
# UNIVERSIDAD UTE

## FACULTAD DE CIENCIAS DE LA INGENIERÍA E INDUSTRIAS

## CARRERA DE INGENIERÍA MECATRÓNICA

# DESARROLLO DE ALGORITMOS DE EVASIÓN DE OBSTÁCULOS PARA UN ROBOT MÓVIL DE TRACCIÓN DIFERENCIAL

**TRABAJO PREVIO A LA OBTENCIÓN DEL TÍTULO**

**DE INGENIERO EN MECATRÓNICA**

**MARLON BENJAMÍN CARAGUAY QUEVEDO**

**DIRECTOR: MSC. RAÚL VICENTE PAREDES LOOR**

**Quito, Julio del 2021**

# FORMULARIO DE REGISTRO BIBLIOGRÁFICO

# TRABAJO DE TITULACIÓN

| DATOS DE CONTACTO | |
|---|---|
| **CÉDULA DE IDENTIDAD**: | **1104126212** |
| **APELLIDO Y NOMBRES:** | **Caraguay Quevedo Marlon Benjamín** |
| **DIRECCIÓN:** | **Av. 6 de Diciembre y de las Cucardas** |
| **EMAIL:** | **marlonlx@hotmail.com** |
| **TELÉFONO FIJO:** | **02 512 2978** |
| **TELÉFONO MOVIL:** | **0997170355** |

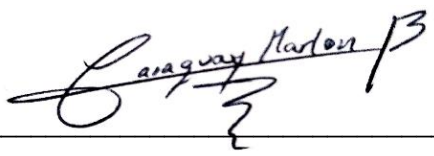
| DATOS DE LA OBRA | |
|---|---|
| **TÍTULO**: | **Desarrollo de Algoritmos de evasión de evasión de obstáculos para un robot móvil de tracción diferencial** |
| **AUTOR O AUTORES**: | **Marlon Benjamín Caraguay Quevedo** |
| **FECHA DE ENTREGA DEL PROYECTO DE TITULACIÓN**: | **16/7/2020** |
| **DIRECTOR DEL PROYECTO DE TITULACIÓN:** | **MSc. Raúl Vicente Paredes Loor** |
| **PROGRAMA** | **PREGRADO** [x] **POSGRADO** [ ] |
| **TÍTULO POR EL QUE OPTA:** | **Ingeniero Mecatrónico** |
| **RESUMEN: Mínimo 250 palabras** | Este proyecto de titulación, propone el análisis del rendimiento y comportamiento de 5 distintos algoritmos de evasión de obstáculos, mientras estos guían a un robot móvil de tracción diferencial desde un punto inicial A hasta un punto objetivo B, todo esto dentro de un mapa estructurado y lleno de obstáculos. El robot móvil de tracción diferencial se modela mediante las ecuaciones de cinemática inversa provistas por la robótica. |

| | |
|---|---|
| | Con el objetivo de garantizar el comportamiento característico de los algoritmos, este proyecto partió desde la lógica primordial de cada uno. Por lo tanto, se analizó la secuencia lógica que sigue cada algoritmo y esta se codificó dentro del software *MatLab*, el cual fue utilizado debido a que su complemento *Simulink* resulta muy útil y versátil para las simulaciones de prueba. El atractivo de este complemento se debe su a su gran variedad de bloques de funciones que facilitan la ejecución y simulación de los algoritmos.<br>Para verificar que todos los algoritmos cumplían con su función primaria, se empleó un mapa de comprobación inicial, el cual se trataba de un mapa sencillo con un solo obstáculo central. La confirmación del buen funcionamiento de cada algoritmo se conseguía al momento en que el algoritmo lograba generar una ruta solución desde una esquina del mapa hasta su extremo opuesto.<br>Para las pruebas se definieron 10 rutas dentro del mapa estructurado y lleno de obstáculos mencionado al inicio, al cual se lo llamó “mapa de prueba”. Para obtener resultados, se utilizó cada algoritmo para guiar al robot móvil de tracción diferencial a través de cada una de las rutas definidas. Los resultados obtenidos en las pruebas fueron utilizados para conseguir el objetivo final de este proyecto, el cual fue obtener un análisis del rendimiento y |

| | |
|---|---|
| | comportamiento de cada algoritmo, así como un análisis comparativo entre ellos. |
| **PALABRAS CLAVES:** | Algoritmo de evasión de obstáculos, navegación autónoma, robótica móvil, campo potencial artificial. |
| **ABSTRACT:** | The present thesis Project proposes an analysis of the performance and behavior of 5 different obstacle avoidance algorithms in the task of leading a differential drive mobile robot from an initial point A to an objective point B inside of a structured map full of obstacles. The differential drive mobile robot is modeled using the inverse kinematics equations provided by robotics.<br>In order to guarantee the characteristic behavior of the algorithms, this project began from the prime logic of each one. Hence, the logical sequence of each algorithm was analyzed and coded using the software MatLab which was used because of the versatility and usefulness of its complement called Simulink for the test simulations. The attractiveness of this complement is caused by the great variety of function blocks that facilitate the execution and simulation of the algorithms.<br>An initial verification map was employed in order to verify that every algorithm achieved their prime function. This map was a simply one which had a single obstacle located in the center part. The confirmation that each algorithm was |

| | |
|---|---|
| | working properly was gotten on the moment that the current algorithm accomplished a solution route from one corner of the map to its opposite end.<br>For the tests, ten routes were defined inside of the structured full of obstacles map which was renamed as “test map”. In order to get results, each algorithm was employed to lead the differential drive mobile robot across each one of the defined routes. The results obtained in the tests were used in order to accomplish the main objective of this project, which was to obtain a performance and behavior analysis of each algorithm as well as a comparative analysis between them. |
| **KEYWORDS** | Obstacle avoidance algorithms, autonomous navigation, mobile robotics, artificial potential field. |

Se autoriza la publicación de este Proyecto de Titulación en el Repositorio Digital de la Institución.

f: 

CARAGUAY QUEVEDO MARLON BENJAMÍN

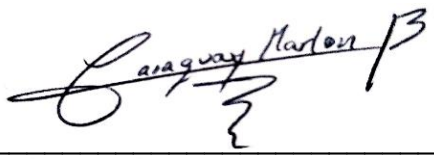
CI: 1104126212

# DECLARACIÓN Y AUTORIZACIÓN

Yo, Marlon Benjamín Caraguay Quevedo, CI: 1104126212 autor del trabajo de titulación: Desarrollo de algoritmos de evasión de obstáculos para un robot móvil de tracción diferencial previo a la obtención del título de Ingeniero Mecatrónico en la Universidad UTE.

