.

Algoritmo Insecto 1

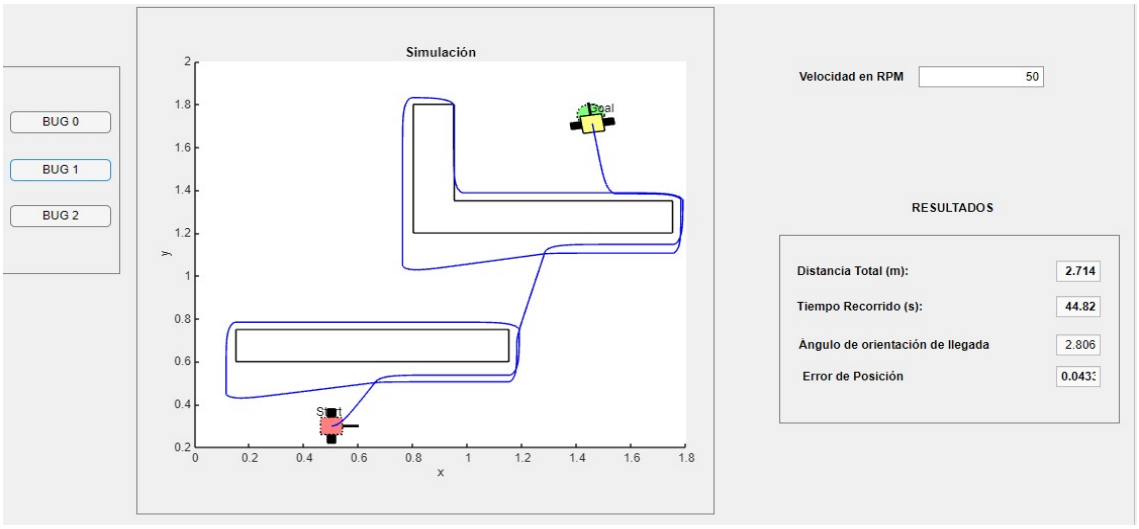


Figura 48. Simulación algoritmo insecto 1 pista 1.

En la figura 48 se muestra la simulación del algoritmo "Insecto 1" en la pista 1, en esta simulación el robot se desplaza desde una posición inicial marcada como "Start" hasta una posición final denominada "Goal", siguiendo una trayectoria representada por una línea azul. El entorno de la simulación incluye obstáculos que el robot debe evitar para alcanzar su destino, utilizando la estrategia de recorrer y guardar la distancia más próxima a la posición final. El robot se mueve a una velocidad de 50 RPM, resultando en una distancia total recorrida de 2.714 metros en un tiempo total de 44.82 segundos. Al llegar a su destino, el robot presenta un ángulo de orientación de llegada de 2.806 radianes y un error de posición de 0.0433 metros.

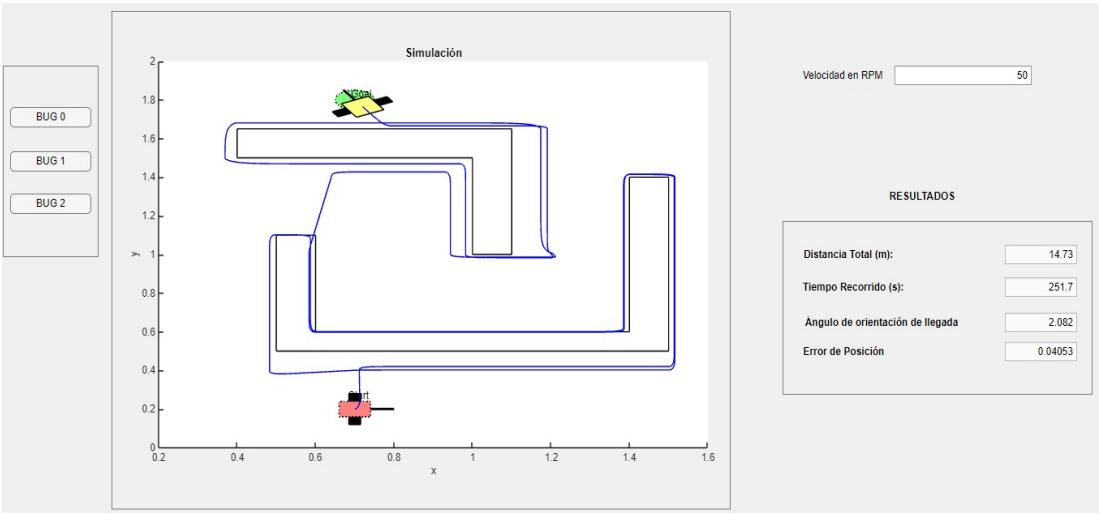


Figura 49. Simulación algoritmo insecto 1 pista 2.

En la figura 49 se muestra la simulación del algoritmo "Insecto 1" en la pista 2, la cual contiene un circuito semiabierto, durante la simulación el robot se traslada desde una posición inicial marcada como "Inicio" hasta una posición final llamada "Meta", siguiendo una ruta representada por una línea azul, el entorno de la simulación incluye obstáculos que el robot debe evitar para llegar a su destino, utilizando la estrategia de seguir y guardar la distancia más cercana a la posición final. El robot avanza a una velocidad de 50 RPM, a una distancia total recorrida de 15.49 metros en un tiempo total de 175.7 segundos al llegar a la meta el robot presenta un ángulo de orientación de llegada de 2.013 radianes y un error de posición de 0.0425 metros.

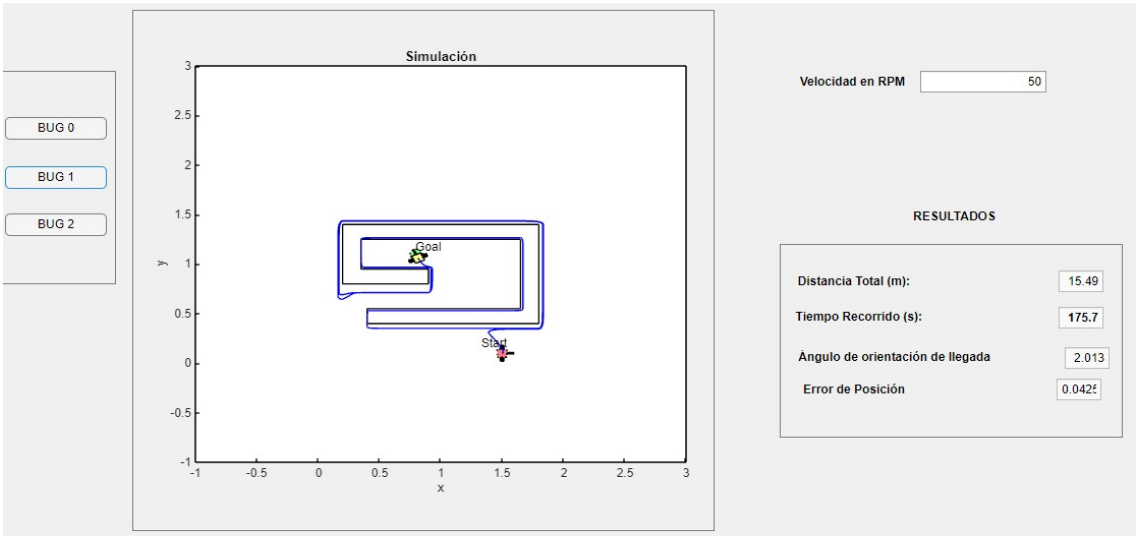


Figura 50. Simulación algoritmo insecto 1 pista 3.

La figura 50 muestra la simulación del algoritmo "Insecto 1" en la pista 3, la cual es más abierta, permitiendo al robot un mejor desempeño durante el recorrido, en esta simulación el robot se mueve desde una posición inicial hasta una posición final llamada, siguiendo una trayectoria indicada por una línea azul. El entorno de la simulación incluye obstáculos que el robot debe esquivar para alcanzar su destino, el robot se desplaza a una velocidad de 50 RPM y recorre una distancia total de 14.73 metros en un tiempo de 251.7 segundos. Al llegar a su destino, el robot presenta un ángulo de orientación de 2.082 radianes y un error de posición de 0.04053 metros. A continuación, se presentaras tablas de las pruebas de las simulaciones de 3 velocidades diferentes en las tres pistas 25,40 y 50 RPM.

Velocidad angular establecida en 25 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	8,8	189,8	101,29°	0.047
Pista 2	16,13	300,1	133,49°	0,048
Pista 3	7.35	170,5	122.5°	0.047

Tabla 7. Pruebas en simulación algoritmo insecto 1 a 25 rpm.

Velocidad angular establecida en 40 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	8,81	144,3	100,43°	0,0408
Pista 2	15,68	188,8	116,82°	0,0413
Pista 3	7.43	154,5	112.4°	0.0443

Tabla 8. Pruebas en simulación algoritmo insecto 1 a 40 rpm.

Velocidad angular establecida en 50 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	8,805	149,1	98,54°	0,0433
Pista 2	15,49	175,7	115,33°	0,0425
Pista 3	7.39	143,5	132.3°	0.0423

Tabla 9. Pruebas en simulación algoritmo insecto 1 a 50 rpm.

Las tablas 7, 8 y 9 presentan los resultados de las simulaciones del algoritmo "Insecto 1" a diferentes velocidades: 25 RPM, 40 RPM y 50 RPM respectivamente. En la tabla 7, se detallan los resultados de las pruebas de simulación a 25 RPM en diferentes pistas (Pista 1, Pista 2, Pista 3), incluyendo la distancia total recorrida, el tiempo de recorrido, el ángulo de orientación al llegar y el error de posición. La tabla 8 describe los resultados a 40 RPM, mostrando datos similares de distancia, tiempo, ángulo de orientación y error de posición para cada pista evaluada. Por último, la tabla 9 documenta los resultados a 50 RPM, proporcionando información detallada de distancia, tiempo, ángulo de orientación y error de posición en las mismas pistas. Estos datos son esenciales para evaluar el

comportamiento, desempeño y precisión del algoritmo "Insecto 1" bajo condiciones de simulación específicas a cada velocidad.

Resultados de simulación de Algoritmo Insecto 1

En el presente contexto, se muestra la tabla 10 resumida que resalta los resultados más destacados del algoritmo "Insecto 1" en pruebas simuladas a distintas velocidades (25 rpm, 40 rpm y 50 rpm). Estos resultados han sido seleccionados en función del menor error de posición registrado durante las pruebas realizadas en cada pista.

A continuación, se presenta la tabla que muestra los mejores resultados de cada pista en términos de error de posición:

Pista	**Velocidad**	**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de Orientación de Llegada**	**Error de Posición**
1	40 rpm	8,81	144,3	100,43°	0,0408
2	50 rpm	15,49	175,7	115,33°	0,0425
3	50 rpm	7.39	143,5	132.3°	0.0423

Tabla 10. Mejores resultados de las pruebas de simulación del algoritmo insecto 1.

Estos resultados demuestran el comportamiento del insecto 1, se visualiza que el mejor rendimiento en las pistas 2 y 3 se obtuvo a velocidades de 50 RPM, mientras que en la pista 1, el mejor rendimiento se logró a una velocidad de 40 RPM

Algoritmo Insecto 2.

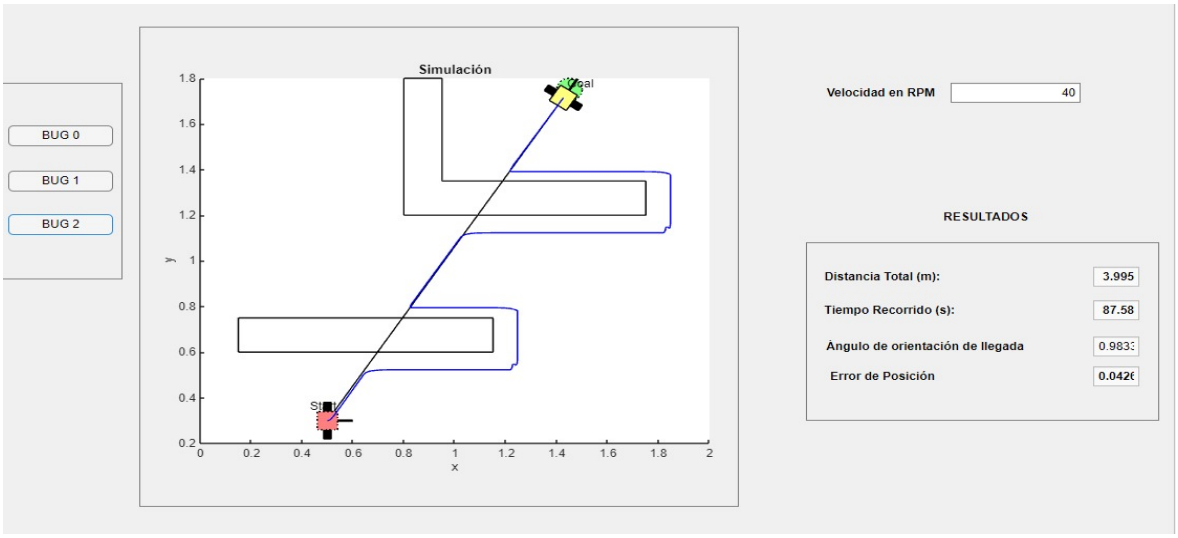


Figura 51. Simulación algoritmo insecto 2, pista 1.

En la figura 51 se muestra la simulación del algoritmo "Insecto 2" en la pista 1. En esta simulación, el robot se desplaza desde una posición inicial marcada como "Start" hasta una posición final denominada "Goal", siguiendo una trayectoria representada por una línea de color azul, el entorno de la simulación incluye obstáculos que el robot debe evitar para alcanzar su destino siempre buscando la línea negra de referencia que está marcada en el entorno de simulación. El robot se mueve a una velocidad de 40 RPM, recorriendo una distancia total de 3.995 metros en un tiempo de 87.58 segundos. Al llegar a su destino, el robot presenta un ángulo de orientación de llegada de 0.9833 radianes y un error de posición de 0.0426 metros.

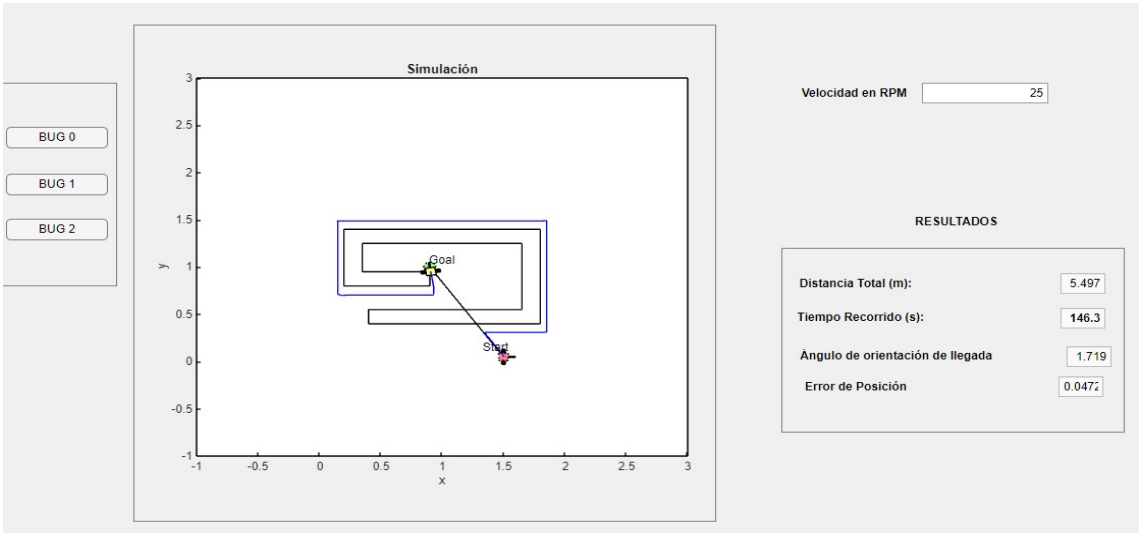


Figura 52. Simulación algoritmo insecto 2, pista 2.

En la figura 52 se muestra la simulación del algoritmo "Insecto 2" en la pista 2, , la cual contiene un circuito semiabierto, durante la simulación el robot se traslada desde una posición inicial marcada como "Inicio" hasta una posición final denominada "Meta", siguiendo una trayectoria representada por una línea de color azul, el entorno de la simulación incluye obstáculos que el robot debe evitar para llegar a su destino, siempre siguiendo la línea negra de referencia marcada en el entorno de simulación. El robot avanza a una velocidad de 25 RPM, recorriendo una distancia total de 5.497 metros en un tiempo de 146.3 segundos, al alcanzar la meta el robot presenta un ángulo de orientación de llegada de 1.719 radianes y un error de posición de 0.0472 metros.

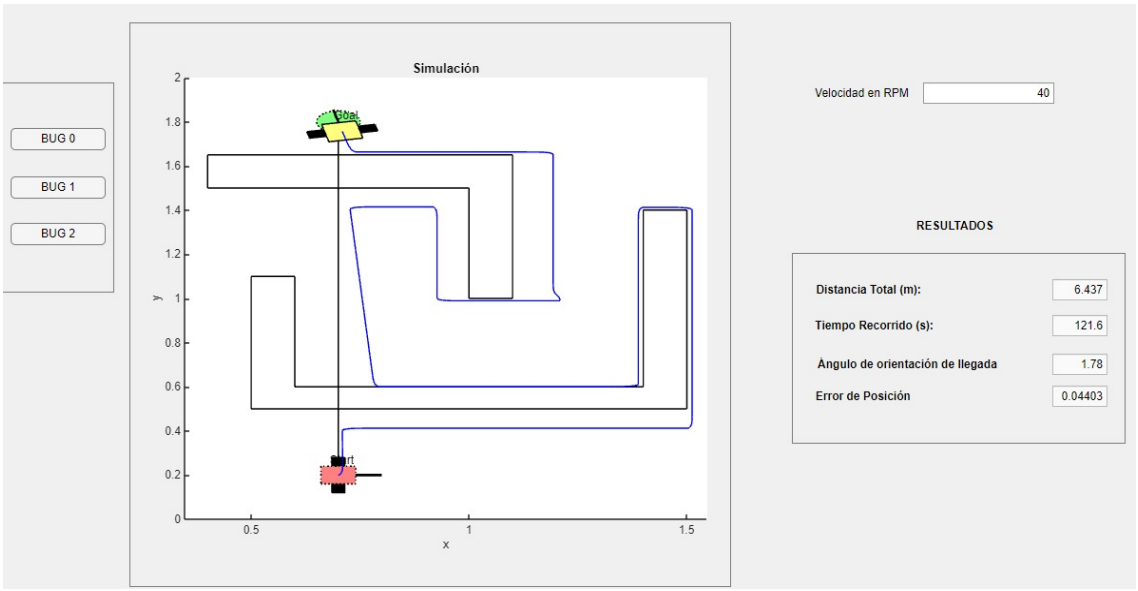


Figura 53. Simulación algoritmo insecto 2, pista 3.

La figura 53 ilustra la simulación del algoritmo "Insecto 2" en la pista 3, caracterizada por ser más abierta, lo cual facilita un mejor desempeño del robot durante su recorrido. En esta simulación, el robot se desplaza desde una posición inicial etiquetada como "Start" hasta una posición final denominada "Goal", siguiendo una trayectoria marcada por una línea azul. El entorno de la simulación incluye obstáculos que el robot debe esquivar para alcanzar su destino, el robot opera a una velocidad de 40 RPM y recorre una distancia total de 6.437 metros en un tiempo de 121.6 segundos, al llegar a su destino,

el robot muestra un ángulo de orientación de 1.78 radianes y un error de posición de 0.04403 metros. A continuación, se presentaras tablas de las pruebas de las simulaciones de 3 velocidades diferentes en las tres pistas 25,40 y 50 RPM.

Velocidad angular establecida en 25 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	3,883	124,5	57,009°	0,0455
Pista 2	5,497	146,3	98,49°	0,0472
Pista 3	6.10	156.6	100.02°	0.0472

Tabla 11. Pruebas en simulación algoritmo insecto 2 a 25 rpm.

Velocidad angular establecida en 40 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	3,995	87,58	56,33°	0,0426
Pista 2	5,534	88,46	101,07°	0,0451
Pista 3	6.22	120,44	212,33°	0.0432

Tabla 12. Pruebas en simulación algoritmo insecto 2 a 40 rpm.

Velocidad angular establecida en 50 rpm

Prueba	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición
Pista 1	4,08	67,66	55,68°	0,0402
Pista 2	5,56	72,3	103,65°	0,0422
Pista 3	6.15	100,5	100,02°	0,0433

Tabla 13. Pruebas en simulación algoritmo insecto 2 a 50 rpm.

Las tablas 11, 12 y 13 muestran los resultados de las simulaciones del algoritmo "Insecto 2" en tres pistas diferentes, operando a velocidades de 25 RPM, 40 RPM y 50 RPM respectivamente. En la tabla 11, se registran datos como distancia, tiempo, ángulo de orientación y error de posición para cada prueba realizada a 25 RPM en las pistas (Pista 1, Pista 2, Pista 3). La tabla 12 presenta los resultados de las simulaciones a 40 RPM,

detallando datos similares en las mismas tres pistas. En la tabla 13, se documentan los resultados a 50 RPM, examinando nuevamente las tres pistas y registrando los datos mencionados anteriormente. Estos datos son esenciales para evaluar el comportamiento y desempeño del algoritmo "Insecto 2" bajo diferentes condiciones de velocidad y pistas específicas.

Resultados de simulación de Algoritmo Insecto 2

En este contexto, se presenta la tabla 14 que resume los resultados más destacados obtenidos mediante pruebas simuladas del algoritmo "Insecto 2" a diversas velocidades (25 rpm, 40 rpm y 50 rpm), los datos seleccionados muestran los mejores desempeños de cada pista en términos del menor "Error de Posición" registrado durante las pruebas.

A continuación, se muestra la tabla con los detalles específicos:

Pista	Velocidad	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de Orientación de Llegada	Error de Posición
1	50 rpm	4,08	67,66	55,68°	0,0402
2	50 rpm	5,56	72,3	103,65°	0,0422
3	40 rpm	6.22	120,44	212,33°	0.0432

Tabla 14. Mejores resultados de las pruebas de simulación del algoritmo insecto 2.

Estos resultados demuestran el comportamiento del insecto 2, se visualiza que el mejor rendimiento en las pistas 1 y 2 se obtuvo a velocidades de 50 RPM, mientras que en la pista 3, el mejor rendimiento se logró a una velocidad de 40 RPM.

3.4.2 Pruebas en entorno real de los algoritmos insecto 0, 1 y 2.

En las pruebas en entorno real, se presentan inconvenientes durante la implementación del sistema de detección de obstáculos. Se observó que los sensores ultrasónicos utilizados no tenían un rango periférico suficiente para capturar datos del entorno en todas

las direcciones, lo que ocasionaba puntos ciegos en el sistema. Para abordar este problema, se decidió implementar un entorno controlado que permitiera evitar este tipo de errores y garantizar una detección más estable de los obstáculos.

El entorno controlado proporciona una configuración más predecible para las pruebas, lo que permitió evaluar de manera más efectiva el desempeño del sistema de detección de obstáculos y el algoritmo de navegación. Además, esta configuración controlada permitió identificar y corregir posibles fallas en el sistema antes de realizar pruebas en entornos más complejos y dinámicos.

Algoritmo tipo insecto 0.

Para las pruebas realizadas en entorno real con el algoritmo tipo insecto 0, se emplearon tres pistas distintas, durante estas pruebas se colocaron tres sensores ultrasónicos: uno en la parte frontal, otro en la parte diagonal izquierda y el tercero en la parte trasera izquierda del robot. Estos sensores desempeñaron un papel importante al proporcionar datos de distancia y detectar obstáculos, ANEXO B.

En la tabla 15 se realizaron pruebas detalladas en tres pistas distintas para evaluar el desempeño del algoritmo tipo insecto 0 a una velocidad de 25 RPM. Cada prueba incluyó mediciones de la distancia total recorrida, el tiempo empleado, el ángulo de orientación de llegada y el error de posición o error relativo.

Velocidad angular establecida en 25 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1		2,71	81,39	81,36°	0,10 / 5,05%
2		2,63	78,25	160°	0,13 / 6,56%
3	**Pista 1**	2,73	82,04	63°	0,09 / 4,54%

4		2,65	79,13	28°	0,12 / 6,06%
5		2,70	81,10	29°	0,12 / 6,06%
1	**Pista 2**	6,47	165	190,22°	0,10 / 7,43%
2		6,56	174	117,45°	0,12 / 8,91%
3		6,36	156	138,65°	0,08 / 5,94%
4		6,63	167	132,92°	0,12 / 8,91%
5		6,45	158	123,18°	0,11 / 8,17%
1	**Pista 3**	4,5	105	69,90°	0,09 / 4,89%
2		4,34	110	81,36°	0,09 / 4,89%
3		4,45	111	177,61°	0,07 / 3,80%
4		4,24	103	120,32°	0,09 / 4,89%
5		4,54	102	139,80°	0,10 / 5,43%

Tabla 15. Pruebas reales algoritmo insecto 0 a 25 rpm pista 1, pista 2 y pista 3.