1. Declaro tener pleno conocimiento de la obligación que tienen las Instituciones de Educación Superior, de conformidad con el Artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de titulación de grado para que sea integrado al Sistema Nacional de información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.
2. Autorizo a la BIBLIOTECA de la Universidad UTE a tener una copia del referido trabajo de titulación de grado con el propósito de generar un Repositorio que democratice la información, respetando las políticas de propiedad intelectual vigentes.

Quito, 16 de julio de 2021.



f:_____

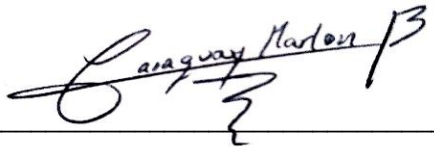
CARAGUAY QUEVEDO MARLON BENJAMÍN

CI: 1104126212

# DECLARACION JURAMENTADA DEL AUTOR

Yo, **Marlon Benjamín Caraguay Quevedo**, portador(a) de la cédula de identidad Nº**1104126212**, declaro que el trabajo aquí descrito es de mi autoría, que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en ese documento.

La Universidad UTE puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

f: 

CARAGUAY QUEVEDO MARLON BENJAMÍN

CI: 1104126212

# CERTIFICACIÓN DEL TUTOR

En mi calidad de tutor de tesis de grado, certifico que el presente trabajo que lleva por título **Desarrollo de algoritmos de evasión de obstáculos para un robot móvil de tracción diferencial** para aspirar al título de **Ingeniero Mecatrónico** fue desarrollado por **Marlon Benjamín Caraguay Quevedo,** bajo mi dirección y supervisión, en la Facultad de Ciencias de la Ingeniería e Industrias; y que dicho trabajo cumple con las condiciones requeridas para ser sometido a la presentación pública y evaluación por parte del Jurado examinador que se designe.

RAUL VICENTE PAREDES LOOR

Firmado digitalmente por RAUL VICENTE PAREDES LOOR
Fecha: 2021.09.08 13:49:27 -05'00'

MSc. Raúl Vicente Paredes Loor

**DIRECTOR DEL TRABAJO**

C.I. 1721986790

# DEDICATORIA

Para comenzar, dedico esta tesis a Dios, al Diviño Niño Jesús, al Espíritu Santo, a la Virgen del Cisne y a San Juan Bosco, quienes han sabido guiar mi mente y mis manos para poder escribir esta tesis. Dedico también esta tesis a las personas más importantes de mi vida, mi familia.

A María, mi madre, quien me ha apoyado, cuidado, guiado y tolerado durante toda mi vida. Siempre tendré en mi mente el recuerdo de cada abrazo, de cada consuelo, y de cada muestra de apoyo y ánimo que ha sabido brindarme a lo largo de mi vida y en especial en los momentos difíciles en los cuales me encontraba triste y perdido.

A Segundo, mi padre, quien siempre ha procurado darme todo lo que he necesitado. Sé que siempre ha estado orgulloso de mí incluso a pesar de todos los errores que he cometido y las pérdidas que le he costado. Enserio le agradezco de corazón por haberme disculpado por mis errores y por seguirme apoyando y queriendo a pesar de todas mis fallas.

A Darwin, mi hermano, a quien le agradezco con todo mi vida el que siempre haya visto por mí incluso por encima de sí mismo. De verdad es un hombre remarcable quien es un ejemplo de cómo debe ser un buen hermano mayor. Por todas las cosas que ha hecho por mí, él siempre tendrá mi cariño, apoyo y consideración en todo momento, lugar y circunstancia.

A Otilia, mi abuela, a quien espero contarle acerca de mi graduación cuando esta tesis sea aceptada y defendida. Su apoyo a la distancia me da fuerzas para seguir adelante, el anhelo de volver a verla me hace sentir motivado, y su recuerdo me reconforta en tiempos difíciles.

A Esperanza, mi tía, quien siempre ha mostrado su apoyo y consejo para mí. Su carisma y forma gentil de ser es algo que siempre me ha reconfortado. Agradeceré siempre el cuidado y la ayuda que me ha brindado a lo largo de mi vida.

Yo no sería el hombre que soy ahora sin la influencia directa de las personas que he nombrado en esta dedicatoria, así que a todas ellas les digo gracias de todo corazón. Ojala hubiera más que palabras para expresar lo agradecido que estoy con todos y cada uno de ustedes. Los amo con todo mi ser y prometo siempre cuidarlos y protegerlos.

# INDICE DE CONTENIDOS

# INDICE DE TABLAS

**PÁGINA**

# INDICE DE FIGURAS

**PÁGINA**

# INDICE DE ANEXOS

**PÁGINA**

# RESUMEN

Este proyecto de titulación, propone el análisis del rendimiento y comportamiento de 5 distintos algoritmos de evasión de obstáculos, mientras estos guían a un robot móvil de tracción diferencial desde un punto inicial A hasta un punto objetivo B, todo esto dentro de un mapa estructurado y lleno de obstáculos. El robot móvil de tracción diferencial se modela mediante las ecuaciones de cinemática inversa provistas por la robótica.

Con el objetivo de garantizar el comportamiento característico de los algoritmos, este proyecto partió desde la lógica primordial de cada uno. Por lo tanto, se analizó la secuencia lógica que sigue cada algoritmo y esta se codificó dentro del software *MatLab*, el cual fue utilizado debido a que su complemento *Simulink* resulta muy útil y versátil para las simulaciones de prueba. El atractivo de este complemento se debe su a su gran variedad de bloques de funciones que facilitan la ejecución y simulación de los algoritmos.

Para verificar que todos los algoritmos cumplían con su función primaria, se empleó un mapa de comprobación inicial, el cual se trataba de un mapa sencillo con un solo obstáculo central. La confirmación del buen funcionamiento de cada algoritmo se conseguía al momento en que el algoritmo lograba generar una ruta solución desde una esquina del mapa hasta su extremo opuesto.

Para las pruebas se definieron 10 rutas dentro del mapa estructurado y lleno de obstáculos mencionado al inicio, al cual se lo llamó “mapa de prueba”. Para obtener resultados, se utilizó cada algoritmo para guiar al robot móvil de tracción diferencial a través de cada una de las rutas definidas. Los resultados obtenidos en las pruebas fueron utilizados para conseguir el objetivo final de este proyecto, el cual fue obtener un análisis del rendimiento y comportamiento de cada algoritmo, así como un análisis comparativo entre ellos.

**Palabras clave:** Algoritmo de evasión de obstáculos, navegación autónoma, robótica móvil, campo potencial artificial.

# ABSTRACT

The present thesis Project proposes an analysis of the performance and behavior of 5 different obstacle avoidance algorithms in the task of leading a differential drive mobile robot from an initial point A to an objective point B inside of a structured map full of obstacles. The differential drive mobile robot is modeled using the inverse kinematics equations provided by robotics.

In order to guarantee the characteristic behavior of the algorithms, this project began from the prime logic of each one. Hence, the logical sequence of each algorithm was analyzed and coded using the software MatLab which was used because of the versatility and usefulness of its complement called Simulink for the test simulations. The attractiveness of this complement is caused by the great variety of function blocks that facilitate the execution and simulation of the algorithms.

An initial verification map was employed in order to verify that every algorithm achieved their prime function. This map was a simply one which had a single obstacle located in the center part. The confirmation that each algorithm was working properly was gotten on the moment that the current algorithm accomplished a solution route from one corner of the map to its opposite end.

For the tests, ten routes were defined inside of the structured full of obstacles map which was renamed as “test map”. In order to get results, each algorithm was employed to lead the differential drive mobile robot across each one of the defined routes. The results obtained in the tests were used in order to accomplish the main objective of this project, which was to obtain a performance and behavior analysis of each algorithm as well as a comparative analysis between them.

**Keywords:** Obstacle avoidance algorithms, autonomous navigation, mobile robotics, artificial potential field.

# 1. INTRODUCCION

Los robots móviles, desde su creación, han sido utilizados en innumerables campos de aplicación. Desde los robots móviles más simples como aspiradoras automáticas hasta los más complejos utilizados para la exploración espacial, la robótica móvil tiene un atractivo esencial que la hace ser objeto estudio y desarrollo continuo. Este motivo es el dar la posibilidad de ejecutar tareas o actividades que un humano no está dispuesto o es incapaz de realizar por distintos motivos como la seguridad, la eficiencia, el espacio, entre otros.

De la necesidad de que estos robots móviles sean capaces de moverse por un entorno sin la asistencia de un operador, nace la idea de la navegación autónoma, la cual permita al robot moverse hacia un objetivo asignado a la par que evita obstáculos dentro de un ambiente que puede o no ser conocido con anticipación. Escenarios como la exploración y cartografiado de cuevas o minas abandonadas, son perfectos ejemplos de tareas que un humano no es capaz de realizar por motivos de seguridad. En China ya se ha creado un prototipo de robot de búsqueda y rescate para exploración de minas, el cual cuenta con tres modos de funcionamiento, uno de ellos, automático. (Zhao et al., 2017)

Dando un paso más adelante, la navegación autónoma no solo responde a la necesidad de evitar un riesgo para el humano, sino también de tomar su lugar en actividades que pudieren ser consideradas repetitivas o tediosas. De allí que la navegación autónoma puede llegar a ser implementada en diversos ambientes como en fábricas, donde un robot autónomo pudiera llevar la materia prima hasta su destino evitando obstáculos y a trabajadores; comercios, donde un robot tomaría el rol de carrito de compras y seguiría al comprador, e incluso de catering, donde un pequeño robot pasearía por un salón ofreciendo bocadillos a los invitados sin chocar contra estos. (Valverde, 2020).

Para que un robot cuente con una navegación autónoma, es necesario que este sea guiado por un algoritmo que le indique su dirección y velocidad. Para este objetivo, se tiene una pequeña variedad de algoritmos los cuales tienen diferentes grados de complejidad. Habiendo opciones, cabe preguntarse cuál es el mejor de ellos, pregunta para la cual no existe una respuesta absoluta ya que esta dependerá de la aplicación que se le fuere a dar al robot. Por lo tanto, es necesario un análisis más profundo de las alternativas que existen para otorgar a un robot de navegación autónoma. Este análisis definiría ventajas, desventajas y comportamiento de los algoritmos, de forma que se pudiera obtener una comparativa detallada entre cada algoritmo, la cual permitiría una proliferación de estos robots a nivel industrial, comercial, doméstico e incluso estudiantil; permitiendo dar un más adelante dentro de la era digital.

La navegación autónoma constituye un tema que crea una gran cantidad de opciones de aplicación a cualquier nivel, desde lo domestico hasta lo comercial, pasando por lo industrial y urbano. A pesar de que este tema fue tocado por primera vez en la década de los 80', este no se ha desarrollado de la misma forma en la que se han desarrollado otras tecnologías como los smartphones, una gran pérdida ya que el concepto tiene un potencial impresionante. De desarrollarse completamente el tema, se podría especular el ver en un futuro a robots móviles realizando entregas de servicios de delivery que actualmente son hechas por humanos.

Pero cabe preguntarse el por qué no existe un desarrollo más a profundidad de este tema. La causa más probable es la complejidad del mismo y el desconocimiento general. Si se supiera que, dentro de una bodega, se pudiera implementar un robot que se dirija directamente a donde está un ítem específico, recogerlo y dejarlo en el área de despacho, seguramente se vería como una gran opción para evitar que el trabajador vaya en busca de estos objetos y dándole tiempo para designarle tareas más productivas.

Es por esto que nace la necesidad de un estudio en el que se explique a detalle la base, lógica, forma de aplicación y comportamiento de los algoritmos de evasión de obstáculos más utilizados. Con base en este estudio y como complemento de este, también se hace necesario un análisis entre los algoritmos presentados para ayudar a resaltar las virtudes y desventajas de cada uno. En conjunto esto podrá ser utilizado para desarrollar robots móviles orientados a cualquier tipo de ambiente según la necesidad, proliferando el desarrollo de estos no solo por grandes fábricas o empresas, sino también por pequeños emprendedores e incluso estudiantes a nivel experimental.

Ciertos proyectos se han gestado tomando como base la navegación autónoma. Uno de estos proyectos es del que se puede ver en la Figura 1, proyecto el cual consiste en la implementación de una red inalámbrica en combinación con una interfaz de MatLab para el control de un robot móvil autónomo. Este robot es conducido por cuatro ruedas independientes accionadas por motores DC, y su objetivo consiste en obtener información del ambiente donde se mueve el robot y enviar la información de manera inalámbrica al computador controlador. Cabe destacar que este robot móvil utiliza un algoritmo de evasión de obstáculos para el cálculo de su trayectoria a través de un lugar desconocido, trayectoria que no es calculada por el procesador del robot, sino que es generada dentro de la PC controladora, la cual recibe inalámbricamente las lecturas de los sensores acoplados al robot y realiza el cálculo dentro del propio MatLab, resultando en un sistema más rápida y menos costoso. (Tamre et al., 2018)