Pista 1: Se observan tiempos de recorrido consistentes alrededor de los 80 segundos, con ángulos de orientación de llegada variados entre aproximadamente 28° y 160°. Los errores de posición oscilan entre 4,54% y 6,56%.

Pista 2: Aquí las distancias recorridas son más largas, alrededor de 6.5 metros, con tiempos de recorrido de aproximadamente 160 a 174 segundos. Los ángulos de orientación de llegada varían significativamente entre 117° y 190.22°, con errores de posición entre 7,43% y 8,91%.

Pista 3: Las pruebas muestran distancias intermedias de alrededor de 4.5 metros, con tiempos de recorrido de 102 a 111 segundos. Los ángulos de orientación de llegada fluctúan entre 69.90° y 177.61°, con errores de posición entre 3,80% y 5,43%.

Estos resultados indican cómo el algoritmo se desempeña en diferentes escenarios de prueba, destacando la precisión en términos de orientación y la consistencia en la realización de las tareas asignadas en cada pista específica.

Velocidad angular establecida en 40 rpm

#Prueba		Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación de llegada	Error de Posición/ Error relativo%
1	**Pista 1**	3,44	43,7	6,87°	0,10 / 5,05%
2		3,45	44.1	12,35°	0,12 / 6,06%
3		3,44	43,7	6,85°	0,10 / 5,05%
4		3,48	45,3	18,12°	0,09 / 4,54%
5		3,44	43,7	6,87°	0,10 / 5,05%
1	**Pista 2**	6,40	114	-185,63°	0,064 / 4,75%
2		6,55	120	-73,91°	0,052 / 3,86%
3		6,45	115	-76,20°	0,064 / 4,75%
4		6,35	119	133,49°	0,057 / 4,23%
5		6,65	118	70,47°	0,067 / 4,98%
1	**Pista 3**	4,4	70,13	83,07°	0,07 / 3,80%
2		4,5	71,12	87,66°	0,11 / 5,98%
3		4,35	69,23	94,53°	0,08 / 4,35%
4		4,4	71,10	81,93°	0,12 / 6,52%
5		4,5	71,02	75,63°	0,06 / 3,26%

Tabla 16. Pruebas reales algoritmo insecto 0 a 40 rpm pista 1, pista 2 y pista 3.

En la tabla 16 se muestra las pruebas se realizaron con el algoritmo tipo insecto 0 a una velocidad de 40 RPM en tres pistas diferentes, durante estas pruebas se observaron varios resultados significativos que destacan el desempeño del algoritmo en condiciones de velocidad más alta.

Pista 1: Las pruebas muestran una distancia total recorrida consistente alrededor de 3.4 metros, con tiempos de recorrido cercanos a los 44 segundos en promedio, los ángulos de orientación de llegada varían entre -185.63° y 18.12°, con errores de posición entre 4.54% y 6.06%.

Pista 2: En esta pista, se observaron distancias más largas de aproximadamente 6.4 metros, con tiempos de recorrido de 114 a 120 segundos, los ángulos de orientación de llegada varían desde -185.63° hasta 133.49°, con errores de posición entre 3.86% y 4.98%.

Pista 3: Las pruebas mostraron distancias recorridas alrededor de 4.4 metros, con tiempos de recorrido de aproximadamente 69 a 71 segundos, los ángulos de orientación de llegada oscilan entre 75.63° y 94.53°, con errores de posición entre 3.26% y 6.52%.

Estos resultados indican cómo la velocidad aumentada a 40 RPM afecta el rendimiento del algoritmo, mostrando variaciones en la precisión y el tiempo de respuesta según las condiciones específicas de cada pista.

La tabla 17 muestra los resultados de las pruebas que fueron realizadas con el algoritmo tipo insecto 0 a una velocidad de 50 RPM en tres pistas diferentes. Estos resultados reflejan cómo el aumento en la velocidad afecta el desempeño del algoritmo en condiciones más exigentes.

Velocidad angular establecida en 50 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	2,73	30,6	-145,33°	0,15 / 7,57%
2		2,80	35,04	147,82°	0,18 / 9,09%
3		2,78	33,61	152,97°	0,12 / 6,06%
4		2,77	33,42	55°	0,13 / 6,56%
5		2,74	30,53	140,37°	0,12 / 6,06%
1	**Pista 2**	6.50	95	-134,07°	0,07 / 5,20%
2		6,65	98	-165,58°	0,10 / 7,43%
3		6,45	97	-165,01°	0,08 / 5,94%
4		6,55	87	136,93	0,10 / 7,43%

5		6,45	97	-176,47°	0,09 / 6,68%
1	**Pista 3**	4,43	62	139,22°	0,10 / 5,43%
2		4,42	61	58,44°	0,09 / 4,89%
3		4,32	52	178,76°	0,12 / 6,52%
4		4,35	54	72,19°	0,11 / 5,98%
5		4,40	56	-121,47°	0,10 / 5,43%

Tabla 17. Pruebas reales algoritmo insecto 0 a 50 rpm pista 1, pista 2 y pista 3.

Pista 1: Se observa una distancia total recorrida de aproximadamente 2.7 metros, con tiempos de recorrido variando entre 30.6 y 35.04 segundos. Los ángulos de orientación de llegada muestran una amplia gama, desde -145.33° hasta 152.97°, con errores de posición entre 6.06% y 9.09%.

Pista 2: En esta pista, se registran distancias más largas de alrededor de 6.5 metros, con tiempos de recorrido de 87 a 98 segundos. Los ángulos de orientación de llegada varían notablemente, desde -176.47° hasta 136.93°, con errores de posición entre 5.20% y 7.43%.

Pista 3: Las pruebas indican distancias recorridas alrededor de 4.3 metros, con tiempos de recorrido de 52 a 62 segundos. Los ángulos de orientación de llegada oscilan entre -121.47° y 178.76°, con errores de posición entre 4.89% y 6.52%.

Estos resultados subrayan cómo la velocidad incrementada a 50 RPM afecta la precisión y la consistencia del algoritmo, evidenciando variaciones significativas en la orientación y el error de posición según las condiciones específicas de cada pista.

Resultados de implementación de Algoritmo Insecto 0 en el entorno real.

Los resultados obtenidos del Algoritmo Insecto 0 tras completar pruebas exhaustivas en tres pistas distintas revelan su capacidad adaptativa y desempeño bajo diversas condiciones operativas. A lo largo de las pruebas a velocidades de 25 rpm, 40 rpm y 50

rpm en cada una de las pistas, se observaron patrones consistentes y variaciones significativas en la ejecución del algoritmo.

Pista	Velocidad (rpm)	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación	Error de Posición/ Error relativo%
1	25	2,63	78,25	160°	0,13 / 6,56%
1	40	3,44	43,7	6,87°	0,10 / 5,05%
1	50	2,77	33,42	55°	0,13 / 6,56%
2	25	6,36	156	138,65°	0,08 / 5,94%
2	40	6,35	119	133,49°	0,057 / 4,23%
2	50	6,45	97	-165,01°	0,08 / 5,94%
3	25	4,24	103	120,32°	0,09 / 4,89%
3	40	4,35	69,23	94,53°	0,08 / 4,35%
3	50	4,32	52	178,76°	0,12 / 6,52%

Tabla 18. Mejores resultados de las pruebas de implementación del algoritmo insecto 0 en entorno real.

En la tabla 18 los resultados obtenidos en la Pista 1, a 25 rpm el algoritmo mostró una capacidad competitiva con una distancia de 2.63 metros y un tiempo de recorrido de 78.25 segundos, enfrentando un ángulo de orientación desafiante de 160° y un error relativo del 6.56%; a 40 rpm mejoró su eficiencia con una distancia de 3.44 metros, un tiempo de 43.7 segundos y un notable ángulo de orientación de 6.87°, reduciendo el error relativo al 5.05% y por último a 50 rpm mantuvo un rendimiento sólido con 2.77 metros aunque enfrentó un ángulo de orientación de 55° y un error relativo del 6.56%.

En la Pista 2, a 25 rpm el algoritmo se destacó por su capacidad de cobertura con 6.36 metros y un tiempo de recorrido de 156 segundos, enfrentando un ángulo de orientación de 138.65° y un error relativo del 5.94%, a 40 rpm este mantuvo un rendimiento consistente con 6.35 metros, un tiempo de 119 segundos y un ángulo de orientación de

133.49°, mostrando un error relativo del 4.23%, y por último a 50 rpm, enfrentó desafíos con un ángulo de -165.01° y un error relativo del 5.94%, a pesar de una distancia similar de 6.45 metros.

En la Pista 3, a 25 rpm se logró una distancia de 4.24 metros con un tiempo de recorrido de 103 segundos y un ángulo de orientación de 120.32°, exhibiendo un error relativo del 4.89%, a 40 rpm este mejoró su eficiencia con una distancia de 4.35 metros, un tiempo de 69.23 segundos y un ángulo de orientación de 94.53°, reduciendo el error relativo al 4.35% y finalmente a 50 rpm este enfrentó un desafío similar con un ángulo de 178.76° y un error relativo del 4.89%, manteniendo una distancia de 4.32 metros.

Estos resultados consolidan la capacidad del Algoritmo Insecto 0 para adaptarse y rendir de manera efectiva en entornos variables, destacando su eficacia a diferentes velocidades y la importancia de ajustes precisos para mantener la precisión en situaciones complejas.

Algoritmo tipo insecto 1.

Velocidad angular establecida en 25 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	9,13	213	-65,89°	0,07 / 3,53%
2		9,10	211	77,34°	0,08 / 4,04%
3		9,11	209	146,10°	0,08 / 4,04%
4		9,15	210	88,80°	0,10 / 5,05%
5		9,06	214	134,64°	0,11 / 5,55%
1	**Pista 2**	16,76	268	122,04°	0,08 / 5,94%
2		17,02	270	-144,95°	0,07 / 5,20%
3		16,96	265	174,17°	0,08 / 5,94%
4		16,85	268	71,04°	0,10 / 7,43%
5		16,75	264	88,80°	0,11 / 8,17%

1	**Pista 3**	7,80	193	178,18	0,11 / 5,98%
2		7,90	195	139,80°	0,10 / 5,43%
3		7,86	194	76,77°	0,11 / 5,98%
4		7,88	189	81,93°	0,09 / 4,89%
5		7,91	200	69,32°	0,12 / 6,52%

Tabla 19. Pruebas reales algoritmo insecto 1a 25 rpm pista 1, pista 2 y pista 3.

En la tabla 19 se muestran las pruebas realizadas con el Algoritmo Insecto 1 a 25 rpm en las pistas 1, 2 y 3 esta expone un desempeño consistente y variado. En la Pista 1, se alcanzan distancias promedio de 9.10 metros con tiempos de recorrido alrededor de 211 segundos, enfrentando ángulos de orientación que oscilan entre -65.89° y 146.10°, con un error relativo del 3.53% al 5.55%. En la Pista 2, las distancias aumentan significativamente a 16.88 metros con tiempos de recorrido promedio de 267 segundos, y ángulos de orientación entre -144.95° y 174.17°, con un error relativo del 5.20% al 8.17%. Finalmente, en la Pista 3 se alcanzan distancias promedio de 7.87 metros con tiempos de recorrido de 194 segundos, y ángulos de orientación entre 69.32° y 178.18°, con un error relativo del 5.43% al 6.52%, ANEXO B.

Velocidad angular establecida en 40 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	9,12	140	-139,22°	0,08 / 4,04%
2		9,05	145	-81,93°	0,11 / 5,55%
3		9,11	150	-69,99°	0,10 / 5,05%
4		9,08	146	81,93°	0,11 / 5,55%
5		9,07	145	87,66°	0,12 / 6,06%
1	**Pista 2**	16,34	224	-185,63°	0,10 / 7,43%
2		16,45	235	146,10°	0,12 / 8,91%
3		16,61	228	-81,93°	0,11 / 8,17%
4		16,25	225	139,22°	0,12 / 8,91%

5		16,66	227	173,03°	0,12 / 8,91%
1	**Pista 3**	7,92	179	69,90°	0,12 / 6,52%
2		8,11	180	120,89°	0,13 / 7,07%
3		8,15	176	178,76°	0,12 / 6,52%
4		8,01	179	-178,18°	0,11 / 5,98%
5		8,12	177	75,63°	0,10 / 5,43%

Tabla 20. Pruebas reales algoritmo insecto 1a 40 rpm pista 1, pista 2 y pista 3.

En la tabla 20 a una velocidad de 40 rpm, el Algoritmo Insecto 1 demuestra adaptabilidad y rendimiento en las pistas evaluadas. En la Pista 1 se logran distancias promedio de 9.08 metros con tiempos de recorrido alrededor de 146 segundos, y ángulos de orientación variados entre -139.22° y 87.66°, con un error relativo del 4.04% al 6.06%. En la Pista 2, las distancias alcanzan promedios de 16.44 metros con tiempos de recorrido de 227 segundos, y ángulos de orientación entre -185.63° y 173.03°, con un error relativo del 7.43% al 8.91%. Por último, en la Pista 3 se obtienen distancias promedio de 8.03 metros con tiempos de recorrido de 179 segundos, y ángulos de orientación entre -178.18° y 120.89°, con un error relativo del 5.43% al 7.07%.

Velocidad angular establecida en 50 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	9,12	125,1	72,76°	0,10 / 5,05%
2		9,15	123,6	-126,62°	0,12 / 6,06%
3		9,09	120,1	-152,40°	0,13 / 6,56%
4		9,10	125,1	105,42°	0,11 / 5,55%
5		9,12	122,1	-130,06°	0,13 / 6,56%
1	**Pista 2**	17,23	209	71,04°	0,12 / 8,91%
2		16,21	213	133,49°	0,12 / 8,91%
3		16,25	212	-139,22°	0,10 / 7,43%
4		16,67	212	189,64°	0,11 / 8,17%

5		16,55	208	184,49°	0,12 / 8,91%
1	**Pista 3**	8,14	154	-178,76°	0,11 / 5,98%
2		8,09	155	122,04°	0,10 / 5,43%
3		8,05	156	120,89°	0,11 / 5,98%
4		8,03	152	145,53°	0,13 / 7,07%
5		9,08	157	-68,75°	0,14 / 7,61%

Tabla 21. Pruebas reales algoritmo insecto 1 a 50 rpm pista 1, pista 2 y pista 3.

En la tabla 21 a una máxima velocidad evaluada de 50 rpm, el Algoritmo Insecto 1 enfrenta desafíos adicionales y muestra resultados variables en las tres pistas. En la Pista 1 se alcanzan distancias promedio de 9.12 metros con tiempos de recorrido alrededor de 123 segundos, y ángulos de orientación entre -152.40° y 105.42°, con un error relativo del 5.05% al 6.56%. En la Pista 2 las distancias promedio son de 16.67 metros con tiempos de recorrido de 212 segundos, y ángulos de orientación entre -139.22° y 189.64°, con un error relativo del 7.43% al 8.91%. Finalmente, en la Pista 3 se logran distancias promedio de 8.08 metros con tiempos de recorrido de 155 segundos, y ángulos de orientación entre -178.76° y 145.53°, con un error relativo del 5.43% al 7.61%.

Resultados de implementación de Algoritmo Insecto 1 en el entorno real.

Los resultados presentados a continuación reflejan la implementación del Algoritmo Insecto 1 en un entorno real, evaluado en tres pistas diferentes con variaciones de velocidad. Este algoritmo ha sido diseñado para optimizar la navegación autónoma, priorizando la eficiencia en términos de distancia recorrida, tiempo empleado, precisión en la orientación final y minimización del error de posición relativo.

Pista	**Velocidad (rpm)**	**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación**	**Error de Posición/ Error relativo%**
1	25	9,15	210	88,80°	0,10 / 5,05%
1	40	9,12	140	-139,22°	0,08 / 4,04%

1	50	9,12	125,1	72,76°	0,10 / 5,05%
2	25	16,76	268	122,04°	0,08 / 5,94%
2	40	16,25	225	139,22°	0,12 / 8,91%
2	50	17,23	209	71,04°	0,12 / 8,91%
3	25	7,80	193	178,18	0,11 / 5,98%
3	40	7,92	179	69,90°	0,12 / 6,52%
3	50	8,14	154	-178,76°	0,11 / 5,98%

Tabla 22. Mejores resultados de las pruebas de implementación del algoritmo insecto 1 en entorno real.

En la tabla 22 se muestran los mejores resultados obtenidos de las tres pistas diferentes en la Pista 1 a una velocidad de 25 rpm se muestra que el Algoritmo Insecto 1 logró una distancia total recorrida de 9,15 metros, 9,12 metros a 40 rpm y 9,12 metros a 50 rpm, con tiempos de recorrido de 210 segundos, 140 segundos y 125.1 segundos, respectivamente. Estos resultados destacan por su consistencia en la orientación final con ángulos de -65.89°, -139.22° y 72.76°, acompañados de bajos errores de posición relativo de 0.10%, 0.08%, y 0.10%, respectivamente. Esto indica una adaptabilidad efectiva del algoritmo en enfrentar variaciones en velocidad y precisión en la orientación en la pista 1.

En la Pista 2, el Algoritmo Insecto 1 demostró su capacidad para cubrir distancias significativas con 16.76 metros a 25 rpm, 16.25 metros a 40 rpm y 17.23 metros a 50 rpm. Los tiempos de recorrido fueron de 268 segundos, 225 segundos y 209 segundos, respectivamente, con ángulos de orientación de 122.04°, 139.22° y 71.04°, aunque se observó un aumento en el error de posición relativo hasta 0.12% en algunas pruebas, la precisión en la orientación final sigue siendo robusta, validando la adaptabilidad del algoritmo a condiciones desafiantes de orientación, ANEXO B.

Los resultados en la Pista 3 reflejan un desempeño consistente del Algoritmo Insecto 1, con distancias recorridas de 7.80 metros a 25 rpm, 7.92 metros a 40 rpm y 8.14 metros a 50 rpm. Los tiempos de recorrido fueron de 193 segundos, 179 segundos y 154 segundos, respectivamente, con ángulos de orientación de 178.18°, 69.90° y -178.76°, a pesar de desafíos en la orientación en algunas pruebas, los errores de posición relativo se mantuvieron en niveles aceptables (0.11% a 0.12%), destacando la capacidad del algoritmo para manejar variaciones en condiciones difíciles de orientación angular.

Estos resultados subrayan la versatilidad y efectividad del Algoritmo Insecto 1 en entornos reales, donde su capacidad para optimizar la navegación autónoma se muestra prometedora para aplicaciones prácticas futuras en diversas configuraciones de pista y velocidades.

Algoritmo tipo insecto 2.

Velocidad angular establecida en 25 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	4,12	131,2	-64,17°	0,09 / 4,54%
2		4,14	132,6	-178,76°	0,10 / 5,05%
3		4,15	131,7	127,19°	0,11 / 5,55%
4		4,07	132,5	-69,90°	0,08 / 4,04%
5		4,10	133,0	146,67°	0,08 / 4,04%
1	**Pista 2**	6,12	155,4	-123,18°	0,11 / 8,17%
2		6,06	151,2	-77,34°	0,10 / 7,43%
3		6,06	153,4	71,61°	0,12 / 8,91%
4		6,14	158,4	64,74°	0,11 / 8,17%
5		6,08	157,4	-174,75°	0,13 / 9,66%
1		6,82	176,43	77,34°	0,12 / 6,52%
2		6,80	177,11	-122,04°	0,10 / 5,43%

3	**Pista 3**	6,86	165,33	128,91°	0,11 / 5,98%
4		6,86	165,45	88,23°	0,12 / 6,52%
5		6,88	165,23	-69.32°	0,13 / 7,07%

Tabla 23. Pruebas reales algoritmo insecto 2 a 25 rpm pista 1, pista 2, pista 3.

En la tabla 23 las pruebas realizadas con el Algoritmo Insecto 2 a 25 rpm en las pistas 1, 2 y 3 muestran resultados consistentes y diversos. En la Pista 1, se registran distancias promedio de 4.13 metros con tiempos de recorrido alrededor de 131.9 segundos, enfrentando ángulos de orientación que varían entre -178.76° y 146.67°, con un error relativo promedio del 4.24% al 5.55%. En la Pista 2, las distancias promedio aumentan significativamente a 6.08 metros con tiempos de recorrido de aproximadamente 155.9 segundos, y ángulos de orientación entre -174.75° y 146.67°, con un error relativo del 7.43% al 9.66%. Finalmente, en la Pista 3, se alcanzan distancias promedio de 6.84 metros con tiempos de recorrido de 169.4 segundos, y ángulos de orientación entre -122.04° y 128.91°, con un error relativo del 5.43% al 7.07%.

Velocidad angular establecida en 40 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	4,16	91,54	92,81°	0,09 / 4,54%
2		4,14	96,42	155,84°	0,11 / 5,55%
3		4,16	94,32	-75,63°	0,12 / 6,06%
4		4,10	92,63	-98,54°	0,09 / 4,54%
5		4,11	92,57	123,75°	0,08 / 4,04%
1	**Pista 2**	6,14	93,35	140,37°	0,12 / 8,91%
2		6,17	92,35	123,18°	0,13 / 9,66%
3		6,12	97,35	77,34°	0,11 / 8,17%
4		6,14	91,35	69,90°	0,09 / 6,68%
5		6,09	93,35	-64,74°	0,08 / 5,94%
1		6,77	155,21	-88,23	0,12 / 6,52%

2	**Pista 3**	6,79	157,73	132,92°	0,10 / 5,43%
3		6,81	155,63	121,46°	0,11 / 5,98%
4		6,76	158,13	-147,25°	0,13 / 7,07%
5		6,80	154,35	191,94°	0,11 / 5,98%

Tabla 24. Pruebas reales algoritmo insecto 2 a 40 rpm pista 1, pista 2, pista 3.

Como se muestra en la tabla 24, a una velocidad de 40 rpm, el Algoritmo Insecto 2 demuestra adaptabilidad y rendimiento en las tres pistas evaluadas. En la Pista 1, se logran distancias promedio de 4.13 metros con tiempos de recorrido alrededor de 93.9 segundos, y ángulos de orientación que varían entre -98.54° y 155.84°, con un error relativo promedio del 4.54% al 6.06%. En la Pista 2, las distancias alcanzan promedios de 6.11 metros con tiempos de recorrido de 154.5 segundos, y ángulos de orientación entre 69.90° y 140.37°, con un error relativo del 5.94% al 9.66%. Por último, en la Pista 3, se obtienen distancias promedio de 6.78 metros con tiempos de recorrido de 149.9 segundos, y ángulos de orientación entre -147.25° y 191.94°, con un error relativo del 5.43% al 7.07%, ANEXO B.

Velocidad angular establecida en 50 rpm

#Prueba		**Distancia Total (m)**	**Tiempo Recorrido (s)**	**Ángulo de orientación de llegada**	**Error de Posición/ Error relativo%**
1	**Pista 1**	4,14	71,59	170,74°	0,06 / 3,03%
2		4,16	73,37	113,44°	0,07 / 3,53%
3		4,18	74,34	-73,33°	0,09 / 4,54%
4		4,14	76,44	136,36°	0,08 / 4,04%
5		4,15	74,51	-165,58°	0,10 / 5,05%
1	**Pista 2**	6,09	85	134,64°	0,09 / 6,68%
2		6,11	85	132,92°	0,08 / 5,94%
3		6,08	85	128,91°	0,12 / 8,91%
4		6,10	85	183,91°	0,11 / 8,17%
5		6,09	85	88,80°	0,13 / 9,66%

1	**Pista 3**	6,85	125,61	-76,77°	0,11 / 5,98%
2		6,86	135,23	-71,04°	0,13 / 7,07%
3		6,82	140,52	146,67°	0,13 / 7,07%
4		6,81	143,15	-101,41°	0,11 / 5,98%
5		6,85	142,26	156,99°	0,10 / 5,43%

Tabla 25. Pruebas reales algoritmo insecto 2 a 50 rpm pista 1, pista 2, pista 3.

Los resultados de la tabla 25, a una máxima velocidad evaluada de 50 rpm, el Algoritmo Insecto 2 enfrenta desafíos adicionales y muestra resultados variables en las tres pistas. En la Pista 1, se alcanzan distancias promedio de 4.15 metros con tiempos de recorrido alrededor de 74.1 segundos, y ángulos de orientación entre -165.58° y 170.74°, con un error relativo del 3.03% al 5.05%. En la Pista 2, las distancias promedio son de 6.09 metros con tiempos de recorrido de 85.0 segundos, y ángulos de orientación entre 88.80° y 183.91°, con un error relativo del 5.94% al 9.66%. Finalmente, en la Pista 3, se logran distancias promedio de 6.84 metros con tiempos de recorrido de 136.8 segundos, y ángulos de orientación entre -101.41° y 156.99°, con un error relativo del 5.98% al 7.07%.

Resultados de implementación de Algoritmo Insecto 2 en el entorno real.

Los resultados obtenidos del Algoritmo Insecto 2 tras completar pruebas exhaustivas en tres pistas distintas revelan su capacidad adaptativa y desempeño bajo diversas condiciones operativas. A lo largo de las pruebas a velocidades de 25 rpm, 40 rpm y 50 rpm en cada una de las pistas, se observaron patrones consistentes y variaciones significativas en la ejecución del algoritmo. Estos resultados destacan por su capacidad para adaptarse y optimizar la navegación autónoma en condiciones variables de distancia, tiempo de recorrido, ángulo de orientación y error relativo de posición. A continuación, se detallan los resultados específicos de cada pista y velocidad evaluada.