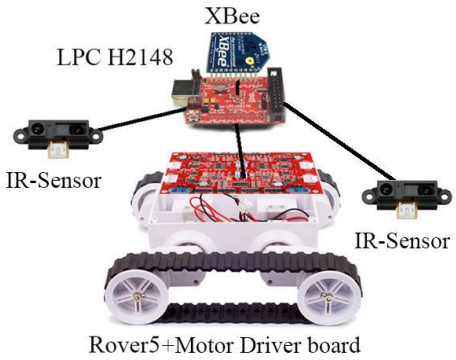



**Figura 1.** Robot móvil autónomo teleguiado
(Tame et al., 2017)

En la Figura 2 se puede observar un robot móvil diferencial autónomo con opción a ser tele-operado. Este robot desarrollado en México cuenta con dos sistemas para su navegación, el uno es una cámara digital que se utiliza cuando es teleguiado por un operador y el otro es un conjunto de 8 sensores ultrasónicos que son utilizados para la navegación autónoma. (Velázquez et al., 2020)


**Figura 2.** Robot móvil diferencial autónomo y tele-operado
(Velázquez et al., 2020)

El robot tiene como resultado una trayectoria simple y acortada entre los distintos punto de control que se le ordenó, trayectoria que puede ser observada en la Figura 3. (Velázquez et al., 2020)

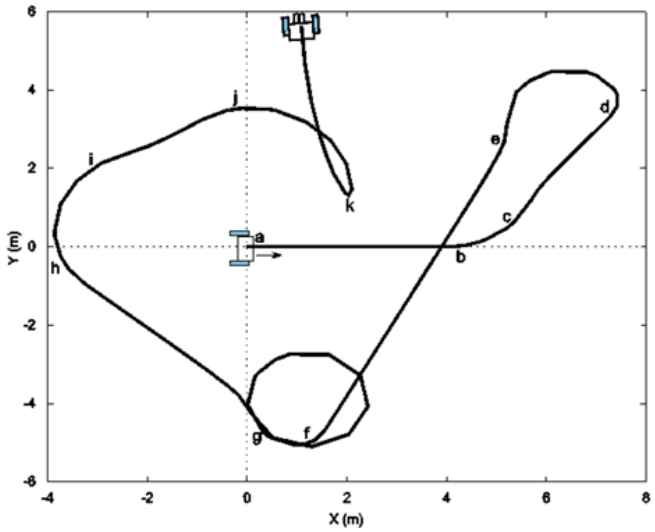



**Figura 3.** Trayectoria lograda por robot móvil autónomo y tele-operado (Velázquez et al., 2020)

Dentro de la Figura 4 se presenta un robot móvil autónomo diseñado para el movimiento dentro de espacios cerrados e irregulares, como los que se pudieran presentar dentro de oficinas con dos niveles. Este robot pretende aprovechar que la arquitectura moderna cada vez incluye más accesos para personas discapacitadas y usar estos accesos para su propia movilidad. El robot tiene como plataforma un segway (diferencial) y cuenta con dos sensores, los cuales son una cámara digital y un scanner láser 2D con un rango de distancia de 10m a 30 y un rango radial de 270° frente al sensor. Este robot planifica su trayectoria con una variante del algoritmo RRT (Rapid Random Tree). (Wang et al., 2017)

**Figura 4.** Robot móvil autónomo diferencial para ambientes irregulares (Wang et al., 2017)

Los resultados de este robot han sido satisfactorios, con un mapeo tridimensional de la zona bastante fiable y una generación de trayectoria bastante fluida y regular que se puede observar en la Figura 5. Nótese las ramificaciones en la planeación de la trayectoria, del cual se origina el nombre RRT. (Wang et al., 2017)

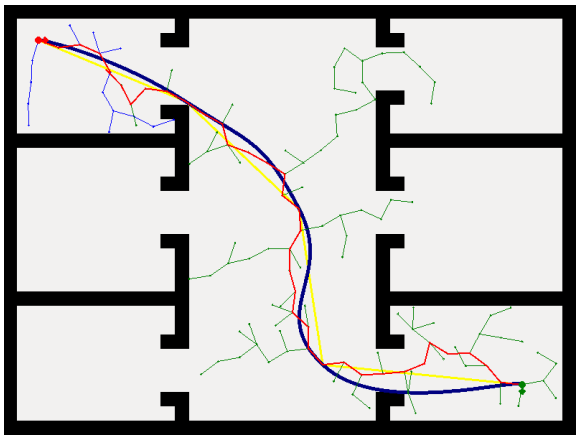


**Figura 5.** Trayectoria generada por la variante del algoritmo RRT (Wang et al., 2017)

Para entender mejor el proceso de planeamiento de ruta que realizan los algoritmos de evasión de obstáculos, es necesario definir ciertos términos como:

- **Navegación autónoma**

  La navegación autónoma es el movimiento inteligente planificado y ejecutado por un robot móvil. Se conforma por diversas tareas cumplidas secuencialmente y que cumplen la función de percibir el ambiente junto con sus obstáculos, planificar la ruta con respecto a dicho ambiente, y controlar el movimiento del robot móvil hasta su meta final. Todas estas tareas deben cumplirse correctamente para una correcta navegación autónoma. (Rodríguez, 2019)

- **Percepción del ambiente**

  Esta se realiza mediante uno o varios sensores que detectarán si es que existen objetos alrededor del robot y, de ser así, a qué distancia se encuentran estos objetos. Existen varios tipos de sensores que pueden realizar esta tarea, como los presentados a continuación:

  - **Sensor ultrasónico.-** basado en la velocidad del sonido, un sensor ultrasónico como el presentado en la Figura 6 emite una pulsación que, al rebotar contra la superficie de un objeto, regresa a un receptor. La distancia al objeto es calculada según el tiempo que demora la pulsación en volver. (Pérez, 2019)


**Figura 6.** Sensor ultrasónico HS-SR04

- **Sensor infrarrojo.-** sensor óptico que mide la distancia hacia un objeto usando un emisor infrarrojo y un receptor que detecta el ángulo de incidencia con el que llega la luz infrarroja tras chocar con un objeto frente al sensor. Este tipo de sensor, del cual se puede ver un ejemplo en la Figura 7, calcula la distancia al objeto mediante triangulación. (Pérez, 2019)

**Figura 7.** Sensor Infrarrojo GP2Y0A21YKF0

- **Sensor de distancia láser.-** este es un sensor con un principio similar a los sensores infrarrojos, su diferencia es que en lugar de una luz infrarroja, lo que se emite es un láser que viaja, choca contra el objeto y regresa a un receptor. Debido a que es un láser lo que se emite, la distancia es calculada basándose en la velocidad del láser (la misma que la velocidad de la luz), y el tiempo que transcurre entre la emisión del láser y el retorno del mismo. Este tipo de sensores puede ser útil incluso en largas distancias de hasta kilómetros, pero pierde efectividad a distancias muy cortas debido a que es el retorno de la luz en distancias de menos de 15 cm es casi instantáneo, por lo que es difícil tomar con precisión el tiempo transcurrido. De este tipo de sensores existe uno en específico que, en lugar de ser solo un

láser lineal, es un scanner láser que analiza un espacio radial de hasta 270° frente de sí. En la Figura 8 se puede ver un modelo de sensor scanner láser con la capacidad de detectar objetos hasta 30m de distancia en un rango de 270°. (PCE-instruments, 2021)

**Figura 8.** Sensor scanner láser Hoyuko UTM-30LX

- **Planificación de ruta**

  La planificación de ruta es el resultado final de los algoritmos de evasión de obstáculos y cumplen con la misión de guiar al robot móvil desde el punto A hasta el punto B con el menor consumo de energía posible, esto mientras se esquiva cualquier tipo de obstáculo que pueda encontrarse en el entorno. (Aradena, 2020)

  La planificación de ruta se puede clasificar en planificación global y planificación local.

  - **Planificación global**

    La planificación global es la planeación basada en un ambiente previamente establecido. Los algoritmos basados en una planeación global son ciertamente más efectivos respecto a la ruta a seguir dentro de un ambiente más complicado, pero carecen de capacidad de respuesta frente a obstáculos dinámicos o desconocidos, y ambientes cambiantes. (Álvarez y Flor, 2020)

  - **Planificación local**

    La planificación local se caracteriza por su nula necesidad de información previa acerca del ambiente donde se moverá el

robot. Alimentados por información recogida por sensores, los algoritmos basados en este tipo de planificación son capaces de reaccionar ante cualquier tipo de obstáculo en un ambiente desconocido. La desventaja es la eficiencia, debido a que no se conoce nada sobre el ambiente, la ruta planeada no siempre resulta ser eficiente ni en distancia ni en tiempo. (Aradena, 2020)

- **Algoritmo de evasión de obstáculos**

  Un algoritmo de evasión de obstáculos funciona de la misma manera que lo haría cualquier otro algoritmo, contando con un estructura general la cual se repetirá una y otra vez hasta que se alcance el objetivo final.

  - **Entradas:** se conforman por sensores (planificación local), mapas (planificación global) y el objetivo final a alcanzar por el robot móvil.
  - **Proceso:** está conformado por todas las operaciones y cálculos necesarios por los que pasa la información de entrada para llegar a la salida deseada. En el caso de un algoritmo de evasión de obstáculos, el proceso toma toda la información, ya sea de parte de sensores o de un mapa precargado, y realiza el planeamiento de ruta antes mencionado.
  - **Salidas:** en este caso se conforma solamente por la dirección angular deseada en la cual deberá moverse el robot móvil.

- **Robot móvil de tracción diferencial**

  Los robots móviles son robots que se encuentran en constante movimiento, por lo tanto, su eje de referencia y posición cambian en el tiempo. Esto conlleva a la necesidad de un reconocimiento continuo del ambiente externo al robot móvil además de un cálculo lo más preciso posible de la posición actual donde se encuentra el robot. Lo primero se satisface con la implementación de sensores o mapas previamente cargados, y lo segundo lo hace con ayuda de la odometría o un sistema GPS, aunque cabe recalcar que la odometría es lo más común y barato. (Hernández et al., 2017)

  Un robot móvil de tracción diferencial es un robot no holonómico que logra su movimiento gracias al giro de dos ruedas colocadas en forma paralela y a una distancia igual con respecto al centro del robot. Su nombre viene del hecho que la diferencia de velocidades angulares de

las ruedas permite al robot avanzar, retroceder y girar hacia los lados o sobre su propio eje.

La ventaja del robot móvil de tracción diferencial es su simpleza y maniobrabilidad ya que con solo dos ruedas, que se traducen en dos motores, es capaz de realizar cambios de dirección abruptos que serían imposibles para robots móviles de cuatro ruedas. Por lo tanto, se puede denominar a este tipo de robot como versátil y eficiente.

En la Figura 9 se puede observar un ejemplo de robot móvil de tracción diferencial. Cabe destacar que, a pesar de que se puede observar una tercera rueda, esta solo cumple la función de apoyo y no es dirigida ni accionada por un actuador.

**Figura 9.** Robot móvil de tracción diferencial
(Hernández et al., 2017)

Tomando como motivación los conocimientos sobre algoritmos de evasión de obstáculos y su gran potencial para ser implementados, es que se plantea como objetivo principal para este proyecto el siguiente objetivo principal:

- Codificar algoritmos de evasión de obstáculos y aplicarlos, mediante una simulación, a un robot móvil de tracción diferencial dentro de un ambiente cerrado con obstáculos para obtener información del comportamiento de los algoritmos.

Para la consecución de este objetivo principal, es necesario segmentarlo en objetivos más específicos cuyos cumplimientos apoyarán individualmente, y de una manera sinérgica, a la meta final. Estos objetivos específicos son:

- Calcular e implementar el modelo matemático y la cinemática inversa de un robot móvil de tracción diferencial.
- Investigar la lógica de los algoritmos de evasión de obstáculos siguientes:
    - RRT (Rapid Random Tree)
    - RRT* (Rapid Random Tree Star)
    - Potential Field Path Planning
    - VFF (Virtual Force Field)
    - VFH (Vector Field Histogram)
- Implementar los algoritmos mencionados para el guiado del modelo de robot móvil de tracción diferencial.
- Analizar el comportamiento de los algoritmos y su rendimiento para el guiado de un robot móvil de tracción diferencial.

El proceso a seguir en este proyecto, en forma general, se describe de la siguiente manera: “Se investigará la lógica de los algoritmos de evasión de obstáculos para luego crear un código que se aplicará a un modelo de robot móvil de tracción diferencial. Con todos y cada uno de los algoritmos que se aplicarán, el robot deberá ser capaz de llegar del punto inicial A hasta el punto final B mientras sortea los obstáculos presentados a lo largo del mapa. Todo esto se ejecutará dentro de una simulación en tiempo real en MatLab donde se podrá observar el movimiento del robot y su comportamiento al ser guiado por los diferentes algoritmos.”

# 2. METODOLOGÍA

## 2.1. METODOLOGÍA MECATRÓNICA

Para el desarrollo de este proyecto se seguirá la Metodología en V, la cual está ilustrada en la Figura 10 y describe el procedimiento que se realizó con cada uno de los algoritmos, relacionando sus etapas de análisis del algoritmo, diseño, codificación, compilación, verificación, fase de prueba y verificación.