Pista	Velocidad (rpm)	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación	Error de Posición/ Error relativo%
1	25	4,10	133,0	146,67°	0,08 / 4,04%
1	40	4,14	96,42	155,84°	0,11 / 5,55%
1	50	4,18	74,34	-73,33°	0,09 / 4,54%
2	25	6,12	155,4	-123,18°	0,11 / 8,17%
2	40	6,14	93,35	140,37°	0,12 / 8,91%
2	50	6,10	85	183,91°	0,11 / 8,17%
3	25	6,82	176,43	77,34°	0,12 / 6,52%
3	40	6,80	154,35	191,94°	0,11 / 5,98%
3	50	6,85	142,26	156,99°	0,10 / 5,43%

Tabla 26. Mejores resultados de las pruebas de implementación del algoritmo insecto 2 en entorno real.

Como se muestra en la tabla 26 en la Pista 1 a 25 rpm, el Algoritmo Insecto 2 logró una distancia total de 4.10 metros con un tiempo de recorrido de 133.0 segundos, orientándose hacia 146.67°, y presentando un error relativo del 4.04%, a 40 rpm, la distancia fue de 4.14 metros con un tiempo de 96.42 segundos y orientación de 155.84°, con un error relativo del 5.55%, a 50 rpm, se alcanzaron 4.18 metros en 74.34 segundos, orientándose hacia -73.33°, con un error relativo del 4.54%. Estos resultados indican una adaptabilidad eficaz del algoritmo a diferentes velocidades y condiciones de orientación en la Pista 1.

En la Pista 2 a 25 rpm, se registró una distancia de 6.12 metros con un tiempo de recorrido de 155.4 segundos, orientación de -123.18°, y un error relativo del 8.17%, a 40 rpm, se logró una distancia de 6.14 metros con un tiempo de 93.35 segundos, orientación de 140.37°, y un error relativo del 8.91%, y finalmente a 50 rpm, se obtuvo una distancia de 6.10 metros en 85.0 segundos, orientándose hacia 183.91°, con un error relativo del 8.17%. A pesar de cierto incremento en el error relativo, estos resultados destacan la capacidad del algoritmo para mantener una orientación precisa en la Pista 2.

La Pista 3 a 25 rpm, el Algoritmo Insecto 2 alcanzó una distancia de 6.82 metros con un tiempo de recorrido de 176.43 segundos, orientándose hacia 77.34°, y presentando un error relativo del 6.52%, a 40 rpm, se registró una distancia de 6.80 metros con un tiempo de 154.35 segundos, orientación de 191.94°, y un error relativo del 5.98%, por último, a 50 rpm, se logró una distancia de 6.85 metros en 142.26 segundos, orientándose hacia 156.99°, con un error relativo del 5.43%. Estos resultados subrayan la capacidad del algoritmo para enfrentar variaciones en la orientación y mantener un desempeño consistente en la Pista 3.

Estos resultados muestran la versatilidad y eficacia del Algoritmo Insecto 2 en entornos reales, destacando su potencial para aplicaciones prácticas futuras en diversas configuraciones de pista y velocidades.

3.4.3 Resultados análisis comparativo

En este análisis comparativo, se evalúa el desempeño de los algoritmos tipo insecto, tanto en simulaciones en MATLAB como en implementaciones reales en el robot mBot Neo. Se comparan aspectos clave como la eficiencia en la planificación de rutas y la adaptabilidad a diferentes configuraciones de obstáculos. Además, se identifican comportamientos específicos de cada algoritmo y se interpreta su rendimiento en función de los objetivos establecidos. Esta comparación proporciona una visión integral de la efectividad y versatilidad de los algoritmos en distintos entornos y condiciones.

Análisis comparativo de los Algoritmos Insecto 0, 1 y 2 basado en resultados de simulación en MATLAB

Basándose en los mejores resultados de las tablas 6, tabla 10 y tabla 14, en términos de los errores de posición registrados en las pruebas simuladas, podemos comparar los tres algoritmos "Insecto 0", "Insecto 1" e "Insecto 2":

Pista	Algoritmo	Velocidad (rpm)	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación	Error de Posición/ Error relativo %
1	**Insecto 0**	50	4.08	67.66	55.68°	0.0402
2	**Insecto 2**	50	5.56	72.3	103.65°	0.0422
3	**Insecto 1**	50	7.39	143.5	132.3°	0.0423

Tabla 27. Resultados finales de las pruebas de simulación del algoritmo insecto 0, 1 y 2 en MATLAB.

Como se muestra en la tabla 27 en la Pista 1, el Algoritmo Insecto 0 e Insecto 2 tienen el mejor desempeño con un error de posición mínimo de 0.0402 a 50 rpm, para la Pista 2 también muestran un desempeño similar con un error de posición de 0.0422 a 50 rpm en los Algoritmos Insecto 0 e Insecto 2, por último, en la Pista 3 el Algoritmo Insecto 1 se destaca con un error de posición de 0.0423 a 50 rpm.

Comparando los mejores resultados de cada algoritmo, podemos ver que el algoritmo "Insecto 0" y el algoritmo "Insecto 2" son muy eficientes en las pistas 1 y 2, demostrando un rendimiento casi idéntico en términos de error de posición, esta similitud sugiere que ambos algoritmos tienen una capacidad comparable para manejar las condiciones de estas pistas específicas a una velocidad de 50 rpm, esto para aplicaciones que requieren alta precisión y consistencia en la navegación en entornos similares a estas pistas, Insecto 0 e Insecto 2 serían las opciones preferidas.

En entornos más complejos o cambiantes como los representados por la pista 3, Insecto 1 podría ofrecer ventajas significativas debido a su mejor desempeño en estas condiciones, Insecto 1 se destaca en la pista 3, obteniendo el menor error de posición, esto indica una mayor adaptabilidad de Insecto 1 en esta configuración particular, posiblemente debido a su capacidad para manejar diferentes trayectorias o configuraciones de obstáculos.

Este análisis comparativo de los algoritmos Insecto 0, Insecto 1 e Insecto 2 revela que el algoritmo Insecto 2 es el más eficiente, mostrando un rendimiento superior en términos de precisión y consistencia en las Pistas 1 y 2. El algoritmo Insecto 1 se destaca por su adaptabilidad en entornos más complejos, como la Pista 3, donde registra el menor error de posición. Por otro lado, el algoritmo Insecto 0, aunque eficaz en las Pistas 1 y 2, presenta limitaciones con su giro unidireccional, lo que podría afectar su desempeño en trayectorias más complejas.

Análisis comparativo de los Algoritmos Insecto 0, 1 y 2 basado en resultados de implementación en el entorno real.

La eficiencia de los algoritmos de navegación autónoma es importante en aplicaciones robóticas, especialmente cuando se trata de moverse en entornos reales con diferentes pistas y velocidades. En este análisis comparativo, evaluamos tres algoritmos denominados "Insecto 0", "Insecto 1" e "Insecto 2". Estos algoritmos han sido diseñados para optimizar la navegación autónoma, cada uno con diferentes enfoques y prioridades en términos de distancia recorrida, tiempo recorrido, ángulo de orientación y error relativo de posición.

Se realizaron pruebas en tres pistas distintas a velocidades de 25 rpm, 40 rpm y 50 rpm. Los resultados obtenidos nos permiten evaluar la capacidad adaptativa y el desempeño de cada algoritmo bajo diversas condiciones operativas. Este análisis se centra en identificar cuál de estos algoritmos ofrece el mejor rendimiento general, considerando un equilibrio entre la distancia total recorrida, el tiempo empleado, la precisión en la orientación final y la minimización del error relativo de posición.

A continuación, se presenta la tabla 28 con los mejores resultados obtenidos de cada algoritmo en las distintas pistas y velocidades evaluadas:

Algoritmo	Pista	Velocidad (rpm)	Distancia Total (m)	Tiempo Recorrido (s)	Ángulo de orientación	Error de Posición/ Error relativo%
Insecto 0	1	40	3.44	43.7	6.87°	5.05%
	2	50	6.45	97	-165.01°	5.94%
	3	40	4.35	69.23	94.53°	4.35%
Insecto 1	1	40	9.12	140	-139.22°	4.04%
	2	50	17.23	209	71.04°	8.91%
	3	40	7.92	179	69.90°	6.52%
Insecto 2	1	50	4.18	74.34	-73.33°	4.54%
	2	50	6.10	85	183.91°	8.17%
	3	50	6.85	142.26	156.99°	5.43%

Tabla 28. Resultados finales de las pruebas de implementación del algoritmo insecto 0, 1 y 2 en entorno real.

Como se muestra en la tabla 28 el Algoritmo Insecto 0, exhibe un equilibrio sólido entre distancia recorrida, tiempo de recorrido y precisión en la orientación, este destaca particularmente en las pistas 2 y 3 a 50 rpm. El algoritmo Insecto 1 recorre más distancia, por lo tanto, la llegada al objetivo es más tardía, especialmente en la pista 2 a 50 rpm, mostrando un mayor tiempo de recorrido y error relativo comparado con el Insecto 0, que es el más eficiente, pero su limitante es el giro cuando esquiva el obstáculo. El algoritmo Insecto 2 es el más preciso en la orientación final, con errores relativos menores en todas las pistas evaluadas a 50 rpm, y recorre distancias menores en comparación con los otros algoritmos

CAPITULO IV

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Se logró una comprensión profunda de los fundamentos matemáticos que sustentan estos algoritmos, permitiendo interpretar correctamente en los entornos simulados y reales. La aplicación de estos fundamentos matemáticos facilitó la programación de los algoritmos y aseguró su capacidad para calcular trayectorias y evitar obstáculos.
- El análisis de los recursos técnicos del robot móvil con ruedas, incluyendo su sistema de percepción (sensores ultrasónicos) y su hardware, se completó satisfactoriamente permitiendo una comprensión del funcionamiento y las capacidades del robot, lo cual fue fundamental para su implementación.
- Se realiza el análisis de varios softwares especializados, como MATLAB, Python y mBlock con lenguaje Scratch. MATLAB se determinó como la mejor opción para simulaciones debido a su entorno matemático y visual, permitiendo representaciones detalladas y análisis avanzados de datos complejos. Sin embargo, Scratch aunque intuitivo mostró limitaciones para operaciones complejas y aplicaciones que requieren bibliotecas no disponibles en su entorno. Por otro lado, mBlock ofrece la posibilidad de programar en Python, resultando una opción más viable, Python en mBlock proporciona un entorno adecuado para programación avanzada con la flexibilidad de integrar bibliotecas adicionales, asegurando una programación más robusta. Por lo tanto, se concluye que MATLAB es ideal para simulaciones detalladas, mientras que Python en mBlock es la mejor opción para la implementación, combinando la capacidad de análisis avanzado con una programación flexible y robusta.

- Se realizó las simulaciones de los algoritmos tipo insecto en entornos virtuales, cabe mencionar que se completaron exitosamente en MATLAB. Los resultados de las simulaciones en MATLAB demostraron la validez y efectividad de los algoritmos para la navegación autónoma y la evitación de obstáculos en cada tipo insecto (0, 1 y 2) en tres mapas diferentes en un entorno de simulación para esta esta propuesta.
- Se realizó el análisis del comportamiento del robot móvil en términos de tiempo y distancia recorrida para cada uno de los algoritmos tipo insecto en las simulaciones. Los resultados mostraron variaciones en el desempeño de los algoritmos en diferentes condiciones y pistas, proporcionando información para la evaluación comparativa y futuras mejoras.
- La funcionalidad de los algoritmos implementados se verificó en entornos reales. Los experimentos demostraron que los algoritmos pueden guiar al robot móvil desde el punto de partida hasta el punto de llegada de manera efectiva. Sin embargo, se encontraron inconvenientes en la ejecución de los algoritmos insecto1 y insecto2 debido a las limitaciones de los sensores ultrasónicos, los cuales presentan puntos ciegos que impiden al mBot Neo capturar correctamente el entorno y rodear obstáculos de manera eficiente.
- Se realizó una evaluación del comportamiento del robot móvil en escenarios con alteraciones en el entorno físico en tres mapas diferentes. Los resultados comparativos entre simulación y pruebas reales mostraron que ciertos algoritmos son más robustos y eficientes bajo diferentes condiciones. La evaluación autónoma se vio afectada directamente en la implementación debido a las limitaciones del entorno, como los sensores ultrasónicos y la capacidad de reacción del microcontrolador del mBot Neo. Las limitaciones incluyen que el

mBot Neo solo admite ciertos sensores y no se enlaza con otros programas más robustos que no sean en Python de Mblock.

- Se realizó el análisis comparativo entre los tres algoritmos tipo insecto, valorando su desempeño en la planeación de trayectorias. Se emitieron recomendaciones y criterios de funcionalidad basados en métricas clave como tiempo de navegación, distancia recorrida y capacidad de adaptación a entornos cambiantes. Este análisis permitió identificar que el algoritmo tipo insecto 2 destacó especialmente en entornos con obstáculos densamente distribuidos, minimizando tanto el tiempo de navegación como los errores de posición en comparación con los otros tipos. El algoritmo tipo insecto 0 mostró limitaciones debido a su tendencia a girar siempre en una dirección para esquivar obstáculos, lo que puede llevarlo a perder la orientación en ciertos tipos de pistas. Por otro lado, el algoritmo tipo insecto 1 requiere más tiempo para analizar y seleccionar la ruta óptima al rodear obstáculos, pero una vez que ha aprendido el entorno, puede ser el más efectivo debido a su capacidad para realizar movimientos más precisos y eficientes.

4.2 RECOMENDACIONES

- Es recomendable seguir profundizando en el estudio de los fundamentos matemáticos que sustentan los algoritmos tipo insecto. Esta comprensión avanzada facilita la programación y optimización de los algoritmos, asegurando su capacidad para calcular trayectorias eficientes y evitar obstáculos con precisión. Además, se sugiere aplicar estos conocimientos matemáticos en la creación de nuevas variantes de algoritmos que puedan adaptarse aún mejor a diferentes entornos y condiciones, mejorando así el rendimiento general del robot móvil mBot Neo en tareas de navegación autónoma.

- Se recomienda optar por la integración de componentes personalizados en la construcción de un robot móvil, aunque implique un mayor costo inicial, presenta numerosas ventajas. Esta estrategia permite la selección e incorporación de una gama más amplia de sensores, optimizando la capacidad de detección del entorno. La flexibilidad de los sistemas personalizados facilita la adaptación del robot a requisitos específicos de percepción y navegación. Además, esta opción favorece la implementación de soluciones avanzadas y precisas, esenciales para una detección y respuesta robustas ante cambios en el entorno
- Para trabajar con el mBot Neo, se sugiere utilizar Python para la programación debido a su capacidad para realizar acciones más complejas en comparación con Scratch. Python permite un control más detallado sobre los movimientos del robot y la integración con algoritmos más avanzados de navegación y percepción. Esto es especialmente beneficioso para proyectos que requieren decisiones autónomas más sofisticadas y adaptabilidad a diferentes entornos.
- Es recomendable elegir una plataforma de programación que no limite las operaciones matemáticas y funciones esto esencial para optimizar el desarrollo y desempeño del robot móvil. Alternativas como Python ofrecen capacidades extendidas en procesamiento numérico y algoritmos avanzados de machine learning, lo cual es fundamental para mejorar la capacidad predictiva y adaptativa del robot ante entornos dinámicos y variables.
- Si se opta por usar mBlock para este estudio, sería recomendable considerar la adición de sensores adicionales al mBot Neo, aumentar el número y tipo de sensores puede mejorar significativamente la precisión de la detección en todo su rango, reduciendo la posibilidad de errores en la periferia o puntos ciegos del robot. Es importante tener en cuenta que el mBot Neo tiene limitaciones en cuanto

a los tipos de sensores que se pueden integrar directamente, por lo que es fundamental seleccionar sensores compatibles que mejoren la funcionalidad deseada sin exceder las capacidades de procesamiento del robot, Para superar las limitaciones del mBot Neo, sería beneficioso considerar la construcción de un robot personalizado que permita la integración de una mayor variedad de sensores y actuadores, así como un procesamiento más potente.

- Para futuros proyectos, se recomienda que esta investigación sirva como una base sólida para que estudiantes y desarrolladores profundicen en el estudio y mejora de los algoritmos tipo insecto. Este enfoque no solo facilita la comprensión práctica de los principios fundamentales de la planificación de rutas en robots móviles, sino que también ofrece la oportunidad de explorar y aplicar técnicas innovadoras como el aprendizaje automático y otras tecnologías avanzadas. Al incorporar estas herramientas, los investigadores pueden expandir las capacidades de los algoritmos de navegación autónoma y optimizar su rendimiento tanto en entornos reales como simulados.

BIBLIOGRAFÍA

[1] L. a. o. Caballero Arnaldos, «Diseño y evaluación de algoritmos de planificación de rutas eficientes para flotas de vehículos,» 2017.

[2] H. Choset, «Planificación de movimiento robótico algoritmos de insectos,» 2010.

[3] G. T. T. a. B. B. Conde, «Algoritmos de enjambre,» *Revista Boliviano de Ciencias,* vol. 12, p. 23, 2016.

[4] E. Cuevas, «El algoritmo 'Artificial Bee Colony'(ABC) y su uso en el Procesamiento digital de Imágenes,» *Revista Iberoamericana de Inteligencia Artificial,* vol. 18, nº 55, p. 1–19, 2015.

[5] B. R. y. o. T. Asokan, «Un nuevo algoritmo de planificación de ruta de errores múltiples para la navegación de robots en entornos conocidos,» *Conferencia de la Región 10 de IEEE de 2016 (TENCON) ,* p. 3363–3367, 2016.

[6] J. A. G. a. A. G. Patiño, «Análisis comparativo entre el algoritmo A* y el algoritmo tipo insecto para la planeación de rutas aplicadas sobre un robot móvil con ruedas,» Guayaquil, 2022.

[7] F. Pineda-Torres, «Técnicas de slam con filtros probabilísticos: caracterización y resultados en robots móviles,» *Mundo FESC,* vol. 9, nº 18, pp. 7-15, 2019.

[8] J. A. G. a. A. G. Patiño, «Análisis comparativo entre el algoritmo A* y el algoritmo tipo insecto para la planeación de rutas aplicadas sobre un robot móvil con ruedas,» Guayaquil, 2022.

[9] V. R. B. S. J. R. G. &. O. R. S. Sotelo, «Robots móviles: Evolución y estado del arte,» *Polibits ,* pp. 12 -17, 2007.

[10] J. C. Dehesa, «Sensorizacion y control de robot moviles,» Mexico , 2015.

[11] M. A. Montiel, «ResearchGate,» [En línea]. Available: https://www.researchgate.net/figure/Figura-54-Robots-moviles-con-diversos-sistemas-de-locomocion-a-ruedas-b-orugas-c_fig3_309398090. [Último acceso: 27 Enero 2023].

[12] MecatrónicaLATAM, «MecatrónicaLATAM,» [En línea]. Available: https://www.mecatronicalatam.com/es/tutoriales/tipos-de-robots/. [Último acceso: 27 Enero 2023].

[13] l. e. s. AM990, «AM990, llega en serio.,» 2 Enero 2017. [En línea]. Available: https://am990formosa.com/robots-y-drones-expectativa-en-las-vegas-por-las-novedades-de-2017/. [Último acceso: 27 Enero 2023].

[14] K. Navarro, «Mi patente,» [En línea]. Available: https://www.mipatente.com/disenan-robot-submarino-para-mantenimiento-portuario/. [Último acceso: 27 Enero 2023].

[15] J. A. C. Meneses, «DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GENERACIÓN DE TRAYECTORIAS PARA UN ROBOT MÓVIL UTILIZANDO CONTROL ODOMÉTRICO,» Tesis para optar el Título de Ingeniero Electrónico, Lima, 2012.

[16] V. T. Mora, «Entender los diferentes tipos de locomoción de los robots,» Robótica Avanzada (L41087), Zumpango de Ocampo, 2015.

[17] L. I. G. Calandín, «Modelado Cinemático y Control de Robots Móviles con Ruedas,» Tesis Doctoral, Universidad Politécnica de Valencia, 2006.

[18] H. J. Valencia, «DISEÑO Y DESARROLLO DE ROBOTS CON LOCOMOCIÓN ESFERICA PARA SEGUIR TRAYECTORIAS PREESTABLECIDAS,» Trabajo de grado, INSTITUTO TECNOLÓGICO METROPOLITANO, 2016.

[19] A. Z. S. B. GREGOR KLANCAR, «WHEELED MOBILE ROBOTICS,» Elsevier, Ljubljana, Slovenia, 2016.

[20] M. W. M. D. S. M. B. Mgt. Junior Figueroa, Cinématica y Dinámica de robots Móvil con Ruedas, Guayaquil: CILADI, 2023.

[21] I. Acosta, «ROBOT MÓVIL PARA INVESTIGACIÓN EN ALGORITMOS DE PLANEAMIENTO DE RUTAS,» Carrera de Ingeniería en Mecatrónica, Universidad Técnica del Norte, Ibarra.

[22] E. A. G. Ríos, «Desarrollo de un algoritmo planificador de rutas con capacidad de implementación en diversas aplicaciones de la robótica móvil.,» Proyecto de grado, Universidad Tecnológica de Pereira, Pereira, 2015.

[23] C. Electrónicas, «Componentes Electrónicas,» [En línea]. Available: https://www.compelect.com.co/que-es-matlab/#:~:text=MATLAB%20es%20un%20lenguaje%20basado,natural%20de%20las%20matem%C3%A1ticas%20computacionales.

&text=Es%20el%20entorno%20inform%C3%A1tico%20m%C3%A1s,y%20el%20trabajo%20que%20realiza.. [Último acceso: 27 Febrero 2023].

[24] MathWorks, «MathWorks,» [En línea]. Available: https://la.mathworks.com/solutions/robotics.html. [Último acceso: 27 Febrero 2023].

[25] Papitek, «Papitek,» [En línea]. Available: https://papitek.es/scratch. [Último acceso: 27 Febrero 2023].

[26] M. S. Hernández, «Navegación de robots móviles utilizandolos algoritmos insectos extendidos,» Mexico, 2018.

[27] W. V. Paredes, «Robot móvil para investigación en algoritmos de planeamiento de rutas: sistema de Odometría,» Ibarra, 2018.

[28] M. Education, «Makeblock Education,» 28 Septiembre 2022. [En línea]. Available: https://education.makeblock.com/help/mbuild-quad-rgb-sensor/. [Último acceso: 12 Septiembre 2023].

[29] M. Education, «Makeblock Education,» 2 Mayo 2022. [En línea]. Available: https://education.makeblock.com/help/cyberpi-series-180-optical-encoder-motor/. [Último acceso: 12 Septiembre 2023].

[30] M. EDUCATIONAL, «MAKEBLOCK EDUCATIONAL,» 2 MAYO 2022. [En línea]. Available: https://education.makeblock.com/help/cyberpi-series-cyberpi/. [Último acceso: 19 SEPTIEMBRE 2023].

[31] M. EDUCATION, «MAKEBLOCK EDUCATION,» [En línea]. Available: https://education.makeblock.com/help/mblock-block-based-mblock-3-vs-mblock-5/. [Último acceso: 30 mayo 2024].

[32] S. Becas, «Santander Becas,» [En línea]. Available: https://www.becas-santander.com/es/blog/python-que-es.html#:~:text=Python%20es%20un%20lenguaje%20sencillo,permite%20desarrollar%20software%20sin%20l%C3%ADmites.. [Último acceso: 27 Febrero 2023].

[33] 1000MARCAS, «1000MARCAS,» [En línea]. Available: https://1000marcas.net/python-logo/. [Último acceso: 27 Febrero 2023].

[34] M. Education, «Makeblock Education,» 1 Mayo 2022. [En línea]. Available: https://education.makeblock.com/help/cyberpi-series-ms-1-5a-servo/. [Último acceso: 12 Septiembre 2023].