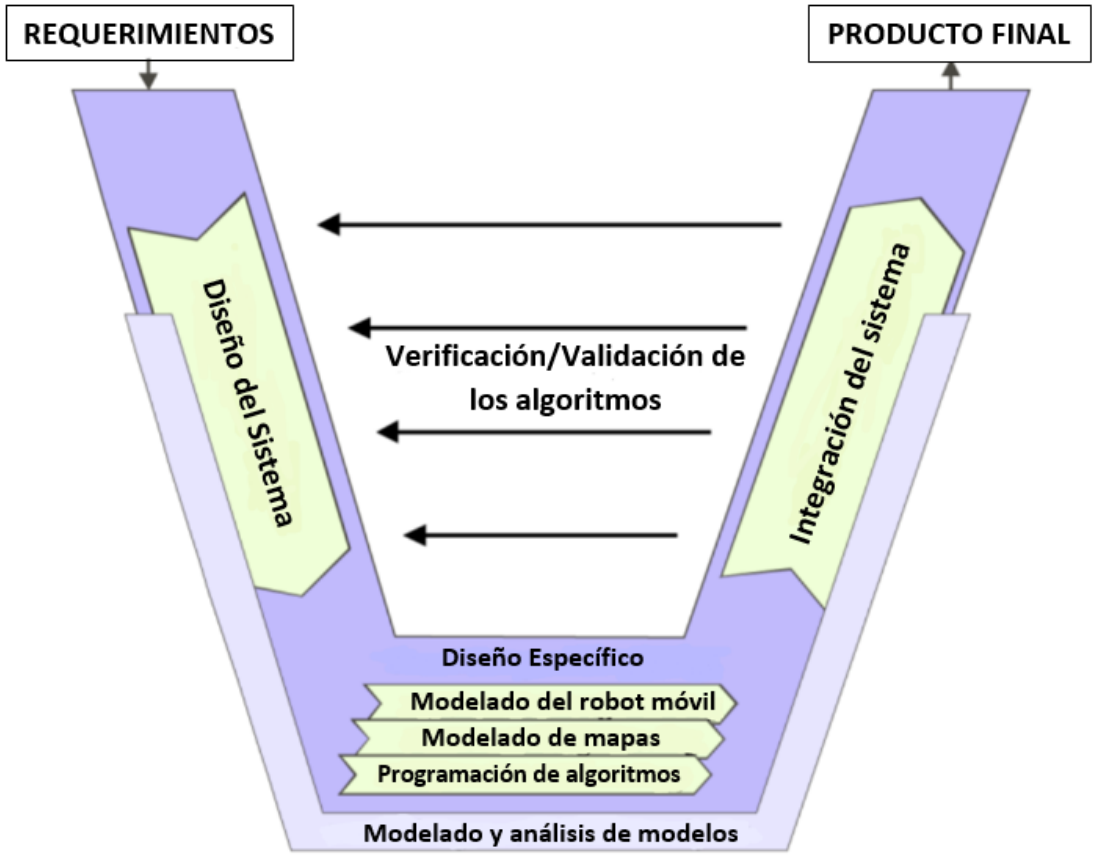



**Figura 10.** Metodología en V (Colomer, 2013).

## 2.2. ANÁLISIS DE REQUERIMIENTOS DEL SISTEMA

A continuación se presentan ciertas características necesarias para cumplir con el estudio y análisis de los algoritmos de evasión de obstáculos.

### 2.2.1. MODELO DE ROBOT MÓVIL DE TRACCIÓN DIFERENCIAL

El modelo debe considerar dos ruedas colocadas en el mismo eje a una distancia $L/2$ del punto central P del robot. Ambas ruedas serán de un radio r. La velocidad angular $\dot{\theta}_L$ y $\dot{\theta}_R$ de las ruedas izquierda y derecha respectivamente, serán las que determinen la velocidad angular W y velocidad lineal V del robot móvil. (Tiep et al., 2018)

### 2.2.2. MAPAS DE PRUEBA

- Los mapas de prueba deben simular un ambiente cerrado con muros a su alrededor y obstáculos dentro de él.
- Los mapas no deben ofrecer una vía directa sin obstáculos entre un punto A y un punto B que se le asignase al robot móvil.

### 2.2.3. ALGORITMOS DE EVASIÓN DE OBSTÁCULOS

Los algoritmos de evasión de obstáculos deben ser capaces de llegar desde un punto A hasta un punto B, ambos colocados a extremos opuestos de un mapa con un solo obstáculo central.

## 2.3. MODELADO DEL ROBOT MÓVIL DE TRACCIÓN DIFERENCIAL

Suponiendo un robot móvil de tracción diferencial, se contempla ciertos parámetros del robot como son la velocidad angular y lineal de las ruedas, el radio de las mismas y la distancia entre las ruedas del robot. Estos parámetros se pueden observar en la Figura 11.

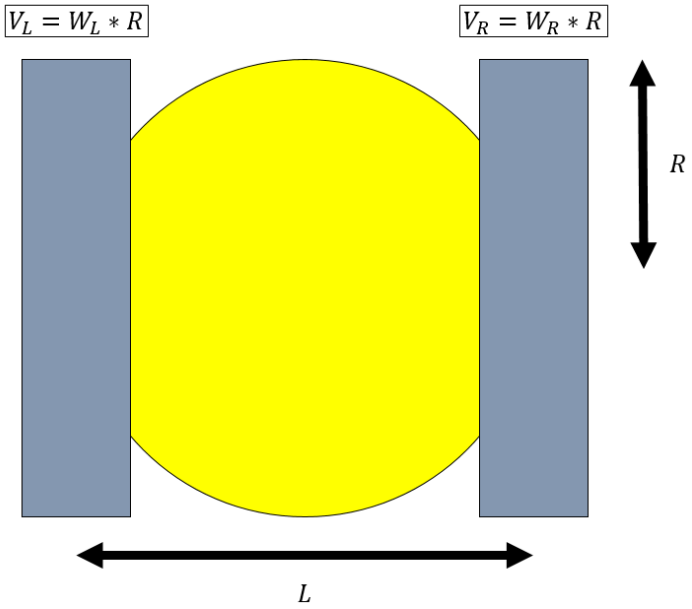



**Figura 11.** Parámetros robot móvil de tracción diferencial.

Como su nombre lo indica, un robot móvil de tracción diferencial, puede cambiar su posición angular mediante la diferencia de las velocidades de sus ruedas, como se puede observar en la Figura 12.