ANEXOS

ANEXOS A

- **Código de simulación MATLAB**

```
classdef SimulacionBugs1 < matlab.apps.AppBase


    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        TabGroup                        matlab.ui.container.TabGroup
        MAPA1Tab                        matlab.ui.container.Tab
        VelocidadenRPMEditField         matlab.ui.control.NumericEditField
        VelocidadenRPMEditFieldLabel    matlab.ui.control.Label
        RESULTADOSLabel                 matlab.ui.control.Label
        Panel_3                         matlab.ui.container.Panel
        ErrordePosicinEditField         matlab.ui.control.NumericEditField
        ErrordePosicinEditFieldLabel    matlab.ui.control.Label
        ngulodeorientacindellegadaEditField  matlab.ui.control.NumericEditField
        ngulodeorientacindellegadaEditFieldLabel  matlab.ui.control.Label
        TiempoRecorridosEditField       matlab.ui.control.NumericEditField
        TiempoRecorridosEditFieldLabel  matlab.ui.control.Label
        DistanciaTotalmEditField        matlab.ui.control.NumericEditField
        DistanciaTotalmLabel            matlab.ui.control.Label
        Panel                           matlab.ui.container.Panel
        UIAxes                          matlab.ui.control.UIAxes
        Panel_2                         matlab.ui.container.Panel
        BUG2Button                      matlab.ui.control.Button
        BUG1Button                      matlab.ui.control.Button
        BUG0Button                      matlab.ui.control.Button
        MAPA2Tab                        matlab.ui.container.Tab
        VelocidadenRPMEditField_2       matlab.ui.control.NumericEditField
        VelocidadenRPMEditField_2Label  matlab.ui.control.Label
        RESULTADOSLabel_2               matlab.ui.control.Label
        Panel_6                         matlab.ui.container.Panel
        ErrordePosicinEditField_2       matlab.ui.control.NumericEditField
        ErrordePosicinEditField_2Label  matlab.ui.control.Label
        ngulodeorientacindellegadaEditField_2  matlab.ui.control.NumericEditField
        ngulodeorientacindellegadaEditField_2Label  matlab.ui.control.Label
        TiempoRecorridosEditField_2     matlab.ui.control.NumericEditField
        TiempoRecorridosEditField_2Label  matlab.ui.control.Label
        DistanciaTotalmEditField_2      matlab.ui.control.NumericEditField
        DistanciaTotalmEditField_2Label  matlab.ui.control.Label
        Panel_5                         matlab.ui.container.Panel
        UIAxes_2                        matlab.ui.control.UIAxes
        Panel_4                         matlab.ui.container.Panel
        BUG2Button_2                    matlab.ui.control.Button
```

```
        BUG1Button_2                matlab.ui.control.Button
        BUG0Button_2                matlab.ui.control.Button
        MAPA3Tab                    matlab.ui.container.Tab
        VelocidadenRPMEditField_3       matlab.ui.control.NumericEditField
        VelocidadenRPMEditField_3Label  matlab.ui.control.Label
        RESULTADOSLabel_3               matlab.ui.control.Label
        Panel_9                     matlab.ui.container.Panel
        ErrordePosicinEditField_3       matlab.ui.control.NumericEditField
        ErrordePosicinEditField_3Label  matlab.ui.control.Label
        ngulodeorientacindellegadaEditField_3  matlab.ui.control.NumericEditField
        ngulodeorientacindellegadaEditField_3Label  matlab.ui.control.Label
        TiempoRecorridosEditField_3     matlab.ui.control.NumericEditField
        TiempoRecorridosEditField_3Label  matlab.ui.control.Label
        DistanciaTotalmEditField_3      matlab.ui.control.NumericEditField
        DistanciaTotalmEditField_3Label  matlab.ui.control.Label
        Panel_8                     matlab.ui.container.Panel
        UIAxes_3                      matlab.ui.control.UIAxes
        Panel_7                     matlab.ui.container.Panel
        BUG2Button_3                    matlab.ui.control.Button
        BUG1Button_3                    matlab.ui.control.Button
        BUG0Button_3                    matlab.ui.control.Button
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: BUG0Button
        function BUG0ButtonPushed(app, event)
             clc
            clearvars -except app

            % Capturar el dato de velocidad en RPM
            Vrmp = app.VelocidadenRPMEditField.Value;

            Ts = 0.03; % Tiempo de muestreo
            t = 0:Ts:190; % Tiempo de simulación
            r = 0.032;
            b = 0.065;
            rps = Vrmp / 60;
            Vs = (rps * (2 * pi)) * r;

            % MAPA 1
            q = [0.50; 0.30; 0]; % Posición inicial
            goal = [1.45; 1.75]; % Posición objetivo
            % Obstáculos
            obstacles = cell(1);
            obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);
            obstacles{1} = flipud([0.15 0.6; 1.15 0.6; 1.15 0.75; 0.15 0.75;0.15 0.6]);
            obstacles{2} = flipud([0.8 1.2; 1.75 1.2; 1.75 1.35; 0.95 1.35; 0.95 1.8; 0.8
1.8;0.8 1.2]);
```

```
        windowSize = 5; % Tamaño de la ventana para la media móvil
        dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
        umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
        distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
        distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
        obstacleDetected = false; % Bandera para detectar obstáculos
        obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
        closestPointToGoal = []; % Almacena el punto más cercano al objetivo
        g = [Vs, 5];
        ePhi = 0;
        dGoal = 0;
        rodeo = 1;
        posicionAnterior=0
        distanciaTotal=0
        epsilon = 0.05;
        ob = 0;
        obst = [];
        for i = 1:length(obstacles)
            obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
        end

        % Generar la primera figura en UIAxes
        cla(app.UIAxes);
        hold(app.UIAxes, 'on');
        for i = 1:length(obstacles)
            if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
                plot(app.UIAxes, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
            else
                patch(app.UIAxes, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
            end
        end
        xlabel(app.UIAxes, 'x'); ylabel(app.UIAxes, 'y');

        ref = OMarker1(app.UIAxes); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
        wmr0 = OWmr1(app.UIAxes); wmr0.setPose(q); wmr0.setTxt('Start', 0.05, pi/4,
'rb'); wmr0.showWmr('r:');
        wmr = OWmr1(app.UIAxes); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

        % Generar la segunda figura en UIAxes2
        OFig1.pause(Ts);
        obstacleAvoided = false;
        phi = pi/9; % Ángulo de giro controlado
        phiObst = 0;
        umbralLlegada = 0.05; % Umbral de llegada al objetivo
        distanciaTotal = 0; % Inicia el contador de distancia total recorrida
```

```
        posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la
distancia
        tiempoInicial = tic; % Guarda el tiempo inicial

        for k = 1:length(t)
            [dObst, ~, z] = nearestSegment(q(1:2).', obst);
            dObst_buffer = [dObst_buffer(2:end), dObst];
            dObst_filtered = mean(dObst_buffer);
            dObst = dObst_filtered;
            if dObst > distanciaMinima
                phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));
            end
            if dObst > distanciaMinima
                phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
                ePhi = wrapToPi(phiRef - q(3));
                dGoal = sqrt(sum((goal - q(1:2)).^2));
                g = [Vs, 1];
                obstacleAvoided = false;
            elseif dObst <= distanciaMinima
                ePhi = wrapToPi(pi + phiObst - q(3));
                dGoal = sqrt(sum((goal - q(1:2)).^2));
                g = [Vs, 10];
            end

            v = g(1) * abs(cos(ePhi));
            w = g(2) * ePhi;
            v = min([v, 0.3]);

            vx = v * cos(q(3));
            vy = v * sin(q(3));
            wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w);
            wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w);
            wdRMP = wd * 60 / (2 * pi);
            wiRMP = wi * 60 / (2 * pi);

            dq = [vx; vy; w];
            noise = 0.00;
            q = q + Ts * dq + randn(3,1) * noise;

            distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
            posicionAnterior = q(1:2);

            wmr.setPose(q);
            OFig1.pause(Ts);

            if dGoal < umbralLlegada
                break;
            end
        end
```

```
            tiempoRecorrido = toc(tiempoInicial);
            anguloOrientacionLlegada = wrapToPi(q(3));
            errorPosicion = sqrt(sum((goal - q(1:2)).^2));

            fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
            fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
            fprintf('Ángulo de orientación de llegada: %.2f rad\n',
anguloOrientacionLlegada);
            fprintf('Error de Posición: %.2f m\n', errorPosicion);

            % Colocar los resultados en los campos de texto correspondientes
            app.DistanciaTotalmEditField.Value = distanciaTotal;
            app.TiempoRecorridosEditField.Value = tiempoRecorrido;
            app.ngulodeorientacindellegadaEditField.Value = anguloOrientacionLlegada;
            app.ErrordePosicinEditField.Value = errorPosicion;
        end

        % Button pushed function: BUG1Button
        function BUG1ButtonPushed(app, event)
          clc
            clearvars -except app

            % Capturar el dato de velocidad en RPM
            Vrmp = app.VelocidadenRPMEditField.Value;

Ts = 0.03; % Tiempo de muestreo
t = 0:Ts:190; % Tiempo de simulación
r = 0.032;
b = 0.065;

rps = Vrmp / 60;
Vs = (rps * (2 * pi)) * r;

  % MAPA 2

% q = [1.5; 0.05; 0]; % Posición inicial
%  goal = [0.80; 1.1]; % Posición objetivo
% obstacles = cell(1);
% obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);
%
% obstacles{2} = flipud([0.40 0.40; 1.80 0.40; 1.80 1.40 ; 0.2 1.4; 0.20 0.80;0.9 0.8;0.9
0.95;0.35 0.95; 0.35 1.25;1.65 1.25; 1.65 0.55;0.4 0.55]);

% % MAPA 1
q = [0.50; 0.30; 0]; % Posición inicial
goal = [1.45; 1.75]; % Posición objetivo
% Obstáculos
obstacles = cell(1);
%obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);
obstacles{1} = flipud([0.15 0.6; 1.15 0.6; 1.15 0.75; 0.15 0.75;0.15 0.6]);
```

```
obstacles{2} = flipud([0.8 1.2; 1.75 1.2; 1.75 1.35; 0.95 1.35; 0.95 1.8; 0.8 1.8;0.8 1.2]);

windowSize = 5; % Tamaño de la ventana para la media móvil
dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
obstacleDetected = false; % Bandera para detectar obstáculos
obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
closestPointToGoal = []; % Almacena el punto más cercano al objetivo
g = [Vs, 5];
ePhi=0
dGoal=0
rodeo = 1;
epsilon = 0.05;
ob = 0;
obst = [];
          cla(app.UIAxes);
          hold(app.UIAxes, 'on');
for i = 1:length(obstacles)
   obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
end

% Configuración de la figura y marcadores
         for i = 1:length(obstacles)
             if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
                plot(app.UIAxes, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
             else
                patch(app.UIAxes, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
             end
          end

estado = 0;
estado2 = 0;
guardar = 1;

dp = inf;  % Inicializa la distancia mínima a infinito
qm = [];
q_H_i = [];

% Inicialización de variables de seguimiento
distanciaTotal = 0; % Inicia el contador de distancia total recorrida
posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la distancia
tiempoInicial = tic; % Guarda el tiempo inicial
xlabel(app.UIAxes, 'x'); ylabel(app.UIAxes, 'y');

ref = OMarker1(app.UIAxes); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
```

```
wmr0 = OWmr1(app.UIAxes); wmr0.setPose(q); wmr0.setTxt('Start', 0.05, pi/4, 'rb');
wmr0.showWmr('r:');
wmr = OWmr1(app.UIAxes); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

OFig1.pause(Ts);
 obstacleDetected = false;
f=0
ff=0
phi = pi/9; % Ángulo de giro controlado
phiObst=0;
umbralLlegada = 0.05; % Umbral de llegada al objetivo
rodeo=0
for k = 1:length(t)
    [dObst, ~, z] = nearestSegment(q(1:2).', obst);
    % Actualizar el buffer del filtro de media móvil

    phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));

if dObst < umbralObstaculo
    if ~obstacleDetected
        obstacleDetected = true;
        if guardar == 1
            q_H_i = q(1:2);  % Almacena solo la primera vez que se detecta un obstáculo
            guardar = 0;
             % Evita sobrescribir q_H_i mientras se rodea el mismo obstáculo
       rodeo=1
        end
    end
else
    if obstacleDetected==1 && dObst>0.20
        obstacleDetected = false;
        guardar = 1;  % Permite que q_H_i se actualice si se detecta un nuevo obstáculo

    end
end



switch estado

    case 0
 if dObst > umbralObstaculo
        phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
        ePhi = wrapToPi(phiRef - q(3));
        dGoal = sqrt(sum((goal - q(1:2)).^2));
        g = [Vs, 5]; % Ganancias de control
 else

estado =1;
```

```
    end

   case 1



      if dObst < umbralObstaculo && obstacleDetected == true % Conducir alrededor
del obstáculo
            phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [0.2, 5]; % Ganancias de control
         else
            if dObst > umbralObstaculo
               estado = 2;
            end
         end

   case 2

         ePhi = wrapToPi(pi+phiObst-q(3));
         dGoal = sqrt(sum((goal - q(1:2)).^2));
          g = [Vs, 5];
            if dObst > 0.15
               estado = 3;
            end

   case 3


          if dObst > umbralObstaculo  % Conducir hacia la meta
            phiRef = atan2(q_H_i(2)-q(2), q_H_i(1)-q(1))
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((goal-q(1:2)).^2));
            g = [Vs, 5]; % Ganancias de control
         else % Conducir alrededor del obstáculo
            % Conducir hacia el punto más cercano en la ruta original
              phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 5]; % Ganancias de control
            ff=sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2);
      if sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2) < 0.09
               estado = 4; % Cambiar al estado de conducción hacia la meta
      rodeo=0;
      end
        end
   case 4

      if dObst > umbralObstaculo  % Conducir hacia la meta
```

```
            phiRef = atan2(qm(2)-q(2), qm(1)-q(1));
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((qm-q(1:2)).^2));
            g = [Vs, 5]; % Ganancias de control
        else % Conducir alrededor del obstáculo
            phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 5]; % Ganancias de control
            f=sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2);
             if sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2) < 0.1
               estado = 5 % Cambiar al estado de conducción hacia la meta

               guardar=1;
               rodeo=0;

            end
      end

   case 5

      phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
      ePhi = wrapToPi(phiRef - q(3));
      dGoal = sqrt(sum((goal - q(1:2)).^2));
      g = [Vs, 5]; % Ganancias de control
      if rodeo==1
estado=0
      end


end


if rodeo==1

distanciaActual = norm(q(1:2) - goal);
            % Actualiza la distancia mínima y el punto si es necesario
            if distanciaActual < dp
               dp = distanciaActual;
               qm = q(1:2);
            end

end



   v = g(1) * abs(cos(ePhi));
   w = g(2) * ePhi;
   v = min([v, 0.3]); % Limitar la velocidad lineal angular máxima del robot de 3 rad/s
```

```
    vx = v * cos(q(3)); % Velocidad lineal en x del robot
    vy = v * sin(q(3)); % Velocidad lineal en y del robot
    wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w); % Velocidad angular de la
rueda derecha en rad/s
    wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w); % Velocidad angular de la rueda
izquierda en rad/s
    wdRMP = wd * 60 / (2 * pi); % Velocidad angular de la rueda derecha en rpm
    wiRMP = wi * 60 / (2 * pi); % Velocidad angular de la rueda izquierda en rpm

    dq = [vx; vy; w]; % Vector de derivadas del vector de postura
    noise = 0.00; % Para experimentar añadiendo ruido (ejemplo 0.001)
    q = q + Ts * dq + randn(3,1) * noise; % Integración de Euler

    distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
    posicionAnterior = q(1:2);

    % Simulación del movimiento del robot
    wmr.setPose(q);

    OFig1.pause(Ts);

    % Verificar si ha llegado al objetivo
    if dGoal < umbralLlegada
        break;
    end
end

% Cálculos finales
tiempoRecorrido = toc(tiempoInicial); % Tiempo transcurrido
anguloOrientacionLlegada = wrapToPi(q(3)); % Ángulo de orientación de llegada
errorPosicion = sqrt(sum((goal - q(1:2)).^2)); % Error de posición

% Mostrar resultados
fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
fprintf('Ángulo de orientación de llegada: %.2f rad\n', anguloOrientacionLlegada);
fprintf('Error de Posición: %.2f m\n', errorPosicion);
        % Colocar los resultados en los campos de texto correspondientes
            app.DistanciaTotalmEditField.Value = distanciaTotal;
            app.TiempoRecorridosEditField.Value = tiempoRecorrido;
            app.ngulodeorientacindellegadaEditField.Value = anguloOrientacionLlegada;
            app.ErrordePosicinEditField.Value = errorPosicion;
        end

        % Button pushed function: BUG2Button
        function BUG2ButtonPushed(app, event)

  clc
            clearvars -except app
```

```
        % Capturar el dato de velocidad en RPM
        Vrmp = app.VelocidadenRPMEditField.Value;

        Ts = 0.03; % Tiempo de muestreo
        t = 0:Ts:190; % Tiempo de simulación
        r = 0.032;
        b = 0.065;
        rps = Vrmp / 60;
        Vs = (rps * (2 * pi)) * r;

        % MAPA 1
        q = [0.50; 0.30; 0]; % Posición inicial
        goal = [1.45; 1.75]; % Posición objetivo

q1 = [0.50; 0.30; 0]
        % Obstáculos
        obstacles = cell(1);
        obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);
        obstacles{1} = flipud([0.15 0.6; 1.15 0.6; 1.15 0.75; 0.15 0.75;0.15 0.6]);
        obstacles{2} = flipud([0.8 1.2; 1.75 1.2; 1.75 1.35; 0.95 1.35; 0.95 1.8; 0.8
1.8;0.8 1.2]);

        windowSize = 5; % Tamaño de la ventana para la media móvil
        dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
        umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
        distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
        distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
        obstacleDetected = false; % Bandera para detectar obstáculos
        obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
        closestPointToGoal = []; % Almacena el punto más cercano al objetivo
        g = [Vs, 5];
        ePhi = 0;
        dGoal = 0;
        rodeo = 1;
        posicionAnterior=0
        distanciaTotal=0
        epsilon = 0.05;
        ob = 0;
        obst = [];
        for i = 1:length(obstacles)
            obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
        end

        % Generar la primera figura en UIAxes
        cla(app.UIAxes);
        hold(app.UIAxes, 'on');

        for i = 1:length(obstacles)
            if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
```

```
                plot(app.UIAxes, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
            else
                patch(app.UIAxes, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
            end
        end
        xlabel(app.UIAxes, 'x'); ylabel(app.UIAxes, 'y');
 plot(app.UIAxes,[q(1), goal(1)], [q(2), goal(2)],'color', 'black', 'LineWidth', 1);

        ref = OMarker1(app.UIAxes); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
        wmr0 = OWmr1(app.UIAxes); wmr0.setPose(q); wmr0.setTxt('Start', 0.05, pi/4,
'rb'); wmr0.showWmr('r:');
        wmr = OWmr1(app.UIAxes); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

        % Generar la segunda figura en UIAxes2
        OFig1.pause(Ts);
        obstacleAvoided = false;
        phi = pi/9; % Ángulo de giro controlado
        phiObst = 0;
        umbralLlegada = 0.05; % Umbral de llegada al objetivo
        distanciaTotal = 0; % Inicia el contador de distancia total recorrida
        posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la
distancia
        tiempoInicial = tic; % Guarda el tiempo inicial
estado = 0;
estado2 = 0;
guardar = 1;

dp = inf;  % Inicializa la distancia mínima a infinito
qm = [];
q_H_i = [];
        for k = 1:length(t)
  [dObst, ~, z] = nearestSegment(q(1:2).', obst);
   if dObst>umbralObstaculo
   phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));
   end


if dObst < umbralObstaculo
   if ~obstacleDetected
      obstacleDetected = true;
      if guardar == 1
         q_H_i = q(1:2);  % Almacena solo la primera vez que se detecta un obstáculo
         guardar = 0;
           estado=1;% Evita sobrescribir q_H_i mientras se rodea el mismo obstáculo
      rodeo=1
      end
```

```
    end
else
    if obstacleDetected==1 && dObst >0.15
        obstacleDetected = false;
        guardar = 1;  % Permite que q_H_i se actualice si se detecta un nuevo obstáculo
rodeo=0
    end
end


   d_line_m = abs((goal(1) - q1(1))*(q1(2) - q(2)) - (q1(1) - q(1))*(goal(2) - q1(2)))
/sqrt((goal(1) - q1(1))^2 + (goal(2) - q1(2))^2)

    switch estado
        case 0
            if dObst > umbralObstaculo  % Conducir hacia la meta
                phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
                ePhi = wrapToPi(phiRef - q(3));
                dGoal = sqrt(sum((goal-q(1:2)).^2));
                g = [Vs, 10]; % Ganancias de control
            else % Conducir alrededor del obstáculo
                estado = 1;
            end

        case 1



            switch estado2


                case 0
   if dObst < umbralObstaculo && obstacleDetected == true % Conducir alrededor del
obstáculo
                phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
                ePhi = wrapToPi(phiRef-q(3));
                g = [Vs, 5]; % Ganancias de control
            else
                if dObst > umbralObstaculo
                    estado2 = 1;
                end
            end

                case 1
  if dObst > umbralObstaculo  % Conducir hacia el punto más cercano en la ruta original
             %   phiRef = atan2(q_H_i(2)-q(2), q_H_i(1)-q(1));
phiRef = phiObst +pi
ePhi = wrapToPi(phiRef - q(3));
                dqh = sqrt(sum((q_H_i-q(1:2)).^2));
```

```
            g = [Vs, 10]; % Ganancias de control
          else % Conducir alrededor del obstáculo
            estado2 = 2;
          end
            case 2
if dObst < umbralObstaculo % Conducir alrededor del obstáculo
            phiRef = wrapToPi(phiObst);  % Agregar pi para ir siempre a la izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 5]; % Ganancias de control
                ff= sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2)
       if sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2) < 0.09
              estado2 = 1; % Cambiar al estado de conducción hacia la meta
            end
else

           estado2=3;

          end

            case 3

          if dObst > umbralObstaculo  % Conducir hacia la meta
            phiRef = atan2(q_H_i(2)-q(2), q_H_i(1)-q(1));
             %   phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
            ePhi = wrapToPi(phiObst+pi - q(3));
            dGoal = sqrt(sum((goal-q(1:2)).^2));
            g = [Vs, 15]; % Ganancias de control
          else % Conducir alrededor del obstáculo
            % Conducir hacia el punto más cercano en la ruta original
              phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 5]; % Ganancias de control
            ff= sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2);
       if d_line_m< 0.01
       estado = 2; % Cambiar al estado de conducción hacia la meta
        rodeo=0
    estado2=0
       end
          end
          end

      case 2


            phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((goal-q(1:2)).^2));
            g = [Vs, 15]; % Ganancias de control
```

```
      if d_line_m<0.05 && dObst >umbralObstaculo
         estado=0
         end

          end

    v = g(1) * abs(cos(ePhi));
    w = g(2) * ePhi;
    v = min([v, 0.3]); % Limitar la velocidad lineal angular máxima del robot de 3 rad/s

    vx = v * cos(q(3)); % Velocidad lineal en x del robot
    vy = v * sin(q(3)); % Velocidad lineal en y del robot
    wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w); % Velocidad angular de la
rueda derecha en rad/s
    wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w); % Velocidad angular de la rueda
izquierda en rad/s
    wdRMP = wd * 60 / (2 * pi); % Velocidad angular de la rueda derecha en rpm
    wiRMP = wi * 60 / (2 * pi); % Velocidad angular de la rueda izquierda en rpm

    dq = [vx; vy; w]; % Vector de derivadas del vector de postura
    noise = 0.00; % Para experimentar añadiendo ruido (ejemplo 0.001)
    q = q + Ts * dq + randn(3,1) * noise; % Integración de Euler

    distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
    posicionAnterior = q(1:2);

    % Simulación del movimiento del robot
    wmr.setPose(q);

    OFig1.pause(Ts);

    % Verificar si ha llegado al objetivo
    if dGoal < umbralLlegada
       break;
    end
end



% Cálculos finales
tiempoRecorrido = toc(tiempoInicial); % Tiempo transcurrido
anguloOrientacionLlegada = wrapToPi(q(3)); % Ángulo de orientación de llegada
errorPosicion = sqrt(sum((goal - q(1:2)).^2)); % Error de posición

          tiempoRecorrido = toc(tiempoInicial);
          anguloOrientacionLlegada = wrapToPi(q(3));
          errorPosicion = sqrt(sum((goal - q(1:2)).^2));

          fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
          fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
```

```
        fprintf('Ángulo de orientación de llegada: %.2f rad\n',
anguloOrientacionLlegada);
        fprintf('Error de Posición: %.2f m\n', errorPosicion);

        % Colocar los resultados en los campos de texto correspondientes
        app.DistanciaTotalmEditField.Value = distanciaTotal;
        app.TiempoRecorridosEditField.Value = tiempoRecorrido;
        app.ngulodeorientacindellegadaEditField.Value = anguloOrientacionLlegada;
        app.ErrordePosicinEditField.Value = errorPosicion;




    end

    % Button pushed function: BUG0Button_2
    function BUG0Button_2Pushed(app, event)
            clc
        clearvars -except app

        % Capturar el dato de velocidad en RPM
        Vrmp = app.VelocidadenRPMEditField_2.Value;

        Ts = 0.03; % Tiempo de muestreo
        t = 0:Ts:190; % Tiempo de simulación
        r = 0.032;
        b = 0.065;
        rps = Vrmp / 60;
        Vs = (rps * (2 * pi)) * r;


   % Obstáculos
q = [1.5; 0.1; 0]; % Posición inicial
 goal = [0.80; 1.1]; % Posición objetivo
obstacles = cell(1);
obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);

obstacles{2} = flipud([0.40 0.40; 1.80 0.40; 1.80 1.40 ; 0.2 1.4; 0.20 0.80;0.9 0.8;0.9
0.95;0.35 0.95; 0.35 1.25;1.65 1.25; 1.65 0.55;0.4 0.55]);
  windowSize = 5; % Tamaño de la ventana para la media móvil
        dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
        umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
        distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
        distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
        obstacleDetected = false; % Bandera para detectar obstáculos
        obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
        closestPointToGoal = []; % Almacena el punto más cercano al objetivo
        g = [Vs, 5];
        ePhi = 0;
        dGoal = 0;
```

```
        rodeo = 1;
        posicionAnterior=0
        distanciaTotal=0
        epsilon = 0.05;
        ob = 0;
        obst = [];
        for i = 1:length(obstacles)
            obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
        end

        % Generar la primera figura en UIAxes
        cla(app.UIAxes_2);
        hold(app.UIAxes_2, 'on');
        for i = 1:length(obstacles)
            if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
                plot(app.UIAxes_2, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
            else
                patch(app.UIAxes_2, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
            end
        end
        xlabel(app.UIAxes_2, 'x'); ylabel(app.UIAxes_2, 'y');

        ref = OMarker1(app.UIAxes_2); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
        wmr0 = OWmr1(app.UIAxes_2); wmr0.setPose(q); wmr0.setTxt('Start', 0.05,
pi/4, 'rb'); wmr0.showWmr('r:');