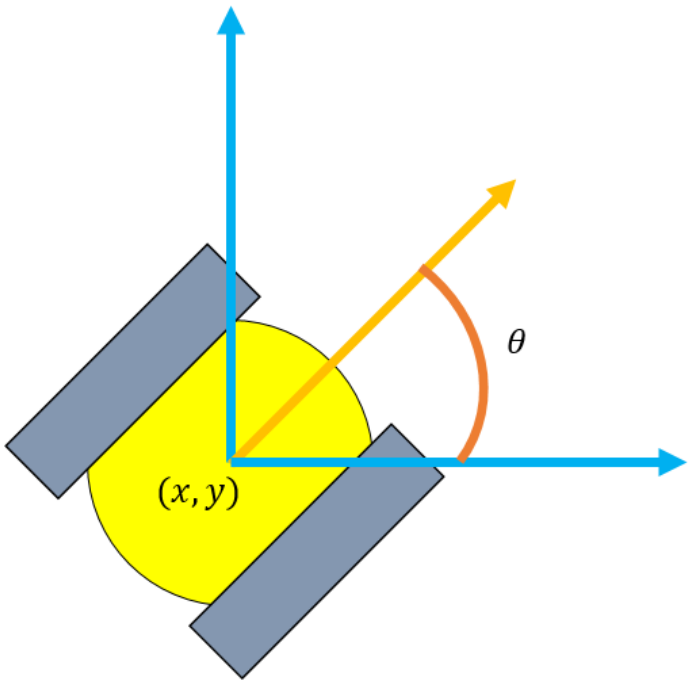



**Figura 12.** Movimiento angular producido por la diferencia de velocidades de las ruedas.

La velocidad del robot móvil producida por las velocidad individuales de cada rueda se puede calcular mediante la siguiente ecuación.

$$V = \left(\frac{V_R + V_L}{2}\right) * R \quad [1]$$

La velocidad angular del robot se puede calcular mediante la siguiente ecuación.

$$\dot{\theta} = W = \frac{R}{L} * (V_R - V_L) \quad [2]$$

Donde (para ecuación 1 y 2):

$V$: Velocidad lineal del robot en [m/s]
$W$: Velocidad angular del robot en [grados/s]
$V_R$: Velocidad lineal de la rueda derecha del robot en [m/s]
$V_L$: Velocidad lineal de la rueda izquierda del robot en [m/s]
$R$: Radio de las ruedas del robot en [m]
$L$: Diámetro del robot (distancia de rueda a rueda del robot) en [m]

Simplificando, se expresa la velocidad lineal como la variación de posición del robot en el eje X & eje Y, y la velocidad angular como la variación de la posición angular del mismo.

$$\begin{cases} \dot{x} = \frac{R}{2} * (V_R + V_L) * \cos(\theta) \\ \dot{y} = \frac{R}{2} * (V_R + V_L) * \text{sen}(\theta) \\ \dot{\theta} = \frac{R}{L} * (V_R - V_L) \end{cases} \quad [3]$$

Con las ecuaciones expresadas en 3, se llega a lo que se denomina **cinemática directa**, donde las velocidades lineal y angular de un robot son calculadas a partir de las velocidades individuales de cada rueda. Sin embargo, para propósitos de este proyecto, se utiliza la **Cinemática Inversa** para la navegación del robot, la cual calcula las velocidades de las ruedas a partir de las velocidades angular y lineal deseadas para el robot.

Aislando $V_R$ en la ecuación 1.

$$V_R = \frac{(2V) - (RV_L)}{R} \quad [4]$$

Aislando $V_L$ en la ecuación 1.

$$V_L = \frac{(2V) - (RV_R)}{R} \quad [5]$$

[3] en [2]

$$V_R = \frac{2V+WL}{2R} \quad [6]$$

[4] en [2]

$$V_L = \frac{2V-WL}{2\mathrm{R}} \quad [7]$$

Con las ecuaciones 5 y 6 tenemos la cinemática inversa que servirá para tener un mejor control del robot ya que solo se le ordenará las velocidades angular y lineal deseadas.

### 2.3.1. SIMULACIÓN DE ROBOT MÓVIL DE TRACCIÓN DIFERENCIAL

Dado que la simulación se realiza en el Software MatLab y su extensión Simulink, la cinemática inversa del robot de tracción diferencial es introducida dentro de una función de MatLab, como se puede ver representado en la Figura 13, la cual tendrá como entradas la Velocidad angular (W) y velocidad lineal (V) deseadas para el robot, y como salidas la velocidad lineal deseadas para cada rueda, esto usando las ecuaciones 3 y 4. Para este caso, se simulará un robot de 20 cm de diámetro (L) con ruedas de radio de 5 cm (R).

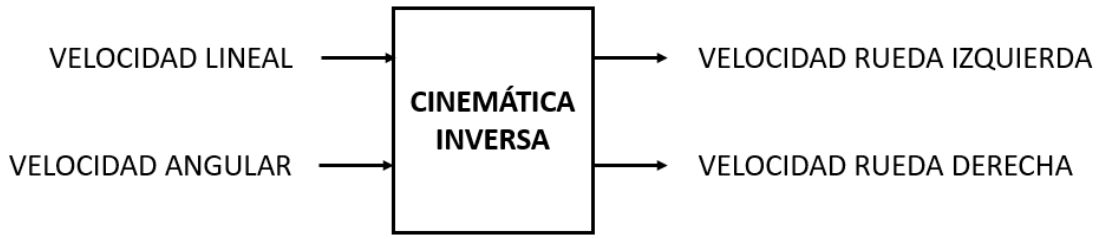



**Figura 13.** Bloque de cinemática inversa.

Para la simulación del movimiento del robot dentro de un mapa de pruebas, se utiliza la librería *Mobile Robotics Simulation Toolbox* y en específico el bloque *Robot Visualizer*, esta librería es de libre uso fue creada por estudiantes en Octubre del 2020. Esta librería utiliza la posición (x,y) del robot junto con su posición angular Θ para posicionarlo dentro de un mapa, la definición de este mapa se presentará en una siguiente sección.

Debido a que es necesario tener las posiciones x, y & Θ del robot para la simulación, nos ayudamos del grupo de ecuaciones [3] junto con las ya

calculadas velocidades de las ruedas, y agregamos una etapa integrativa en cada una de las salidas. En la Figura 14 se muestra una representación de la función de MatLab utilizada para obtener las posiciones del robot.