        wmr = OWmr1(app.UIAxes_2); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

        % Generar la segunda figura en UIAxes2
        OFig1.pause(Ts);

        phi = pi/9; % Ángulo de giro controlado
        phiObst = 0;
        umbralLlegada = 0.05; % Umbral de llegada al objetivo
        distanciaTotal = 0; % Inicia el contador de distancia total recorrida
        posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la
distancia
        tiempoInicial = tic; % Guarda el tiempo inicial

        for k = 1:length(t)
            [dObst, ~, z] = nearestSegment(q(1:2).', obst);
            dObst_buffer = [dObst_buffer(2:end), dObst];
            dObst_filtered = mean(dObst_buffer);
            dObst = dObst_filtered;
            if dObst > distanciaMinima
                phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));
            end
```

```
            if dObst > distanciaMinima
                phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
                ePhi = wrapToPi(phiRef - q(3));
                dGoal = sqrt(sum((goal - q(1:2)).^2));
                g = [Vs, 1];
                obstacleAvoided = false;
            elseif dObst <= distanciaMinima
                ePhi = wrapToPi(pi + phiObst - q(3));
                dGoal = sqrt(sum((goal - q(1:2)).^2));
                g = [Vs, 10];
            end

            v = g(1) * abs(cos(ePhi));
            w = g(2) * ePhi;
            v = min([v, 0.3]);

            vx = v * cos(q(3));
            vy = v * sin(q(3));
            wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w);
            wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w);
            wdRMP = wd * 60 / (2 * pi);
            wiRMP = wi * 60 / (2 * pi);

            dq = [vx; vy; w];
            noise = 0.00;
            q = q + Ts * dq + randn(3,1) * noise;

            distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
            posicionAnterior = q(1:2);

            wmr.setPose(q);
            OFig1.pause(Ts);

            if dGoal < umbralLlegada
                break;
            end
        end

        tiempoRecorrido = toc(tiempoInicial);
        anguloOrientacionLlegada = wrapToPi(q(3));
        errorPosicion = sqrt(sum((goal - q(1:2)).^2));

        fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
        fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
        fprintf('Ángulo de orientación de llegada: %.2f rad\n',
anguloOrientacionLlegada);
        fprintf('Error de Posición: %.2f m\n', errorPosicion);
app.DistanciaTotalmEditField_2.Value = distanciaTotal;
        app.TiempoRecorridosEditField_2.Value = tiempoRecorrido;
        app.ngulodeorientacindellegadaEditField_2.Value = anguloOrientacionLlegada;
```

```
            app.ErrordePosicinEditField_2.Value = errorPosicion;
        end

        % Button pushed function: BUG1Button_2
        function BUG1Button_2Pushed(app, event)
            clc
            clearvars -except app

            % Capturar el dato de velocidad en RPM
            Vrmp = app.VelocidadenRPMEditField_2.Value;

Ts = 0.03; % Tiempo de muestreo
t = 0:Ts:400; % Tiempo de simulación
r = 0.032;
b = 0.065;

rps = Vrmp / 60;
Vs = (rps * (2 * pi)) * r;

  % MAPA 2

q = [1.5; 0.1; 0]; % Posición inicial
 goal = [0.80; 1.1]; % Posición objetivo
obstacles = cell(1);
obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);

obstacles{2} = flipud([0.40 0.40; 1.80 0.40; 1.80 1.40 ; 0.2 1.4; 0.20 0.80;0.9 0.8;0.9
0.95;0.35 0.95; 0.35 1.25;1.65 1.25; 1.65 0.55;0.4 0.55]);

% % MAPA 1
% q = [0.50; 0.30; 0]; % Posición inicial
% goal = [1.45; 1.75]; % Posición objetivo
% % Obstáculos
% obstacles = cell(1);
% %obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);
% obstacles{1} = flipud([0.15 0.6; 1.15 0.6; 1.15 0.75; 0.15 0.75;0.15 0.6]);
% obstacles{2} = flipud([0.8 1.2; 1.75 1.2; 1.75 1.35; 0.95 1.35; 0.95 1.8; 0.8 1.8;0.8
1.2]);

windowSize = 5; % Tamaño de la ventana para la media móvil
dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
obstacleDetected = false; % Bandera para detectar obstáculos
obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
closestPointToGoal = []; % Almacena el punto más cercano al objetivo
g = [Vs, 5];
ePhi=0
dGoal=0
```

```
rodeo = 1;
epsilon = 0.05;
ob = 0;
obst = [];
          cla(app.UIAxes_2);
          hold(app.UIAxes_2, 'on');
for i = 1:length(obstacles)
   obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
end

% Configuración de la figura y marcadores
         for i = 1:length(obstacles)
             if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
                plot(app.UIAxes_2, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
             else
                patch(app.UIAxes_2, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
             end
          end

estado = 0;
estado2 = 0;
guardar = 1;

dp = inf;  % Inicializa la distancia mínima a infinito
qm = [];
q_H_i = [];

% Inicialización de variables de seguimiento
distanciaTotal = 0; % Inicia el contador de distancia total recorrida
posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la distancia
tiempoInicial = tic; % Guarda el tiempo inicial
xlabel(app.UIAxes_2, 'x'); ylabel(app.UIAxes_2, 'y');

ref = OMarker1(app.UIAxes_2); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
wmr0 = OWmr1(app.UIAxes_2); wmr0.setPose(q); wmr0.setTxt('Start', 0.05, pi/4, 'rb');
wmr0.showWmr('r:');
wmr = OWmr1(app.UIAxes_2); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

OFig1.pause(Ts);
 obstacleDetected = false;
f=0
ff=0
phi = pi/9; % Ángulo de giro controlado
phiObst=0;
umbralLlegada = 0.05; % Umbral de llegada al objetivo
rodeo=0
```

```
for k = 1:length(t)
   [dObst, ~, z] = nearestSegment(q(1:2).', obst);
   % Actualizar el buffer del filtro de media móvil

   phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));

if dObst < umbralObstaculo
   if ~obstacleDetected
      obstacleDetected = true;
      if guardar == 1
         q_H_i = q(1:2);  % Almacena solo la primera vez que se detecta un obstáculo
         guardar = 0;
          % Evita sobrescribir q_H_i mientras se rodea el mismo obstáculo
      rodeo=1
      end
   end
else
   if obstacleDetected==1 && dObst>0.20
      obstacleDetected = false;
      guardar = 1;  % Permite que q_H_i se actualice si se detecta un nuevo obstáculo

   end
end




switch estado

   case 0
 if dObst > umbralObstaculo
      phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
      ePhi = wrapToPi(phiRef - q(3));
      dGoal = sqrt(sum((goal - q(1:2)).^2));
      g = [Vs, 5]; % Ganancias de control
 else

estado =1;
    end

   case 1



      if dObst < umbralObstaculo && obstacleDetected == true % Conducir alrededor
del obstáculo
            phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
```

```
            g = [Vs, 5]; % Ganancias de control
        else
            if dObst > umbralObstaculo
                estado = 2;
            end
        end

    case 2

        ePhi = wrapToPi(pi+phiObst-q(3));
        dGoal = sqrt(sum((goal - q(1:2)).^2));
         g = [Vs, 5];
            if dObst > 0.15
                estado = 3;
            end

    case 3


                if dObst > 0.1  % Conducir hacia la meta
            phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((goal-q(1:2)).^2));
            g = [Vs, 15]; % Ganancias de control
        else % Conducir alrededor del obstáculo
            % Conducir hacia el punto más cercano en la ruta original
              phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 10]; % Ganancias de control
            ff= sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2);

      if sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2) < 0.09
                estado = 4; % Cambiar al estado de conducción hacia la meta
      rodeo=0;
      end
         end
    case 4

      if dObst > umbralObstaculo  % Conducir hacia la meta
            phiRef = atan2(qm(2)-q(2), qm(1)-q(1));
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((qm-q(1:2)).^2));
            g = [Vs, 5]; % Ganancias de control
        else % Conducir alrededor del obstáculo
                phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 5]; % Ganancias de control
            f=sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2);
```

```
            if sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2) < 0.1
               estado = 5 % Cambiar al estado de conducción hacia la meta

               guardar=1;
               rodeo=0;

            end
        end

    case 5

        phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
        ePhi = wrapToPi(phiRef - q(3));
        dGoal = sqrt(sum((goal - q(1:2)).^2));
        g = [Vs, 5]; % Ganancias de control
        if rodeo==1
estado=0
        end


end


if rodeo==1

distanciaActual = norm(q(1:2) - goal);
            % Actualiza la distancia mínima y el punto si es necesario
            if distanciaActual < dp
               dp = distanciaActual;
               qm = q(1:2);
            end

end



    v = g(1) * abs(cos(ePhi));
    w = g(2) * ePhi;
    v = min([v, 0.3]); % Limitar la velocidad lineal angular máxima del robot de 3 rad/s

    vx = v * cos(q(3)); % Velocidad lineal en x del robot
    vy = v * sin(q(3)); % Velocidad lineal en y del robot
    wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w); % Velocidad angular de la
rueda derecha en rad/s
    wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w); % Velocidad angular de la rueda
izquierda en rad/s
    wdRMP = wd * 60 / (2 * pi); % Velocidad angular de la rueda derecha en rpm
    wiRMP = wi * 60 / (2 * pi); % Velocidad angular de la rueda izquierda en rpm

    dq = [vx; vy; w]; % Vector de derivadas del vector de postura
```

```
    noise = 0.00; % Para experimentar añadiendo ruido (ejemplo 0.001)
    q = q + Ts * dq + randn(3,1) * noise; % Integración de Euler

    distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
    posicionAnterior = q(1:2);

    % Simulación del movimiento del robot
    wmr.setPose(q);

    OFig1.pause(Ts);

    % Verificar si ha llegado al objetivo
    if dGoal < umbralLlegada
        break;
    end
end

% Cálculos finales
tiempoRecorrido = toc(tiempoInicial); % Tiempo transcurrido
anguloOrientacionLlegada = wrapToPi(q(3)); % Ángulo de orientación de llegada
errorPosicion = sqrt(sum((goal - q(1:2)).^2)); % Error de posición

% Mostrar resultados
fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
fprintf('Ángulo de orientación de llegada: %.2f rad\n', anguloOrientacionLlegada);
fprintf('Error de Posición: %.2f m\n', errorPosicion);
        % Colocar los resultados en los campos de texto correspondientes
            app.DistanciaTotalmEditField_2.Value = distanciaTotal;
            app.TiempoRecorridosEditField_2.Value = tiempoRecorrido;
            app.ngulodeorientacindellegadaEditField_2.Value = anguloOrientacionLlegada;
            app.ErrordePosicinEditField_2.Value = errorPosicion;
        end

        % Button pushed function: BUG2Button_2
        function BUG2Button_2Pushed(app, event)
            clc
            clearvars -except app

            % Capturar el dato de velocidad en RPM
            Vrmp = app.VelocidadenRPMEditField_2.Value;

            Ts = 0.03; % Tiempo de muestreo
            t = 0:Ts:190; % Tiempo de simulación
            r = 0.032;
            b = 0.065;
            rps = Vrmp / 60;
            Vs = (rps * (2 * pi)) * r;

            % MAPA 1
```

```
    q = [1.5; 0.1; 0]; % Posición inicial
    q1= q;
 goal = [0.9; 1]; % Posición objetivo
obstacles = cell(1);
obstacles{1} = flipud([-1 -1; 3 -1; 3 3; -1 3]);

obstacles{2} = flipud([0.40 0.40; 1.80 0.40; 1.80 1.40 ; 0.2 1.4; 0.20 0.80;0.9 0.8;0.9
0.95;0.35 0.95; 0.35 1.25;1.65 1.25; 1.65 0.55;0.4 0.55]);

  q = [1.5; 0.05; 0]; % Posición inicial
        % Obstáculos

        windowSize = 5; % Tamaño de la ventana para la media móvil
        dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
        umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
        distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
        distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
        obstacleDetected = false; % Bandera para detectar obstáculos
        obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
        closestPointToGoal = []; % Almacena el punto más cercano al objetivo
        g = [Vs, 5];
        ePhi = 0;
        dGoal = 0;
        rodeo = 1;
        posicionAnterior=0
        distanciaTotal=0
        epsilon = 0.05;
        ob = 0;
        obst = [];
        for i = 1:length(obstacles)
            obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
        end

        % Generar la primera figura en UIAxes
        cla(app.UIAxes_2);
        hold(app.UIAxes_2, 'on');

        for i = 1:length(obstacles)
            if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
                plot(app.UIAxes_2, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
            else
                patch(app.UIAxes_2, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
            end
        end
        xlabel(app.UIAxes_2, 'x'); ylabel(app.UIAxes_2, 'y');
 plot(app.UIAxes_2,[q(1), goal(1)], [q(2), goal(2)],'color', 'black', 'LineWidth', 1);
```

```
        ref = OMarker1(app.UIAxes_2); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
        wmr0 = OWmr1(app.UIAxes_2); wmr0.setPose(q); wmr0.setTxt('Start', 0.05,
pi/4, 'rb'); wmr0.showWmr('r:');
        wmr = OWmr1(app.UIAxes_2); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

        % Generar la segunda figura en UIAxes2
        OFig1.pause(Ts);
        obstacleAvoided = false;
        phi = pi/9; % Ángulo de giro controlado
        phiObst = 0;
        umbralLlegada = 0.05; % Umbral de llegada al objetivo
        distanciaTotal = 0; % Inicia el contador de distancia total recorrida
        posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la
distancia
        tiempoInicial = tic; % Guarda el tiempo inicial
estado = 0;
estado2 = 0;
guardar = 1;

dp = inf;  % Inicializa la distancia mínima a infinito
qm = [];
q_H_i = [];
        for k = 1:length(t)
 % Distancia al obstáculo más cercano y orientación del segmento
   [dObst, ~, z] = nearestSegment(q(1:2).', obst);
   if dObst>umbralObstaculo
   phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));
   end


if dObst < umbralObstaculo
   if ~obstacleDetected
      obstacleDetected = true;
      if guardar == 1
         q_H_i = q(1:2);  % Almacena solo la primera vez que se detecta un obstáculo
         guardar = 0;
          estado=1;% Evita sobrescribir q_H_i mientras se rodea el mismo obstáculo
      end
   end
else
   if obstacleDetected==1 && dObst >0.15
      obstacleDetected = false;
      guardar = 1;  % Permite que q_H_i se actualice si se detecta un nuevo obstáculo

   end
end
```

```
  d_line_m = abs((goal(1) - q1(1))*(q1(2) - q(2)) - (q1(1) - q(1))*(goal(2) - q1(2)))
/sqrt((goal(1) - q1(1))^2 + (goal(2) - q1(2))^2)

   switch estado
      case 0
         if dObst > umbralObstaculo  % Conducir hacia la meta
            phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((goal-q(1:2)).^2));
             g = [Vs, 10]; % Ganancias de control
         else % Conducir alrededor del obstáculo
            estado = 1;
         end

      case 1



         switch estado2


            case 0
  if dObst < umbralObstaculo && obstacleDetected == true % Conducir alrededor del
obstáculo
            phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 10]; % Ganancias de control
         else
            if dObst > umbralObstaculo
               estado2 = 1;
            end
         end

            case 1
 if dObst > umbralObstaculo  % Conducir hacia el punto más cercano en la ruta original
          %   phiRef = atan2(q_H_i(2)-q(2), q_H_i(1)-q(1));
            ePhi = wrapToPi(pi+phiObst - q(3));
            dqh = sqrt(sum((q_H_i-q(1:2)).^2));
            g = [Vs, 10]; % Ganancias de control
         else % Conducir alrededor del obstáculo
            estado2 = 2;
         end
            case 2
 if dObst < umbralObstaculo % Conducir alrededor del obstáculo
            phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
           g = [Vs, 10]; % Ganancias de control
                ff= sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2)
```

```
        if sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2) < 0.09
                estado2 = 4; % Cambiar al estado de conducción hacia la meta
            end
 else

             estado2=3;

          end

             case 3

           if dObst > 0.1  % Conducir hacia la meta
             phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
             ePhi = wrapToPi(phiRef - q(3));
             dGoal = sqrt(sum((goal-q(1:2)).^2));
             g = [Vs, 10]; % Ganancias de control
          else % Conducir alrededor del obstáculo
             % Conducir hacia el punto más cercano en la ruta original
              phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
             ePhi = wrapToPi(phiRef-q(3));
             g = [Vs, 10]; % Ganancias de control
             ff= sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2);
        if d_line_m< 0.1 && dObst<0.055
                estado = 2; % Cambiar al estado de conducción hacia la meta
        rodeo=0;
        end
           end
          end

       case 2


             phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
             ePhi = wrapToPi(phiRef - q(3));
             dGoal = sqrt(sum((goal-q(1:2)).^2));
              g = [Vs, 10]; % Ganancias de control



          end

    v = g(1) * abs(cos(ePhi));
    w = g(2) * ePhi;
    v = min([v, 0.3]); % Limitar la velocidad lineal angular máxima del robot de 3 rad/s

    vx = v * cos(q(3)); % Velocidad lineal en x del robot
    vy = v * sin(q(3)); % Velocidad lineal en y del robot
    wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w); % Velocidad angular de la
rueda derecha en rad/s
```

```
    wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w); % Velocidad angular de la rueda
izquierda en rad/s
    wdRMP = wd * 60 / (2 * pi); % Velocidad angular de la rueda derecha en rpm
    wiRMP = wi * 60 / (2 * pi); % Velocidad angular de la rueda izquierda en rpm

    dq = [vx; vy; w]; % Vector de derivadas del vector de postura
    noise = 0.00; % Para experimentar añadiendo ruido (ejemplo 0.001)
    q = q + Ts * dq + randn(3,1) * noise; % Integración de Euler

    distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
    posicionAnterior = q(1:2);

    % Simulación del movimiento del robot
    wmr.setPose(q);

    OFig1.pause(Ts);

    % Verificar si ha llegado al objetivo
    if dGoal < umbralLlegada
        break;
    end
end



% Cálculos finales
tiempoRecorrido = toc(tiempoInicial); % Tiempo transcurrido
anguloOrientacionLlegada = wrapToPi(q(3)); % Ángulo de orientación de llegada
errorPosicion = sqrt(sum((goal - q(1:2)).^2)); % Error de posición

            tiempoRecorrido = toc(tiempoInicial);
            anguloOrientacionLlegada = wrapToPi(q(3));
            errorPosicion = sqrt(sum((goal - q(1:2)).^2));

            fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
            fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
            fprintf('Ángulo de orientación de llegada: %.2f rad\n',
anguloOrientacionLlegada);
            fprintf('Error de Posición: %.2f m\n', errorPosicion);

            % Colocar los resultados en los campos de texto correspondientes
app.DistanciaTotalmEditField_2.Value = distanciaTotal;
            app.TiempoRecorridosEditField_2.Value = tiempoRecorrido;
            app.ngulodeorientacindellegadaEditField_2.Value = anguloOrientacionLlegada;
            app.ErrordePosicinEditField_2.Value = errorPosicion;



        end
```

```
        % Button pushed function: BUG0Button_3
        function BUG0Button_3Pushed(app, event)
                        clc
            clearvars -except app

            % Capturar el dato de velocidad en RPM
            Vrmp = app.VelocidadenRPMEditField_3.Value;

            Ts = 0.03; % Tiempo de muestreo
            t = 0:Ts:190; % Tiempo de simulación
            r = 0.032;
            b = 0.065;
            rps = Vrmp / 60;
            Vs = (rps * (2 * pi)) * r;


    % Obstáculos
q = [0.70; 0.20; 0]; % Posición inicial
goal = [0.8; 1.9];
obstacles{1} = flipud([0.5 0.5; 1.5 0.5; 1.5 1.40; 1.4 1.4;1.4 1.4;1.4 0.6; 0.6 0.6;0.6
1.1;0.5 1.1;0.5 0.5]);
obstacles{2} = flipud([1 1; 1.1 1;1.1 1.65; 0.4 1.65;0.4 1.5;1 1.5;1 1]);
  windowSize = 5; % Tamaño de la ventana para la media móvil
            dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
            umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
            distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
            distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
            obstacleDetected = false; % Bandera para detectar obstáculos
            obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
            closestPointToGoal = []; % Almacena el punto más cercano al objetivo
            g = [Vs, 5];
            ePhi = 0;
            dGoal = 0;
            rodeo = 1;
            posicionAnterior=0
            distanciaTotal=0
            epsilon = 0.05;
            ob = 0;
            obst = [];
            for i = 1:length(obstacles)
                obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
            end

            % Generar la primera figura en UIAxes
            cla(app.UIAxes_3);
            hold(app.UIAxes_3, 'on');
            for i = 1:length(obstacles)
                if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
                    plot(app.UIAxes_3, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2),
'Color', 'black', 'LineWidth', 0.01);
```

```
            else
                patch(app.UIAxes_3, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1],
'EdgeColor', 'black', 'LineWidth', 0.01);
            end
        end
        xlabel(app.UIAxes_3, 'x'); ylabel(app.UIAxes_3, 'y');

        ref = OMarker1(app.UIAxes_3); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
        wmr0 = OWmr1(app.UIAxes_3); wmr0.setPose(q); wmr0.setTxt('Start', 0.05,
pi/4, 'rb'); wmr0.showWmr('r:');
        wmr = OWmr1(app.UIAxes_3); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

        % Generar la segunda figura en UIAxes2
        OFig1.pause(Ts);

        phi = pi/9; % Ángulo de giro controlado
        phiObst = 0;
        umbralLlegada = 0.05; % Umbral de llegada al objetivo
        distanciaTotal = 0; % Inicia el contador de distancia total recorrida
        posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la
distancia
        tiempoInicial = tic; % Guarda el tiempo inicial

        for k = 1:length(t)
            [dObst, ~, z] = nearestSegment(q(1:2).', obst);
            dObst_buffer = [dObst_buffer(2:end), dObst];
            dObst_filtered = mean(dObst_buffer);
            dObst = dObst_filtered;
            if dObst > distanciaMinima
                phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));
            end
            if dObst > distanciaMinima
                phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
                ePhi = wrapToPi(phiRef - q(3));
                dGoal = sqrt(sum((goal - q(1:2)).^2));
                g = [Vs, 1];
                obstacleAvoided = false;
            elseif dObst <= distanciaMinima
                ePhi = wrapToPi(pi + phiObst - q(3));
                dGoal = sqrt(sum((goal - q(1:2)).^2));
                g = [Vs, 5];
            end

            v = g(1) * abs(cos(ePhi));
            w = g(2) * ePhi;
            v = min([v, 0.3]);

            vx = v * cos(q(3));
```

```
            vy = v * sin(q(3));
            wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w);
            wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w);
            wdRMP = wd * 60 / (2 * pi);
            wiRMP = wi * 60 / (2 * pi);

            dq = [vx; vy; w];
            noise = 0.00;
            q = q + Ts * dq + randn(3,1) * noise;

            distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
            posicionAnterior = q(1:2);

            wmr.setPose(q);
            OFig1.pause(Ts);

            if dGoal < umbralLlegada
               break;
            end
         end

         tiempoRecorrido = toc(tiempoInicial);
         anguloOrientacionLlegada = wrapToPi(q(3));
         errorPosicion = sqrt(sum((goal - q(1:2)).^2));

         fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
         fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
         fprintf('Ángulo de orientación de llegada: %.2f rad\n',
anguloOrientacionLlegada);
         fprintf('Error de Posición: %.2f m\n', errorPosicion);
app.DistanciaTotalmEditField_3.Value = distanciaTotal;
         app.TiempoRecorridosEditField_3.Value = tiempoRecorrido;
         app.ngulodeorientacindellegadaEditField_3.Value = anguloOrientacionLlegada;
         app.ErrordePosicinEditField_3.Value = errorPosicion;
      end

      % Button pushed function: BUG1Button_3
      function BUG1Button_3Pushed(app, event)
clc
clearvars -except app

% Capturar el dato de velocidad en RPM
Vrmp = app.VelocidadenRPMEditField_3.Value;

Ts = 0.03; % Tiempo de muestreo
t = 0:Ts:190; % Tiempo de simulación
r = 0.032;
b = 0.065;

rps = Vrmp / 60;
```

```
Vs = (rps * (2 * pi)) * r;

% MAPA 2

q = [0.70; 0.20; 0]; % Posición inicial
goal = [0.7; 1.8];
obstacles{1} = flipud([0.5 0.5; 1.5 0.5; 1.5 1.40; 1.4 1.4;1.4 1.4;1.4 0.6; 0.6 0.6;0.6
1.1;0.5 1.1;0.5 0.5]);
obstacles{2} = flipud([1 1; 1.1 1;1.1 1.65; 0.4 1.65;0.4 1.5;1 1.5;1 1]);

windowSize = 5; % Tamaño de la ventana para la media móvil
dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
obstacleDetected = false; % Bandera para detectar obstáculos
obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
closestPointToGoal = []; % Almacena el punto más cercano al objetivo
g = [Vs, 5];
ePhi = 0;
dGoal = 0;
rodeo = 1;
epsilon = 0.05;
ob = 0;
obst = [];
cla(app.UIAxes_3);
hold(app.UIAxes_3, 'on');
for i = 1:length(obstacles)
    obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
end

% Configuración de la figura y marcadores
for i = 1:length(obstacles)
    if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
        plot(app.UIAxes_3, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2), 'Color',
'black', 'LineWidth', 0.01);
    else
        patch(app.UIAxes_3, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1], 'EdgeColor',
'black', 'LineWidth', 0.01);
    end
end

estado = 0;
estado2 = 0;
guardar = 1;

dp = inf;  % Inicializa la distancia mínima a infinito
qm = [];
q_H_i = [];
```

```
% Inicialización de variables de seguimiento
distanciaTotal = 0; % Inicia el contador de distancia total recorrida
posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la distancia
tiempoInicial = tic; % Guarda el tiempo inicial
xlabel(app.UIAxes_3, 'x'); ylabel(app.UIAxes_3, 'y');

ref = OMarker1(app.UIAxes_3); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
wmr0 = OWmr1(app.UIAxes_3); wmr0.setPose(q); wmr0.setTxt('Start', 0.05, pi/4, 'rb');
wmr0.showWmr('r:');
wmr = OWmr1(app.UIAxes_3); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

OFig1.pause(Ts);
obstacleDetected = false;
f = 0;
ff = 0;
phi = pi/9; % Ángulo de giro controlado
phiObst = 0;
umbralLlegada = 0.05; % Umbral de llegada al objetivo
rodeo = 0;
for k = 1:length(t)
  [dObst, ~, z] = nearestSegment(q(1:2).', obst);
    % Actualizar el buffer del filtro de media móvil

    phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));

if dObst < umbralObstaculo
    if ~obstacleDetected
        obstacleDetected = true;
        if guardar == 1
            q_H_i = q(1:2);  % Almacena solo la primera vez que se detecta un obstáculo
            guardar = 0;
             % Evita sobrescribir q_H_i mientras se rodea el mismo obstáculo
       rodeo=1
        end
    end
else
    if obstacleDetected==1 && dObst>0.20
        obstacleDetected = false;
        guardar = 1;  % Permite que q_H_i se actualice si se detecta un nuevo obstáculo

    end
end



switch estado

    case 0
```

```
if dObst > umbralObstaculo
      phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
      ePhi = wrapToPi(phiRef - q(3));
      dGoal = sqrt(sum((goal - q(1:2)).^2));
      g = [Vs, 10]; % Ganancias de control
else

estado =1;
   end

  case 1



      if dObst < umbralObstaculo && obstacleDetected == true % Conducir alrededor
del obstáculo
            phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
            g = [Vs, 10] % Ganancias de control
         else
            if dObst > umbralObstaculo
               estado = 2;
            end
         end

  case 2

         ePhi = wrapToPi(pi+phiObst-q(3));
         dGoal = sqrt(sum((goal - q(1:2)).^2));
          g = [Vs, 10];
            if dObst > 0.15
               estado = 3;
            end

  case 3


          if dObst > umbralObstaculo  % Conducir hacia la meta
            phiRef = atan2(q_H_i(2)-q(2), q_H_i(1)-q(1))
            ePhi = wrapToPi(phiRef - q(3));
            dGoal = sqrt(sum((goal-q(1:2)).^2));
          g = [Vs, 10] % Ganancias de control
         else % Conducir alrededor del obstáculo
            % Conducir hacia el punto más cercano en la ruta original
              phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
            ePhi = wrapToPi(phiRef-q(3));
             g = [Vs, 10] % Ganancias de control
            ff=sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2);
```

```
        if sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2) < 0.12
                estado = 4; % Cambiar al estado de conducción hacia la meta
        rodeo=0;
        end
           end
    case 4

        if dObst > umbralObstaculo  % Conducir hacia la meta
              phiRef = atan2(qm(2)-q(2), qm(1)-q(1));
              ePhi = wrapToPi(phiRef - q(3));
              dGoal = sqrt(sum((qm-q(1:2)).^2));
 g = [Vs, 10] % Ganancias de control
            else % Conducir alrededor del obstáculo
                  phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
              ePhi = wrapToPi(phiRef-q(3));
               g = [Vs, 10] % Ganancias de control
              f=sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2);
               if sqrt((q(1) - qm(1))^2 + (q(2) - qm(2))^2) < 0.1
                  estado = 5 % Cambiar al estado de conducción hacia la meta

                  guardar=1;
                  rodeo=0;
                 obstacleDetected = false;
              end
        end

    case 5

       phiRef = atan2(goal(2) - q(2), goal(1) - q(1));
       ePhi = wrapToPi(phiRef - q(3));
       dGoal = sqrt(sum((goal - q(1:2)).^2));
      g = [Vs, 10] % Ganancias de control
       if rodeo==1
estado=0
       end


end


if rodeo==1

distanciaActual = norm(q(1:2) - goal);
              % Actualiza la distancia mínima y el punto si es necesario
              if distanciaActual < dp
                  dp = distanciaActual;
                  qm = q(1:2);
              end
```

```
end



    v = g(1) * abs(cos(ePhi));
    w = g(2) * ePhi;
    v = min([v, 0.3]); % Limitar la velocidad lineal angular máxima del robot de 3 rad/s

    vx = v * cos(q(3)); % Velocidad lineal en x del robot
    vy = v * sin(q(3)); % Velocidad lineal en y del robot
    wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w); % Velocidad angular de la
rueda derecha en rad/s
    wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w); % Velocidad angular de la rueda
izquierda en rad/s
    wdRMP = wd * 60 / (2 * pi); % Velocidad angular de la rueda derecha en rpm
    wiRMP = wi * 60 / (2 * pi); % Velocidad angular de la rueda izquierda en rpm

    dq = [vx; vy; w]; % Vector de derivadas del vector de postura
    noise = 0.00; % Para experimentar añadiendo ruido (ejemplo 0.001)
    q = q + Ts * dq + randn(3,1) * noise; % Integración de Euler

    distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
    posicionAnterior = q(1:2);

    % Simulación del movimiento del robot
    wmr.setPose(q);

    OFig1.pause(Ts);

    % Verificar si ha llegado al objetivo
    if dGoal < umbralLlegada
        break;
    end
end

% Cálculos finales
tiempoRecorrido = toc(tiempoInicial); % Tiempo transcurrido
anguloOrientacionLlegada = wrapToPi(q(3)); % Ángulo de orientación de llegada
errorPosicion = sqrt(sum((goal - q(1:2)).^2)); % Error de posición


% Mostrar resultados
fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
fprintf('Ángulo de orientación de llegada: %.2f rad\n', anguloOrientacionLlegada);
fprintf('Error de Posición: %.2f m\n', errorPosicion);

% Colocar los resultados en los campos de texto correspondientes
app.DistanciaTotalmEditField_3.Value = distanciaTotal;
app.TiempoRecorridosEditField_3.Value = tiempoRecorrido;
```

```
app.ngulodeorientacindellegadaEditField_3.Value = anguloOrientacionLlegada;
app.ErrordePosicinEditField_3.Value = errorPosicion;

        end

        % Button pushed function: BUG2Button_3
        function BUG2Button_3Pushed(app, event)
            clc
clearvars -except app

% Capturar el dato de velocidad en RPM
Vrmp = app.VelocidadenRPMEditField_3.Value;

Ts = 0.03; % Tiempo de muestreo
t = 0:Ts:190; % Tiempo de simulación
r = 0.032;
b = 0.065;
rps = Vrmp / 60;
Vs = (rps * (2 * pi)) * r;

% MAPA 1


q = [0.70; 0.20; 0];; % Posición inicial
q1= q;
goal = [0.7; 1.8];
obstacles{1} = flipud([0.5 0.5; 1.5 0.5; 1.5 1.40; 1.4 1.4;1.4 1.4;1.4 0.6; 0.6 0.6;0.6
1.1;0.5 1.1;0.5 0.5]);
obstacles{2} = flipud([1 1; 1.1 1;1.1 1.65; 0.4 1.65;0.4 1.5;1 1.5;1 1]);

q = [1.5; 0.05; 0]; % Posición inicial
% Obstáculos

windowSize = 5; % Tamaño de la ventana para la media móvil
dObst_buffer = zeros(1, windowSize); % Buffer para almacenar valores pasados
umbralObstaculo = 0.1; % Umbral de distancia al obstáculo
distanciaMinima = 0.1; % Distancia mínima adicional antes de girar
distanciaBuffer = 0.05; % Buffer de distancia adicional antes de girar
obstacleDetected = false; % Bandera para detectar obstáculos
obstacleStartPos = [0; 0]; % Almacena la posición inicial del obstáculo
closestPointToGoal = []; % Almacena el punto más cercano al objetivo
g = [Vs, 5];
ePhi = 0;
dGoal = 0;
rodeo = 1;
posicionAnterior=0
distanciaTotal=0
epsilon = 0.05;
ob = 0;
obst = [];
```

```
for i = 1:length(obstacles)
   obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
end

% Generar la primera figura en UIAxes
cla(app.UIAxes_3);
hold(app.UIAxes_3, 'on');

for i = 1:length(obstacles)
   if ispolycw(obstacles{i}(:,1), obstacles{i}(:,2))
      plot(app.UIAxes_3, obstacles{i}([1:end,1],1), obstacles{i}([1:end,1],2), 'Color',
'black', 'LineWidth', 0.01);
   else
      patch(app.UIAxes_3, obstacles{i}(:,1), obstacles{i}(:,2), [0.7 1 1], 'EdgeColor',
'black', 'LineWidth', 0.01);
   end
end
xlabel(app.UIAxes_3, 'x'); ylabel(app.UIAxes_3, 'y');
plot(app.UIAxes_3, [q(1), goal(1)], [q(2), goal(2)], 'color', 'black', 'LineWidth', 1);

ref = OMarker1(app.UIAxes_3); ref.showMarker('g:',0.05); ref.setPose(goal);
ref.setTxt('Goal', 0.01, pi, 'lb');
wmr0 = OWmr1(app.UIAxes_3); wmr0.setPose(q); wmr0.setTxt('Start', 0.05, pi/4, 'rb');
wmr0.showWmr('r:');
wmr = OWmr1(app.UIAxes_3); wmr.showWmr('y-'); wmr.showPath('b-', true);
wmr.setPose(q);

% Generar la segunda figura en UIAxes2
OFig1.pause(Ts);
obstacleAvoided = false;
phi = pi/9; % Ángulo de giro controlado
phiObst = 0;
umbralLlegada = 0.05; % Umbral de llegada al objetivo
distanciaTotal = 0; % Inicia el contador de distancia total recorrida
posicionAnterior = q(1:2); % Guarda la posición inicial para el cálculo de la distancia
tiempoInicial = tic; % Guarda el tiempo inicial
estado = 0;
estado2 = 0;
guardar = 1;

dp = inf;  % Inicializa la distancia mínima a infinito
qm = [];
q_H_i = [];
for k = 1:length(t)
   % Distancia al obstáculo más cercano y orientación del segmento
   [dObst, ~, z] = nearestSegment(q(1:2).', obst);
     phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));


   if dObst < umbralObstaculo
```

```
        if ~obstacleDetected
            obstacleDetected = true;
            if guardar == 1
                q_H_i = q(1:2);  % Almacena solo la primera vez que se detecta un obstáculo
                guardar = 0;
                estado = 1; % Evita sobrescribir q_H_i mientras se rodea el mismo obstáculo
            end
        end
    else
        if obstacleDetected == 1 && dObst > 0.15
            obstacleDetected = false;
            guardar = 1;  % Permite que q_H_i se actualice si se detecta un nuevo obstáculo
        end
    end

    d_line_m = abs((goal(1) - q1(1))*(q1(2) - q(2)) - (q1(1) - q(1))*(goal(2) - q1(2))) /
sqrt((goal(1) - q1(1))^2 + (goal(2) - q1(2))^2);

    switch estado
        case 0
            if dObst > umbralObstaculo  % Conducir hacia la meta
                phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
                ePhi = wrapToPi(phiRef - q(3));
                dGoal = sqrt(sum((goal-q(1:2)).^2));
                g = [dGoal/2, 10]; % Ganancias de control
            else % Conducir alrededor del obstáculo
                estado = 1;
            end

        case 1
            switch estado2
                case 0
                    if dObst < umbralObstaculo && obstacleDetected == true % Conducir
alrededor del obstáculo
                        phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
                        ePhi = wrapToPi(phiRef-q(3));
                        g = [0.2, 10]; % Ganancias de control
                    else
                        if dObst > umbralObstaculo
                            estado2 = 1;
                        end
                    end

                case 1
                    if dObst > umbralObstaculo  % Conducir hacia el punto más cercano en la
ruta original
                        ePhi = wrapToPi(pi+phiObst - q(3));
                        dqh = sqrt(sum((q_H_i-q(1:2)).^2));
                        g = [dqh/2, 10]; % Ganancias de control
```

```
            else % Conducir alrededor del obstáculo
                estado2 = 2;
            end

        case 2
            if dObst < umbralObstaculo % Conducir alrededor del obstáculo
                phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
                ePhi = wrapToPi(phiRef-q(3));
                g = [0.2, 10]; % Ganancias de control
                if sqrt((q(1) - q_H_i(1))^2 + (q(2) - q_H_i(2))^2) < 0.09
                    estado2 = 4; % Cambiar al estado de conducción hacia la meta
                end
            else
                estado2 = 3;
            end

        case 3
            if dObst > 0.1  % Conducir hacia la meta
                phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
                ePhi = wrapToPi(phiRef - q(3));
                dGoal = sqrt(sum((goal-q(1:2)).^2));
                g = [dGoal/2, 15]; % Ganancias de control
            else % Conducir alrededor del obstáculo
                phiRef = wrapToPi(phiObst + pi);  % Agregar pi para ir siempre a la
izquierda
                ePhi = wrapToPi(phiRef-q(3));
                g = [0.2, 10]; % Ganancias de control
                if d_line_m < 0.1
                    estado = 2; % Cambiar al estado de conducción hacia la meta
                    rodeo = 0;
                end
            end
    end

    case 2
        phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
        ePhi = wrapToPi(phiRef - q(3));
        dGoal = sqrt(sum((goal-q(1:2)).^2));
        g = [dGoal/2, 15]; % Ganancias de control
end

v = g(1) * abs(cos(ePhi));
w = g(2) * ePhi;
v = min([v, 0.3]); % Limitar la velocidad lineal angular máxima del robot de 3 rad/s

vx = v * cos(q(3)); % Velocidad lineal en x del robot
vy = v * sin(q(3)); % Velocidad lineal en y del robot
wd = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy + b * w); % Velocidad angular de la
rueda derecha en rad/s
```

```
    wi = (1/r) * (cos(q(3)) * vx + sin(q(3)) * vy - b * w); % Velocidad angular de la rueda
izquierda en rad/s
    wdRMP = wd * 60 / (2 * pi); % Velocidad angular de la rueda derecha en rpm
    wiRMP = wi * 60 / (2 * pi); % Velocidad angular de la rueda izquierda en rpm

    dq = [vx; vy; w]; % Vector de derivadas del vector de postura
    noise = 0.00; % Para experimentar añadiendo ruido (ejemplo 0.001)
    q = q + Ts * dq + randn(3,1) * noise; % Integración de Euler

    distanciaTotal = distanciaTotal + sqrt(sum((q(1:2) - posicionAnterior).^2));
    posicionAnterior = q(1:2);

    % Simulación del movimiento del robot
    wmr.setPose(q);

    OFig1.pause(Ts);

    % Verificar si ha llegado al objetivo
    if dGoal < umbralLlegada
        break;
    end
end

% Cálculos finales
tiempoRecorrido = toc(tiempoInicial); % Tiempo transcurrido
anguloOrientacionLlegada = wrapToPi(q(3)); % Ángulo de orientación de llegada
errorPosicion = sqrt(sum((goal - q(1:2)).^2)); % Error de posición

tiempoRecorrido = toc(tiempoInicial);
anguloOrientacionLlegada = wrapToPi(q(3));
errorPosicion = sqrt(sum((goal - q(1:2)).^2));

fprintf('Distancia Total (m): %.2f\n', distanciaTotal);
fprintf('Tiempo Recorrido (s): %.2f\n', tiempoRecorrido);
fprintf('Ángulo de orientación de llegada: %.2f rad\n', anguloOrientacionLlegada);
fprintf('Error de Posición: %.2f m\n', errorPosicion);

% Colocar los resultados en los campos de texto correspondientes
app.DistanciaTotalmEditField_3.Value = distanciaTotal;
app.TiempoRecorridosEditField_3.Value = tiempoRecorrido;
app.ngulodeorientacindellegadaEditField_3.Value = anguloOrientacionLlegada;
app.ErrordePosicinEditField_3.Value = errorPosicion;

        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
```

```
    function createComponents(app)

        % Create UIFigure and hide until all components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.Position = [100 100 1130 670];
        app.UIFigure.Name = 'MATLAB App';

        % Create TabGroup
        app.TabGroup = uitabgroup(app.UIFigure);
        app.TabGroup.Position = [2 1 1124 664];

        % Create MAPA1Tab
        app.MAPA1Tab = uitab(app.TabGroup);
        app.MAPA1Tab.Title = 'MAPA 1';

        % Create Panel_2
        app.Panel_2 = uipanel(app.MAPA1Tab);
        app.Panel_2.Position = [2 285 118 217];

        % Create BUG0Button
        app.BUG0Button = uibutton(app.Panel_2, 'push');
        app.BUG0Button.ButtonPushedFcn = createCallbackFcn(app,
@BUG0ButtonPushed, true);
        app.BUG0Button.Position = [9 147 100 23];
        app.BUG0Button.Text = 'BUG 0';

        % Create BUG1Button
        app.BUG1Button = uibutton(app.Panel_2, 'push');
        app.BUG1Button.ButtonPushedFcn = createCallbackFcn(app,
@BUG1ButtonPushed, true);
        app.BUG1Button.Position = [9 97 100 23];
        app.BUG1Button.Text = 'BUG 1';

        % Create BUG2Button
        app.BUG2Button = uibutton(app.Panel_2, 'push');
        app.BUG2Button.ButtonPushedFcn = createCallbackFcn(app,
@BUG2ButtonPushed, true);
        app.BUG2Button.Position = [9 49 100 23];
        app.BUG2Button.Text = 'BUG 2';

        % Create Panel
        app.Panel = uipanel(app.MAPA1Tab);
        app.Panel.Position = [136 34 572 530];

        % Create UIAxes
        app.UIAxes = uiaxes(app.Panel);
        title(app.UIAxes, 'Simulación')
        xlabel(app.UIAxes, 'X')
        ylabel(app.UIAxes, 'Y')
        zlabel(app.UIAxes, 'Z')
```

```
            app.UIAxes.Position = [20 37 532 454];

            % Create Panel_3
            app.Panel_3 = uipanel(app.MAPA1Tab);
            app.Panel_3.Position = [772 129 337 197];

            % Create DistanciaTotalmLabel
            app.DistanciaTotalmLabel = uilabel(app.Panel_3);
            app.DistanciaTotalmLabel.HorizontalAlignment = 'right';
            app.DistanciaTotalmLabel.FontWeight = 'bold';
            app.DistanciaTotalmLabel.Position = [13 148 115 22];
            app.DistanciaTotalmLabel.Text = 'Distancia Total (m):';

            % Create DistanciaTotalmEditField
            app.DistanciaTotalmEditField = uieditfield(app.Panel_3, 'numeric');
            app.DistanciaTotalmEditField.Editable = 'off';
            app.DistanciaTotalmEditField.FontWeight = 'bold';
            app.DistanciaTotalmEditField.Position = [274 148 44 22];

            % Create TiempoRecorridosEditFieldLabel
            app.TiempoRecorridosEditFieldLabel = uilabel(app.Panel_3);
            app.TiempoRecorridosEditFieldLabel.HorizontalAlignment = 'right';
            app.TiempoRecorridosEditFieldLabel.FontWeight = 'bold';
            app.TiempoRecorridosEditFieldLabel.Position = [13 111 129 22];
            app.TiempoRecorridosEditFieldLabel.Text = 'Tiempo Recorrido (s):';

            % Create TiempoRecorridosEditField
            app.TiempoRecorridosEditField = uieditfield(app.Panel_3, 'numeric');
            app.TiempoRecorridosEditField.Editable = 'off';
            app.TiempoRecorridosEditField.FontWeight = 'bold';
            app.TiempoRecorridosEditField.Position = [274 111 44 22];

            % Create ngulodeorientacindellegadaEditFieldLabel
            app.ngulodeorientacindellegadaEditFieldLabel = uilabel(app.Panel_3);
            app.ngulodeorientacindellegadaEditFieldLabel.HorizontalAlignment = 'right';
            app.ngulodeorientacindellegadaEditFieldLabel.FontWeight = 'bold';
            app.ngulodeorientacindellegadaEditFieldLabel.Position = [13 71 195 22];
            app.ngulodeorientacindellegadaEditFieldLabel.Text = 'Ángulo de orientación de
llegada';

            % Create ngulodeorientacindellegadaEditField
            app.ngulodeorientacindellegadaEditField = uieditfield(app.Panel_3, 'numeric');
            app.ngulodeorientacindellegadaEditField.Editable = 'off';
            app.ngulodeorientacindellegadaEditField.Position = [274 71 44 22];

            % Create ErrordePosicinEditFieldLabel
            app.ErrordePosicinEditFieldLabel = uilabel(app.Panel_3);
            app.ErrordePosicinEditFieldLabel.HorizontalAlignment = 'right';
            app.ErrordePosicinEditFieldLabel.FontWeight = 'bold';
            app.ErrordePosicinEditFieldLabel.Position = [18 38 105 22];
```

```
        app.ErrordePosicinEditFieldLabel.Text = 'Error de Posición';

        % Create ErrordePosicinEditField
        app.ErrordePosicinEditField = uieditfield(app.Panel_3, 'numeric');
        app.ErrordePosicinEditField.Editable = 'off';
        app.ErrordePosicinEditField.FontWeight = 'bold';
        app.ErrordePosicinEditField.Position = [274 38 44 22];

        % Create RESULTADOSLabel
        app.RESULTADOSLabel = uilabel(app.MAPA1Tab);
        app.RESULTADOSLabel.FontWeight = 'bold';
        app.RESULTADOSLabel.Position = [903 343 86 22];
        app.RESULTADOSLabel.Text = 'RESULTADOS';

        % Create VelocidadenRPMEditFieldLabel
        app.VelocidadenRPMEditFieldLabel = uilabel(app.MAPA1Tab);
        app.VelocidadenRPMEditFieldLabel.HorizontalAlignment = 'right';
        app.VelocidadenRPMEditFieldLabel.FontWeight = 'bold';
        app.VelocidadenRPMEditFieldLabel.Position = [787 480 108 22];
        app.VelocidadenRPMEditFieldLabel.Text = 'Velocidad en RPM';

        % Create VelocidadenRPMEditField
        app.VelocidadenRPMEditField = uieditfield(app.MAPA1Tab, 'numeric');
        app.VelocidadenRPMEditField.Position = [910 480 124 22];

        % Create MAPA2Tab
        app.MAPA2Tab = uitab(app.TabGroup);
        app.MAPA2Tab.Title = 'MAPA 2';

        % Create Panel_4
        app.Panel_4 = uipanel(app.MAPA2Tab);
        app.Panel_4.Position = [2 285 118 217];

        % Create BUG0Button_2
        app.BUG0Button_2 = uibutton(app.Panel_4, 'push');
        app.BUG0Button_2.ButtonPushedFcn = createCallbackFcn(app,
@BUG0Button_2Pushed, true);
        app.BUG0Button_2.Position = [9 147 100 23];
        app.BUG0Button_2.Text = 'BUG 0';

        % Create BUG1Button_2
        app.BUG1Button_2 = uibutton(app.Panel_4, 'push');
        app.BUG1Button_2.ButtonPushedFcn = createCallbackFcn(app,
@BUG1Button_2Pushed, true);
        app.BUG1Button_2.Position = [9 97 100 23];
        app.BUG1Button_2.Text = 'BUG 1';

        % Create BUG2Button_2
        app.BUG2Button_2 = uibutton(app.Panel_4, 'push');
```

```
        app.BUG2Button_2.ButtonPushedFcn = createCallbackFcn(app,
@BUG2Button_2Pushed, true);
        app.BUG2Button_2.Position = [9 49 100 23];
        app.BUG2Button_2.Text = 'BUG 2';

        % Create Panel_5
        app.Panel_5 = uipanel(app.MAPA2Tab);
        app.Panel_5.Position = [136 34 572 530];

        % Create UIAxes_2
        app.UIAxes_2 = uiaxes(app.Panel_5);
        title(app.UIAxes_2, 'Simulación')
        xlabel(app.UIAxes_2, 'X')
        ylabel(app.UIAxes_2, 'Y')
        zlabel(app.UIAxes_2, 'Z')
        app.UIAxes_2.Position = [20 37 532 454];

        % Create Panel_6
        app.Panel_6 = uipanel(app.MAPA2Tab);
        app.Panel_6.Position = [772 129 337 197];

        % Create DistanciaTotalmEditField_2Label
        app.DistanciaTotalmEditField_2Label = uilabel(app.Panel_6);
        app.DistanciaTotalmEditField_2Label.HorizontalAlignment = 'right';
        app.DistanciaTotalmEditField_2Label.FontWeight = 'bold';
        app.DistanciaTotalmEditField_2Label.Position = [13 148 115 22];
        app.DistanciaTotalmEditField_2Label.Text = 'Distancia Total (m):';

        % Create DistanciaTotalmEditField_2
        app.DistanciaTotalmEditField_2 = uieditfield(app.Panel_6, 'numeric');
        app.DistanciaTotalmEditField_2.Editable = 'off';
        app.DistanciaTotalmEditField_2.Position = [274 148 44 22];

        % Create TiempoRecorridosEditField_2Label
        app.TiempoRecorridosEditField_2Label = uilabel(app.Panel_6);
        app.TiempoRecorridosEditField_2Label.HorizontalAlignment = 'right';
        app.TiempoRecorridosEditField_2Label.FontWeight = 'bold';
        app.TiempoRecorridosEditField_2Label.Position = [11 112 129 22];
        app.TiempoRecorridosEditField_2Label.Text = 'Tiempo Recorrido (s):';

        % Create TiempoRecorridosEditField_2
        app.TiempoRecorridosEditField_2 = uieditfield(app.Panel_6, 'numeric');
        app.TiempoRecorridosEditField_2.Editable = 'off';
        app.TiempoRecorridosEditField_2.FontWeight = 'bold';
        app.TiempoRecorridosEditField_2.Position = [274 111 44 22];

        % Create ngulodeorientacindellegadaEditField_2Label
        app.ngulodeorientacindellegadaEditField_2Label = uilabel(app.Panel_6);
        app.ngulodeorientacindellegadaEditField_2Label.HorizontalAlignment = 'right';
        app.ngulodeorientacindellegadaEditField_2Label.FontWeight = 'bold';
```

```
        app.ngulodeorientacindellegadaEditField_2Label.Position = [12 71 195 22];
        app.ngulodeorientacindellegadaEditField_2Label.Text = 'Ángulo de orientación
de llegada';

        % Create ngulodeorientacindellegadaEditField_2
        app.ngulodeorientacindellegadaEditField_2 = uieditfield(app.Panel_6,
'numeric');
        app.ngulodeorientacindellegadaEditField_2.Editable = 'off';
        app.ngulodeorientacindellegadaEditField_2.Position = [280 70 44 22];

        % Create ErrordePosicinEditField_2Label
        app.ErrordePosicinEditField_2Label = uilabel(app.Panel_6);
        app.ErrordePosicinEditField_2Label.HorizontalAlignment = 'right';
        app.ErrordePosicinEditField_2Label.FontWeight = 'bold';
        app.ErrordePosicinEditField_2Label.Position = [18 38 105 22];
        app.ErrordePosicinEditField_2Label.Text = 'Error de Posición';

        % Create ErrordePosicinEditField_2
        app.ErrordePosicinEditField_2 = uieditfield(app.Panel_6, 'numeric');
        app.ErrordePosicinEditField_2.Editable = 'off';
        app.ErrordePosicinEditField_2.Position = [272 38 44 22];

        % Create RESULTADOSLabel_2
        app.RESULTADOSLabel_2 = uilabel(app.MAPA2Tab);
        app.RESULTADOSLabel_2.FontWeight = 'bold';
        app.RESULTADOSLabel_2.Position = [903 343 86 22];
        app.RESULTADOSLabel_2.Text = 'RESULTADOS';

        % Create VelocidadenRPMEditField_2Label
        app.VelocidadenRPMEditField_2Label = uilabel(app.MAPA2Tab);
        app.VelocidadenRPMEditField_2Label.HorizontalAlignment = 'right';
        app.VelocidadenRPMEditField_2Label.FontWeight = 'bold';
        app.VelocidadenRPMEditField_2Label.Position = [787 480 108 22];
        app.VelocidadenRPMEditField_2Label.Text = 'Velocidad en RPM';

        % Create VelocidadenRPMEditField_2
        app.VelocidadenRPMEditField_2 = uieditfield(app.MAPA2Tab, 'numeric');
        app.VelocidadenRPMEditField_2.Position = [910 480 124 22];

        % Create MAPA3Tab
        app.MAPA3Tab = uitab(app.TabGroup);
        app.MAPA3Tab.Title = 'MAPA 3';

        % Create Panel_7
        app.Panel_7 = uipanel(app.MAPA3Tab);
        app.Panel_7.Position = [2 285 118 217];

        % Create BUG0Button_3
        app.BUG0Button_3 = uibutton(app.Panel_7, 'push');
```

```
        app.BUG0Button_3.ButtonPushedFcn = createCallbackFcn(app,
@BUG0Button_3Pushed, true);
        app.BUG0Button_3.Position = [9 147 100 23];
        app.BUG0Button_3.Text = 'BUG 0';

        % Create BUG1Button_3
        app.BUG1Button_3 = uibutton(app.Panel_7, 'push');
        app.BUG1Button_3.ButtonPushedFcn = createCallbackFcn(app,
@BUG1Button_3Pushed, true);
        app.BUG1Button_3.Position = [9 97 100 23];
        app.BUG1Button_3.Text = 'BUG 1';

        % Create BUG2Button_3
        app.BUG2Button_3 = uibutton(app.Panel_7, 'push');
        app.BUG2Button_3.ButtonPushedFcn = createCallbackFcn(app,
@BUG2Button_3Pushed, true);
        app.BUG2Button_3.Position = [9 49 100 23];
        app.BUG2Button_3.Text = 'BUG 2';

        % Create Panel_8
        app.Panel_8 = uipanel(app.MAPA3Tab);
        app.Panel_8.Position = [136 34 572 530];

        % Create UIAxes_3
        app.UIAxes_3 = uiaxes(app.Panel_8);
        title(app.UIAxes_3, 'Simulación')
        xlabel(app.UIAxes_3, 'X')
        ylabel(app.UIAxes_3, 'Y')
        zlabel(app.UIAxes_3, 'Z')
        app.UIAxes_3.Position = [20 37 532 454];

        % Create Panel_9
        app.Panel_9 = uipanel(app.MAPA3Tab);
        app.Panel_9.Position = [772 129 337 197];

        % Create DistanciaTotalmEditField_3Label
        app.DistanciaTotalmEditField_3Label = uilabel(app.Panel_9);
        app.DistanciaTotalmEditField_3Label.HorizontalAlignment = 'right';
        app.DistanciaTotalmEditField_3Label.FontWeight = 'bold';
        app.DistanciaTotalmEditField_3Label.Position = [21 148 115 22];
        app.DistanciaTotalmEditField_3Label.Text = 'Distancia Total (m):';

        % Create DistanciaTotalmEditField_3
        app.DistanciaTotalmEditField_3 = uieditfield(app.Panel_9, 'numeric');
        app.DistanciaTotalmEditField_3.Editable = 'off';
        app.DistanciaTotalmEditField_3.Position = [274 148 44 22];

        % Create TiempoRecorridosEditField_3Label
        app.TiempoRecorridosEditField_3Label = uilabel(app.Panel_9);
        app.TiempoRecorridosEditField_3Label.HorizontalAlignment = 'right';
```

```
            app.TiempoRecorridosEditField_3Label.FontWeight = 'bold';
            app.TiempoRecorridosEditField_3Label.Position = [20 111 129 22];
            app.TiempoRecorridosEditField_3Label.Text = 'Tiempo Recorrido (s):';

            % Create TiempoRecorridosEditField_3
            app.TiempoRecorridosEditField_3 = uieditfield(app.Panel_9, 'numeric');
            app.TiempoRecorridosEditField_3.Editable = 'off';
            app.TiempoRecorridosEditField_3.Position = [274 111 44 22];

            % Create ngulodeorientacindellegadaEditField_3Label
            app.ngulodeorientacindellegadaEditField_3Label = uilabel(app.Panel_9);
            app.ngulodeorientacindellegadaEditField_3Label.HorizontalAlignment = 'right';
            app.ngulodeorientacindellegadaEditField_3Label.FontWeight = 'bold';
            app.ngulodeorientacindellegadaEditField_3Label.Position = [21 71 195 22];
            app.ngulodeorientacindellegadaEditField_3Label.Text = 'Ángulo de orientación
de llegada';

            % Create ngulodeorientacindellegadaEditField_3
            app.ngulodeorientacindellegadaEditField_3 = uieditfield(app.Panel_9,
'numeric');
            app.ngulodeorientacindellegadaEditField_3.Editable = 'off';
            app.ngulodeorientacindellegadaEditField_3.Position = [274 71 44 22];

            % Create ErrordePosicinEditField_3Label
            app.ErrordePosicinEditField_3Label = uilabel(app.Panel_9);
            app.ErrordePosicinEditField_3Label.HorizontalAlignment = 'right';
            app.ErrordePosicinEditField_3Label.FontWeight = 'bold';
            app.ErrordePosicinEditField_3Label.Position = [20 38 105 22];
            app.ErrordePosicinEditField_3Label.Text = 'Error de Posición';

            % Create ErrordePosicinEditField_3
            app.ErrordePosicinEditField_3 = uieditfield(app.Panel_9, 'numeric');
            app.ErrordePosicinEditField_3.Editable = 'off';
            app.ErrordePosicinEditField_3.Position = [274 38 44 22];

            % Create RESULTADOSLabel_3
            app.RESULTADOSLabel_3 = uilabel(app.MAPA3Tab);
            app.RESULTADOSLabel_3.FontWeight = 'bold';
            app.RESULTADOSLabel_3.Position = [903 343 86 22];
            app.RESULTADOSLabel_3.Text = 'RESULTADOS';

            % Create VelocidadenRPMEditField_3Label
            app.VelocidadenRPMEditField_3Label = uilabel(app.MAPA3Tab);
            app.VelocidadenRPMEditField_3Label.HorizontalAlignment = 'right';
            app.VelocidadenRPMEditField_3Label.Position = [791 480 104 22];
            app.VelocidadenRPMEditField_3Label.Text = 'Velocidad en RPM';

            % Create VelocidadenRPMEditField_3
            app.VelocidadenRPMEditField_3 = uieditfield(app.MAPA3Tab, 'numeric');
            app.VelocidadenRPMEditField_3.Position = [910 480 124 22];
```

```
            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = SimulacionBugs1

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

- **Código mblock-Python implementación en entorno real**

Bug 0

```
# generated by mBlock5 for CyberPi
# codes make you happy
import math, event, time, cyberpi, mbot2
import time
# initialize variables
Ts = 0
pi = 0
v = 0
sensor1=0
sensor2=0
sensor3=0  # Añadido sensor3
w = 0
vx = 0
```

```
vy = 0
wd = 0
wi = 0
wdRMP = 0
wiRPM = 0
r = 0
b = 0
posX = 0
posY = 0
orientacion = 0
R = 0
D = 0
Dx = 0
Dy = 0
errorOrientacion = 0
pendiente = 0
alpha = 0
anguloPi = 0
beta = 0
AnguloAtan2 = 0
orientacionReferencia = 0
listaPos = []
listaPosRef = []
listaVelocidad = []
estado = 0  # Añadido: Variable de estado
N = 5  # Tamaño de la ventana del filtro de media móvil
lecturas_sensor1 = []
lecturas_sensor2 = []
lecturas_sensor3 = []

def wrapToPi_N(Angulo):
    global Ts, pi, v, w, vx, vy, wd, wi, wdRMP, wiRPM, r, b, posX, posY, orientacion, R,
D, Dx, Dy, errorOrientacion, pendiente, alpha, anguloPi, beta, AnguloAtan2,
orientacionReferencia, listaPos, listaPosRef, listaVelocidad, estado
    anguloPi = Angulo % (2 * pi)
    if anguloPi > pi:
        anguloPi = (anguloPi - 2 * pi)
    return anguloPi

def atan2_N_N(y, x):
    global Ts, pi, v, w, vx, vy, wd, wi, wdRMP, wiRPM, r, b, posX, posY, orientacion, R,
D, Dx, Dy, errorOrientacion, pendiente, alpha, anguloPi, beta, AnguloAtan2,
orientacionReferencia, listaPos, listaPosRef, listaVelocidad, estado
    if x > 0:
        AnguloAtan2 = (math.atan(y / x) * 180.0 / math.pi)
    if x < 0 and (y > 0 or y == 0):
        AnguloAtan2 = ((math.atan(y / x) * 180.0 / math.pi) + 180)
    if x < 0 and y < 0:
        AnguloAtan2 = ((math.atan(y / x) * 180.0 / math.pi) - 180)
    if x == 0 and y > 0:
```

```
        AnguloAtan2 = 90
    if x == 0 and y < 0:
        AnguloAtan2 = -90

    AnguloAtan2 = (AnguloAtan2 * pi) / 180
    return AnguloAtan2

@event.is_press('a')
def is_btn_press():
    global Ts, pi, v, w, vx, vy, wd, wi, wdRMP, wiRPM, r, b, posX, posY, orientacion, R,
D, Dx, Dy, errorOrientacion, pendiente, alpha, anguloPi, beta, AnguloAtan2,
orientacionReferencia, listaPos, listaPosRef, listaVelocidad, estado
    pi = math.pi
    R = 0.032
    b = 0.065
    r = 0.2
    Ts = 0.1
    D = 1
    Vrmp = 40
    rps = Vrmp / 60
    Vs = (rps * (2 * pi)) * R
    listaPosRef = []
    listaPos = []
    listaVelocidad = []
    umbralObstaculo = 0.15
    listaPosRef.insert(1 - 1, 1)
    listaPosRef.insert(2 - 1, 1)
    listaPosRef.insert(3 - 1, 0)
    listaPos.insert(1 - 1, 0.7)
    listaPos.insert(2 - 1, 0.1)
    listaPos.insert(3 - 1, 0)
    cyberpi.audio.set_vol(1)
    cyberpi.audio.play('start')
    cyberpi.display.show_label("INICIO RUTA", 24, "center", index=0)
    time.sleep(1)
    while not D < 0.01:
        sensor1 = cyberpi.ultrasonic2.get(1) / 100
        sensor2 = cyberpi.ultrasonic2.get(2) / 100
        sensor3 = cyberpi.ultrasonic2.get(3) / 100



        # cyberpi.console.println(sensor1)
        # cyberpi.console.println(sensor2)
        atan2_N_N((listaPosRef[2 - 1] - listaPos[2 - 1]), (listaPosRef[1 - 1] - listaPos[1 -
1]))
        orientacionReferencia = AnguloAtan2
        Dx = (listaPosRef[1 - 1] - listaPos[1 - 1])
        Dy = (listaPosRef[2 - 1] - listaPos[2 - 1])
        D = math.sqrt((Dx * Dx + Dy * Dy))
        wrapToPi_N((orientacionReferencia - listaPosRef[3 - 1]))
```

```
    if sensor1 < umbralObstaculo or sensor2< umbralObstaculo or sensor3 <
umbralObstaculo:
        dObst = min(sensor1,sensor2, sensor3)
        estado = 1  # Cambiar estado a 1 (obstáculo detectado)
    else:
        dObst = max(sensor1, sensor2, sensor3)
        estado = 0  # Cambiar estado a 0 (sin obstáculo detectado)

    if sensor1 < umbralObstaculo or sensor2 < umbralObstaculo or sensor3 <
umbralObstaculo:
        # Supongamos que sensor1 está al frente, sensor2 y sensor3 están en los lados
izquierdos
        obst1_x = sensor1
        obst1_y = 0
        obst2_x = sensor3 * math.cos(pi / 2)
        obst2_y = sensor3 * math.sin(pi / 2)

        # Coordenadas globales de los obstáculos
        obst1_global = [posX + obst1_x * math.cos(orientacion), posY + obst1_x *
math.sin(orientacion)]
        obst2_global = [posX + obst2_x * math.cos(orientacion + pi / 2), posY +
obst2_y * math.sin(orientacion + pi / 2)]

        delta_x = obst2_global[0] - obst1_global[0]
        delta_y = obst2_global[1] - obst1_global[1]

        if abs(delta_x) > abs(delta_y):  # Obstáculo predominante en el eje X
            if delta_x > 0:
                phiObst = 0  # Frente
            else:
                phiObst = pi  # Detrás
        else:  # Obstáculo predominante en el eje Y
            if delta_y > 0:
                phiObst = pi / 2  # Izquierda
            else:
                phiObst = -pi / 2  # Derecha
    else:
        phiObst = pi


    if estado == 0:
        alpha = anguloPi
        beta = ((math.atan(r / D) * 180.0 / math.pi) * pi) / 180
        if alpha < 0:
            beta = -1 * beta

        if math.fabs(alpha) < math.fabs(beta):
            wrapToPi_N((((orientacionReferencia - listaPos[3 - 1])) + alpha))
            errorOrientacion = anguloPi
```

```
            else:
                wrapToPi_N((((orientacionReferencia - listaPos[3 - 1])) + beta))
                errorOrientacion = anguloPi

            v = Vs*0.6
            w = 1.5*errorOrientacion
            if math.fabs(w) > 1.5:
                w = 1.5*(math.fabs(w) / w)
        else:
            # Girar a la izquierda para evitar el obstáculo
            phiRef = phiObst  # Gira 90 grados a la izquierda
            errorOrientacion = wrapToPi_N(pi+phiRef - orientacion)
            v = Vs*0.7  # Reducir la velocidad para un giro más controlado
            w = 3*errorOrientacion
            if abs(w) > 3:
                w = 3*(abs(w) / w)

        vx = v * math.cos((listaPos[3 - 1] * (180 / pi)) / 180.0 * math.pi)
        vy = v * math.sin((listaPos[3 - 1] * (180 / pi)) / 180.0 * math.pi)
        wd = (1 / R) * ((math.cos((listaPos[3 - 1] * (180 / pi)) / 180.0 * math.pi) * vx +
((math.sin((listaPos[3 - 1] * (180 / pi)) / 180.0 * math.pi) * vy + b * w))))
        wi = (1 / R) * ((math.cos((listaPos[3 - 1] * (180 / pi)) / 180.0 * math.pi) * vx +
((math.sin((listaPos[3 - 1] * (180 / pi)) / 180.0 * math.pi) * vy - b * w))))
        wdRMP = (wd * 60) / (2 * pi)
        wiRPM = (wi * 60) / (2 * pi)
        mbot2.drive_speed(wiRPM, -1 * wdRMP)
        listaVelocidad = []
        listaVelocidad.insert(1 - 1, vx)
        listaVelocidad.insert(2 - 1, vy)
        listaVelocidad.insert(3 - 1, w)
        posX = (listaPos[1 - 1] + Ts * listaVelocidad[1 - 1])
        posY = (listaPos[2 - 1] + Ts * listaVelocidad[2 - 1])
        orientacion = (listaPos[3 - 1] + Ts * listaVelocidad[3 - 1])
        listaPos = []
        listaPos.insert(1 - 1, posX)
        listaPos.insert(2 - 1, posY)
        wrapToPi_N(orientacion)
        listaPos.insert(3 - 1, anguloPi)
        time.sleep(float((Ts - 0.01)))

    mbot2.EM_stop("ALL")
    cyberpi.console.clear()
    cyberpi.display.show_label("FINAL", 24, "center", index=0)
    cyberpi.audio.play('level-up')
```

Bug 1

```
import math, event, time, cyberpi, mbot2
```

```
# initialize variables
Ts = 0.3
pi = math.pi
v = 0
sensor1 = 0
sensor2 = 0
sensor3 = 0
w = 0
vx = 0
vy = 0
wd = 0
wi = 0
wdRMP = 0
wiRPM = 0
r = 0.2
b = 0.065
R = 0.032
D = 1
dr = 0
puntoCercano = [0, 0]
minDist = float('inf')
listaPos = [0.5, 0.2, 0]
listaPosRef = [0.9, 1.5, 0]
listaVelocidad = []
estado = 0
umbralObstaculo = 0.20
Vrmp = 40
rps = Vrmp / 60
Vs = (rps * (2 * pi)) * R
startAvoiding = False
startPoint = []

def wrapToPi_N(Angulo):
    anguloPi = Angulo % (2 * pi)
    if anguloPi > pi:
        anguloPi -= 2 * pi
    return anguloPi

def atan2_N_N(y, x):
    if x > 0:
        return math.atan(y / x)
    elif x < 0 and y >= 0:
        return math.atan(y / x) + pi
    elif x < 0 and y < 0:
        return math.atan(y / x) - pi
    elif x == 0 and y > 0:
        return pi / 2
    elif x == 0 and y < 0:
        return -pi / 2
```

```
@event.is_press('a')
def is_btn_press():
    global listaPos, listaPosRef, listaVelocidad, sensor1, sensor2, sensor3, D, dr, minDist,
puntoCercano, startAvoiding, startPoint, estado

    cyberpi.audio.set_vol(1)
    cyberpi.audio.play('start')
    cyberpi.display.show_label("INICIO RUTA", 24, "center", index=0)
    time.sleep(1)

    while not D < 0.07:
        sensor1 = cyberpi.ultrasonic2.get(1) / 100
        sensor2 = cyberpi.ultrasonic2.get(2) / 100
        sensor3 = cyberpi.ultrasonic2.get(3) / 100

        # Detección de obstáculos
        if sensor1 < umbralObstaculo or sensor2 < umbralObstaculo or sensor3 <
umbralObstaculo:
            distanciaObstaculo = min(sensor1, sensor2, sensor3)
            obstaculoDetectado = True
        else:
            distanciaObstaculo = max(sensor1, sensor2, sensor3)
            obstaculoDetectado = False

        if obstaculoDetectado and not startAvoiding:
            # Guardar la posición inicial de evitación
            startPoint = listaPos[:2]
            startAvoiding = True
            estado = 1  # Cambiar al estado de evitación

        # Cálculo del ángulo y la distancia al objetivo
        orientacionReferencia = atan2_N_N((listaPosRef[1] - listaPos[1]), (listaPosRef[0]
- listaPos[0]))
        Dx = (listaPosRef[0] - listaPos[0])
        Dy = (listaPosRef[1] - listaPos[1])
        D = math.sqrt((Dx * Dx + Dy * Dy))
        anguloAlpha = wrapToPi_N(orientacionReferencia - listaPos[2])

        if startAvoiding:
            distanciaActual = math.sqrt((listaPos[0] - listaPosRef[0]) ** 2 + (listaPos[1] -
listaPosRef[1]) ** 2)
            if distanciaActual < minDist:
                minDist = distanciaActual
                puntoCercano = listaPos[:2]

            if distanciaObstaculo < umbralObstaculo and obstaculoDetectado and estado ==
1:
                # Girar a la izquierda para evitar el obstáculo
                referenciaOrientacion = 0
                errorOrientacion = wrapToPi_N(referenciaOrientacion - listaPos[2])
```

```
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            if distanciaObstaculo > umbralObstaculo:
                estado = 2  # Cambiar al estado de seguimiento del obstáculo

        elif distanciaObstaculo > umbralObstaculo and obstaculoDetectado and estado
== 2:
            referenciaOrientacion = pi
            errorOrientacion = wrapToPi_N(referenciaOrientacion - listaPos[2])
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            dr = math.sqrt((listaPos[0] - startPoint[0]) ** 2 + (listaPos[1] - startPoint[1])
** 2)
            if dr < 0.1:
                estado = 3  # Cambiar al estado de retorno al punto más cercano

        elif distanciaObstaculo < umbralObstaculo and obstaculoDetectado and estado
== 2:
            referenciaOrientacion = pi
            errorOrientacion = wrapToPi_N(referenciaOrientacion - listaPos[2])
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            dr = math.sqrt((listaPos[0] - startPoint[0]) ** 2 + (listaPos[1] - startPoint[1])
** 2)
            if dr < 0.1:
                estado = 3

        elif estado == 3:
            # Dirigir hacia el punto más cercano
            orientacionReferencia = atan2_N_N((puntoCercano[1] - listaPos[1]),
(puntoCercano[0] - listaPos[0]))
            errorOrientacion = wrapToPi_N(orientacionReferencia - listaPos[2])
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            dr = math.sqrt((listaPos[0] - puntoCercano[0]) ** 2 + (listaPos[1] -
puntoCercano[1]) ** 2)
            if dr < 0.1:
                startAvoiding = False
                estado = 0  # Cambiar al estado de navegación hacia la meta
                minDist = float('inf')  # Reiniciar la distancia mínima
                puntoCercano = []
    else:
```

```
            anguloAlpha = wrapToPi_N(orientacionReferencia - listaPos[2])
            beta = math.atan(r / D)
            if anguloAlpha < 0:
                beta = -beta

            if abs(anguloAlpha) < abs(beta):
                errorOrientacion = wrapToPi_N(orientacionReferencia - listaPos[2] +
anguloAlpha)
            else:
                errorOrientacion = wrapToPi_N(orientacionReferencia - listaPos[2] + beta)

            v = Vs
            if abs(v) > 0.4:
                v = 0.4 * (abs(v) / v)

            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)

        # Calcular velocidades del motor
        vx = v * math.cos(listaPos[2])
        vy = v * math.sin(listaPos[2])
        velocidadDer = (1 / R) * (vx + b * w)
        velocidadIzq = (1 / R) * (vx - b * w)
        velocidadDerRPM = (velocidadDer * 60) / (2 * pi)
        velocidadIzqRPM = (velocidadIzq * 60) / (2 * pi)

        mbot2.drive_speed(velocidadIzqRPM, -velocidadDerRPM)

        listaVelocidad = [vx, vy, w]
        posX = listaPos[0] + Ts * listaVelocidad[0]
        posY = listaPos[1] + Ts * listaVelocidad[1]
        orientacion = listaPos[2] + Ts * listaVelocidad[2]
        listaPos = [posX, posY, wrapToPi_N(orientacion)]

        time.sleep(Ts - 0.01)

    mbot2.EM_stop("ALL")
    cyberpi.console.clear()
    cyberpi.display.show_label("FINAL", 24, "center", index=0)
    cyberpi.audio.play('level-up')
Bug 2

# generated by mBlock5 for CyberPi
# codes make you happy

import math, event, time, cyberpi, mbot2
import time
# initialize variables
Ts = 0
```

```
pi = 0
v = 0
sensor1 = 0
sensor2 = 0
w = 0
vx = 0
vy = 0
wd = 0
wi = 0
wdRMP = 0
wiRPM = 0
r = 0
b = 0
posX = 0
posY = 0
orientacion = 0
R = 0
D = 0
Dx = 0
Dy = 0
errorOrientacion = 0
pendiente = 0
alpha = 0
anguloPi = 0
beta = 0
AnguloAtan2 = 0
orientacionReferencia = 0
listaPos = []
listaPosRef = []
listaVelocidad = []

def puntolinea(Xx, Xy, Sx, Sy, Gx, Gy):
    return abs((Gx - Sx) * (Sy - Xy) - (Sx - Xx) * (Gy - Sy)) / math.sqrt((Gx - Sx) ** 2 +
(Gy - Sy) ** 2)

def wrapToPi_N(Angulo):
    global Ts, pi, v, w, vx, vy, wd, wi, wdRMP, wiRPM, r, b, posX, posY, orientacion, R,
D, Dx, Dy, errorOrientacion, pendiente, alpha, anguloPi, beta, AnguloAtan2,
orientacionReferencia, listaPos, listaPosRef, listaVelocidad
    anguloPi = Angulo % (2 * pi)
    if anguloPi > pi:
        anguloPi -= 2 * pi
    return anguloPi

def atan2_N_N(y, x):
    global Ts, pi, v, w, vx, vy, wd, wi, wdRMP, wiRPM, r, b, posX, posY, orientacion, R,
D, Dx, Dy, errorOrientacion, pendiente, alpha, anguloPi, beta, AnguloAtan2,
orientacionReferencia, listaPos, listaPosRef, listaVelocidad
    if x > 0:
        AnguloAtan2 = (math.atan(y / x) * 180.0 / math.pi)
```

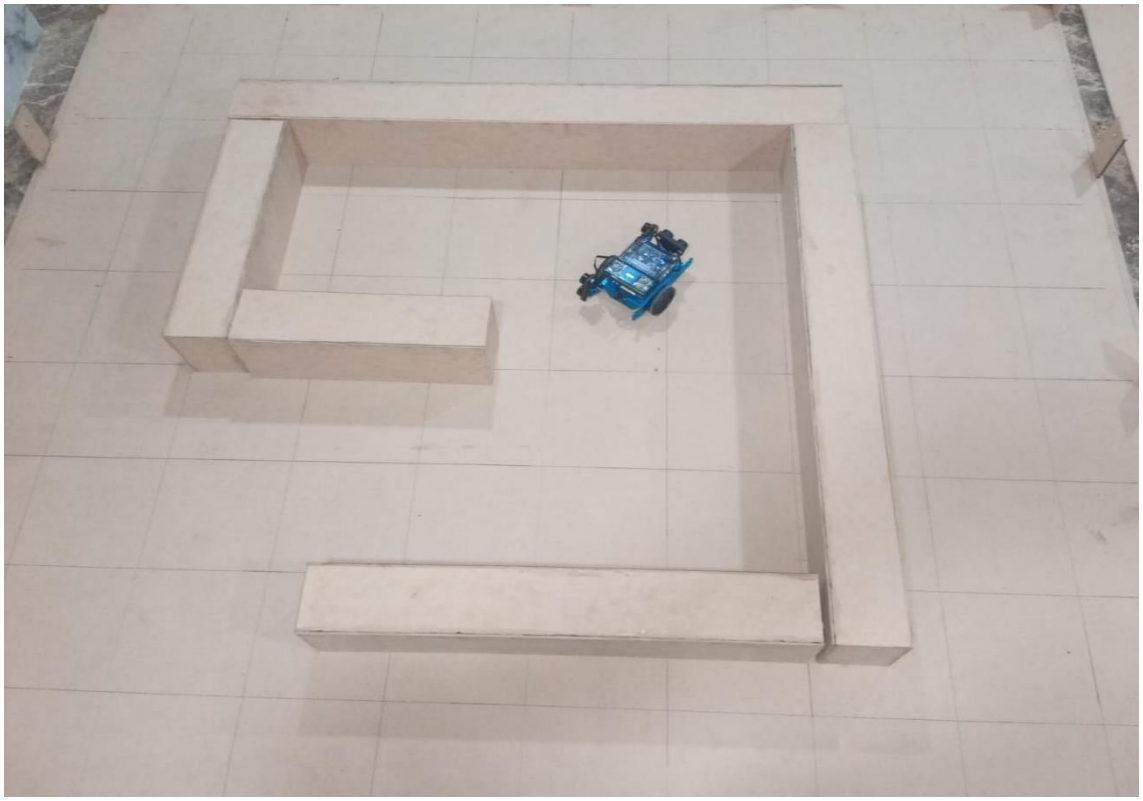
```
    elif x < 0 and (y >= 0):
        AnguloAtan2 = ((math.atan(y / x) * 180.0 / math.pi) + 180)
    elif x < 0 and y < 0:
        AnguloAtan2 = ((math.atan(y / x) * 180.0 / math.pi) - 180)
    elif x == 0 and y > 0:
        AnguloAtan2 = 90
    elif x == 0 and y < 0:
        AnguloAtan2 = -90

    AnguloAtan2 = (AnguloAtan2 * pi) / 180

@event.is_press('a')
def is_btn_press():
    global Ts, pi, v, w, vx, vy, wd, wi, wdRMP, wiRPM, r, b, posX, posY, orientacion, R,
D, Dx, Dy, errorOrientacion, pendiente, alpha, anguloPi, beta, AnguloAtan2,
orientacionReferencia, listaPos, listaPosRef, listaVelocidad
    pi = math.pi
    R = 0.032
    b = 0.065
    r = 0.2
    Ts = 0.5
    D = 1
    Vrmp = 40
    rps = Vrmp / 60
    Vs = (rps * (2 * pi)) * R
    listaPosRef = []
    listaPos = []
    listaVelocidad = []
    estado = 0
    startAvoiding = False
    umbralObstaculo = 0.20
    listaPosRef.insert(0, 1)
    listaPosRef.insert(1, 1.3)
    listaPosRef.insert(2, 0)
    listaPos.insert(0, 0.2)
    listaPos.insert(1, 0.2)
    listaPos.insert(2, 0)
    cyberpi.audio.set_vol(1)
    cyberpi.audio.play('start')
    cyberpi.display.show_label("INICIO RUTA", 24, "center", index=0)
    time.sleep(1)
    while not D < 0.07:
        sensor1 = cyberpi.ultrasonic2.get(1) / 100
        sensor2 = cyberpi.ultrasonic2.get(2) / 100
        sensor3 = cyberpi.ultrasonic2.get(3) / 100
        # cyberpi.console.println(sensor1)
        # cyberpi.console.println(sensor2)
        # Detección de obstáculos
        if sensor1 < umbralObstaculo or sensor2 < umbralObstaculo or sensor3 <
umbralObstaculo:
```

```
            distanciaObstaculo = min(sensor1, sensor2, sensor3)
            obstaculoDetectado = True
        else:
            distanciaObstaculo = max(sensor1, sensor2, sensor3)
            obstaculoDetectado = False

        if obstaculoDetectado and not startAvoiding:
            # Guardar la posición inicial de evitación
            startPoint = listaPos[:2]
            startAvoiding = True

        atan2_N_N((listaPosRef[1] - listaPos[1]), (listaPosRef[0] - listaPos[0]))
        orientacionReferencia = AnguloAtan2
        Dx = (listaPosRef[0] - listaPos[0])
        Dy = (listaPosRef[1] - listaPos[1])
        D = math.sqrt((Dx * Dx + Dy * Dy))
        wrapToPi_N((orientacionReferencia - listaPosRef[2]))
        puntolinea_valor = puntolinea(listaPos[0], listaPos[1], listaPosRef[0],
listaPosRef[1], listaPos[0], listaPos[1])

        if sensor1 < umbralObstaculo or sensor2 < umbralObstaculo or sensor3 <
umbralObstaculo:
            distanciaObstaculo = min(sensor1, sensor2, sensor3)
            obstaculoDetectado = True
        else:
            distanciaObstaculo = max(sensor1, sensor2, sensor3)
            obstaculoDetectado = False

        if puntolinea_valor < 0.1 and estado == 0:
            alpha = anguloPi
            beta = ((math.atan(r / D) * 180.0 / math.pi) * pi) / 180
            if alpha < 0:
                beta = -1 * beta

            if abs(alpha) < abs(beta):
                wrapToPi_N((((orientacionReferencia - listaPos[2])) + alpha))
                errorOrientacion = anguloPi
            else:
                wrapToPi_N((((orientacionReferencia - listaPos[2])) + beta))
                errorOrientacion = anguloPi

            v = Vs
            if abs(v) > 0.4:
                v = 0.4 * (abs(v) / v)
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            if distanciaObstaculo < umbralObstaculo:
                estado = 1
```

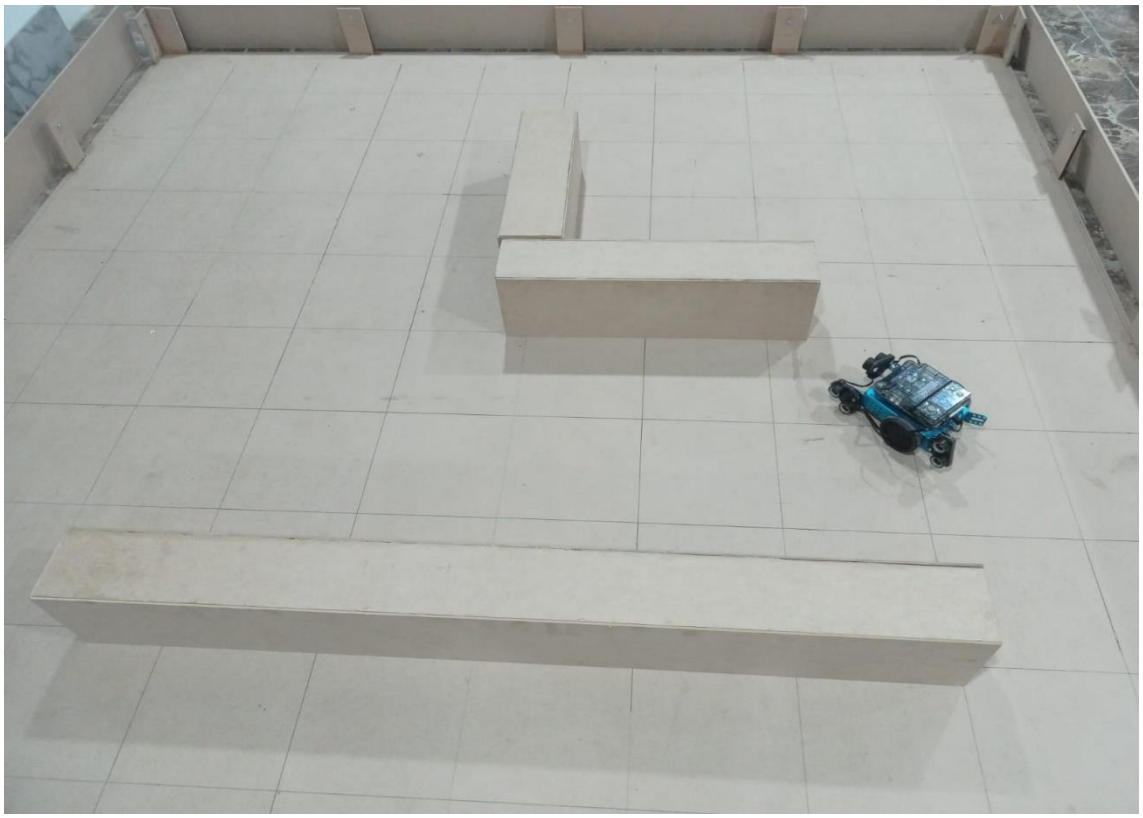
```
        elif distanciaObstaculo < umbralObstaculo  and estado == 1:
            # Girar a la izquierda para evitar el obstáculo
            referenciaOrientacion = 0
            errorOrientacion = wrapToPi_N(referenciaOrientacion - orientacion)
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)

        elif distanciaObstaculo > umbralObstaculo and puntolinea_valor > 0.04:
            referenciaOrientacion = pi
            errorOrientacion = wrapToPi_N(referenciaOrientacion - orientacion)
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            if puntolinea_valor < 0.05:
                estado = 0
        elif distanciaObstaculo < umbralObstaculo  and puntolinea_valor > 0.04 :
            referenciaOrientacion = pi
            errorOrientacion = wrapToPi_N(referenciaOrientacion - orientacion)
            v = Vs
            w = 2 * errorOrientacion
            if abs(w) > 3:
                w = 3 * (abs(w) / w)
            if puntolinea_valor < 0.05:
                estado = 0
        elif distanciaObstaculo > umbralObstaculo:
                estado = 2
        vx = v * math.cos((listaPos[2] * (180 / pi)) / 180.0 * math.pi)
        vy = v * math.sin((listaPos[2] * (180 / pi)) / 180.0 * math.pi)
        wd = (1 / R) * ((math.cos((listaPos[2] * (180 / pi)) / 180.0 * math.pi) * vx +
((math.sin((listaPos[2] * (180 / pi)) / 180.0 * math.pi) * vy + b * w))))
        wi = (1 / R) * ((math.cos((listaPos[2] * (180 / pi)) / 180.0 * math.pi) * vx +
((math.sin((listaPos[2] * (180 / pi)) / 180.0 * math.pi) * vy - b * w))))
        wdRMP = (wd * 60) / (2 * pi)
        wiRPM = (wi * 60) / (2 * pi)
        mbot2.drive_speed(wiRPM, -1 * wdRMP)
        listaVelocidad = []
        listaVelocidad.insert(0, vx)
        listaVelocidad.insert(1, vy)
        listaVelocidad.insert(2, w)
        posX = listaPos[0] + Ts * listaVelocidad[0]
        posY = listaPos[1] + Ts * listaVelocidad[1]
        orientacion = listaPos[2] + Ts * listaVelocidad[2]
        listaPos = []
        listaPos.insert(0, posX)
        listaPos.insert(1, posY)
        wrapToPi_N(orientacion)
        listaPos.insert(2, anguloPi)
```

```
        time.sleep(float((Ts - 0.01)))

    mbot2.EM_stop("ALL")
    cyberpi.console.clear()
    cyberpi.display.show_label("FINAL", 24, "center", index=0)
    cyberpi.audio.play('level-up')

# generated by mBlock5 for CyberPi
# codes make you happy
```

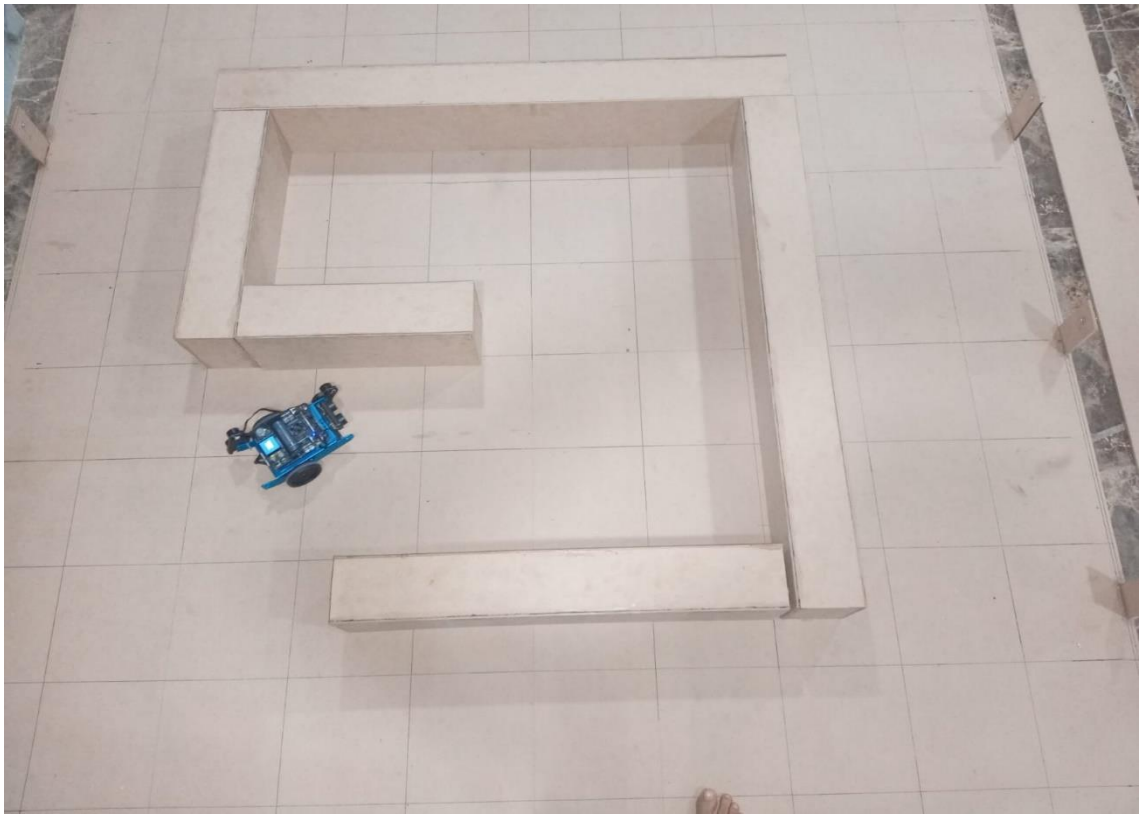
ANEXOS B



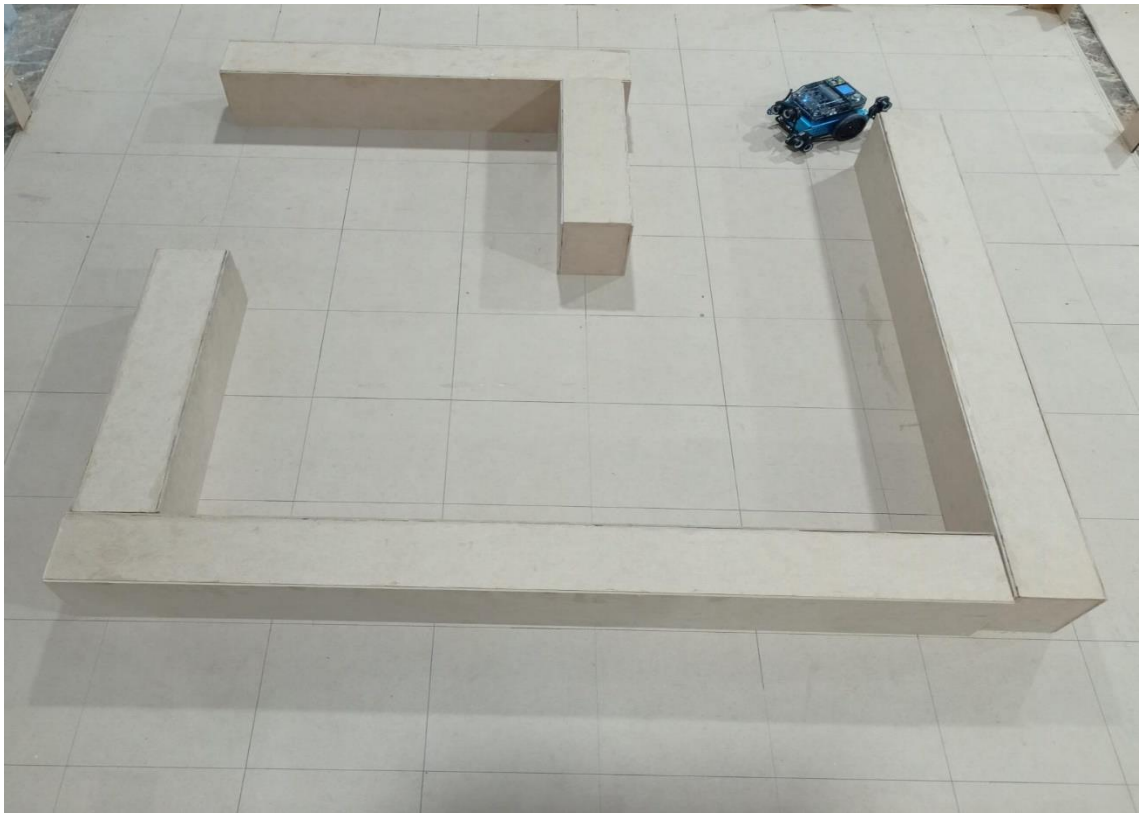
Anexo 1. Implementación de pista 2 para el insecto 2



Anexo 2. Prueba en entorno real pista 1 insecto 0



Anexo 3. Prueba en entorno real pista 2 insecto 1



Anexo 4. Prueba en entorno real pista 3 insecto 2

CERTIFICADO DE ANÁLISIS
magister

TESIS CULMINADA

4% Textos sospechosos

4% Similitudes
< 1% similitudes entre comillas
0% entre las fuentes mencionadas
< 1% Idiomas no reconocidos

Nombre del documento: TESIS CULMINADA.docx
ID del documento: a39d35da9339c6be96d7628fb55c5e79095c1ff8
Tamaño del documento original: 7,35 MB

Depositante: Junior Rafael Figueroa Olmedo
Fecha de depósito: 1/7/2024
Tipo de carga: interface
fecha de fin de análisis: 1/7/2024

Número de palabras: 38.568
Número de caracteres: 279.011

Ubicación de las similitudes en el documento:

Fuentes principales detectadas

Nº	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	ciladi.org https://ciladi.org/wp-content/uploads/Libro-Robots-VF3.pdf	< 1%		Palabras idénticas: < 1% (325 palabras)
2	Documento de otro usuario #9c1170 El documento proviene de otro grupo 23 fuentes similares	< 1%		Palabras idénticas: < 1% (182 palabras)
3	dspace.ups.edu.ec http://dspace.ups.edu.ec/bitstream/123456789/23081/1/TTS885.pdf	< 1%		Palabras idénticas: < 1% (282 palabras)
4	repositorio.utn.edu.ec http://repositorio.utn.edu.ec/bitstream/123456789/14271/2/04 MEL 212 TRABAJO DE GRADO.pdf 23 fuentes similares	< 1%		Palabras idénticas: < 1% (149 palabras)
5	upcommons.upc.edu https://upcommons.upc.edu/bitstream/2117/371091/1/172257_TFG_Control_Rumb_Elvis_Prez_.pdf 20 fuentes similares	< 1%		Palabras idénticas: < 1% (130 palabras)

Fuentes con similitudes fortuitas

Nº	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	Documento de otro usuario #d6a1b8 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (30 palabras)
2	dspace.susu.ru http://dspace.susu.ru/xmlui/bitstream/0001.74/41111/1/2021_426_dolganovmi.pdf	< 1%		Palabras idénticas: < 1% (32 palabras)
3	Documento de otro usuario #32d3dc El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (28 palabras)
4	upcommons.upc.edu https://upcommons.upc.edu/bitstream/2117/401805/2/memoria.pdf	< 1%		Palabras idénticas: < 1% (25 palabras)
5	1library.co | Requisitos funcionales - Identificación de requerimientos https://1library.co/article/requisitos-funcionales-identificación-de-requerimientos.zgw4xevy	< 1%		Palabras idénticas: < 1% (25 palabras)

Fuentes mencionadas (sin similitudes detectadas)

Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1 https://www.researchgate.net/
2 https://www
3 https://am990formosa.com/robots-y-drones-expectativa-en-las-vegas-por-las
4 https://www.mipatente.com/disenan
5 https://la.mathworks.com/

JUNIOR RAFAEL FIGUEROA OLMEDO

Firmado digitalmente por JUNIOR RAFAEL FIGUEROA OLMEDO
Fecha: 2024.07.04 10:33:47 -05'00'

Anexo 5. Certificado de análisis de coincidencia.