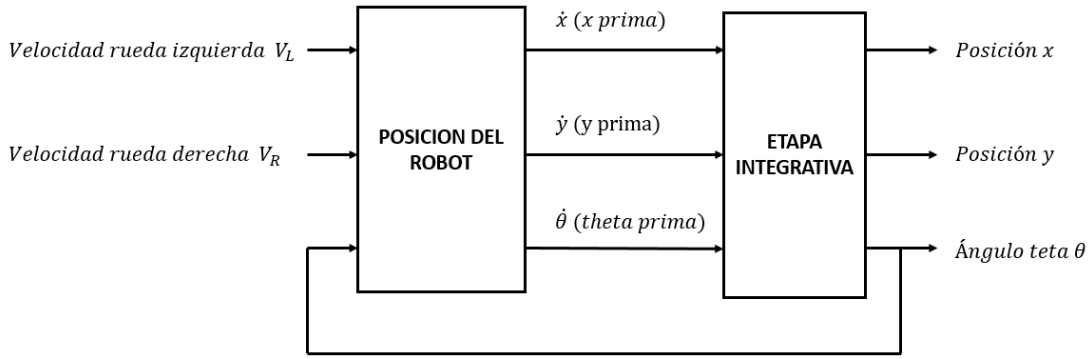



**Figura 14.** Bloques para obtener la posición del robot móvil.

Finalmente, se agregan bloques para definir las posiciones iniciales y se añade al final del sistema el bloque *Robot Visualizer*, el cual nos mostrará finalmente al robot móvil dentro la simulación en un mapa sencillo cerrado. En la Figura 15 se puede ver la integración de todos los bloques.

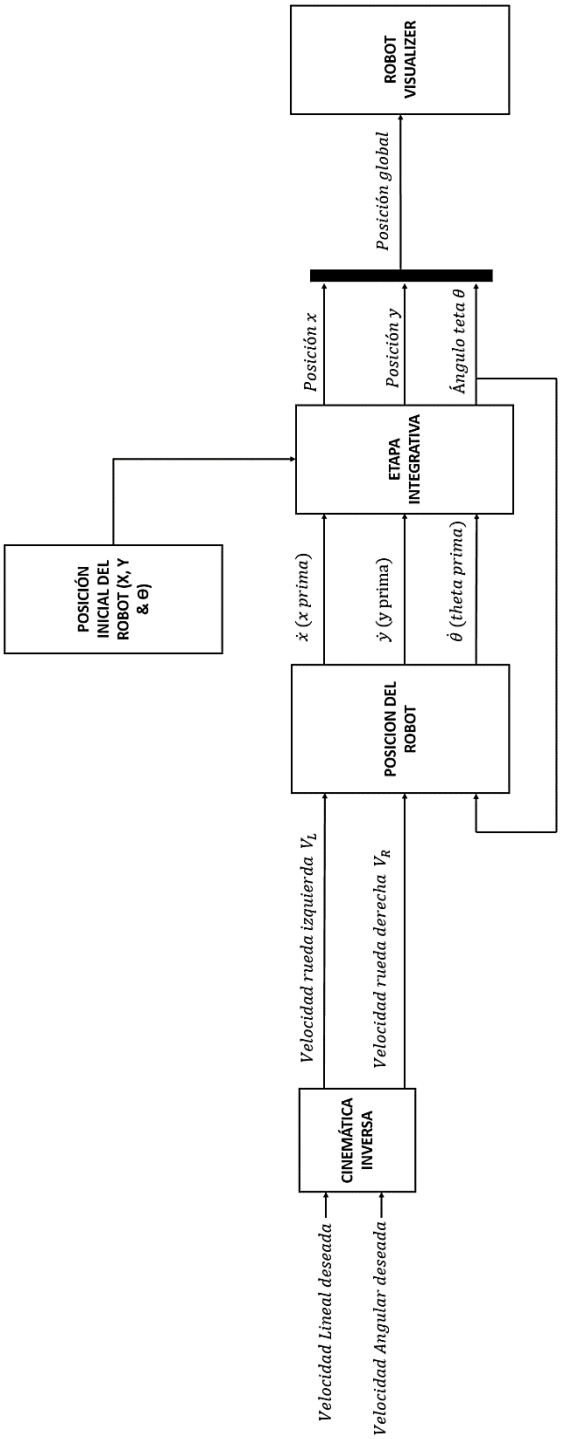



**Figura 15.** Integración del sistema para navegación y simulación del robot móvil.

El resultado de este sistema de bloques es la visualización del robot tal y como se ve en la Figura 16.

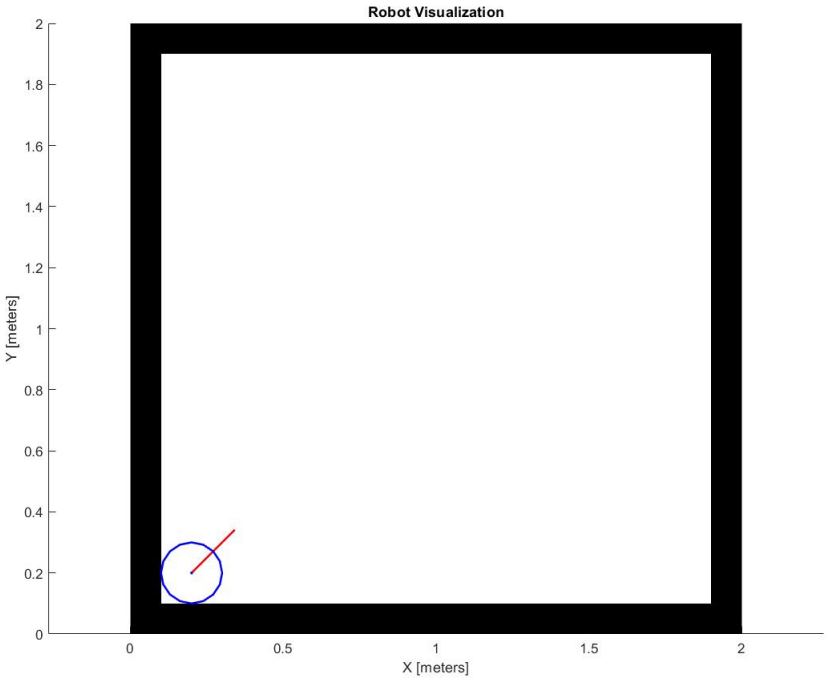



**Figura 16.** Visualización del robot en una posición inicial (0.2, 0.2) con un ángulo inicial de 45°.

## 2.4. MODELADO DE MAPAS DE PRUEBA

El modelado de mapas se realiza mediante el comando *occupancyMap* de Matlab, el cual sirve para crear una malla de espacios de un ancho, alto y resolución determinados. En la Figura 17 se puede observar un mapa de 20 espacios de alto x 20 espacios de ancho con una resolución de (1/10) metros por espacio, lo que devendrá en un mapa de 2x2 metros.

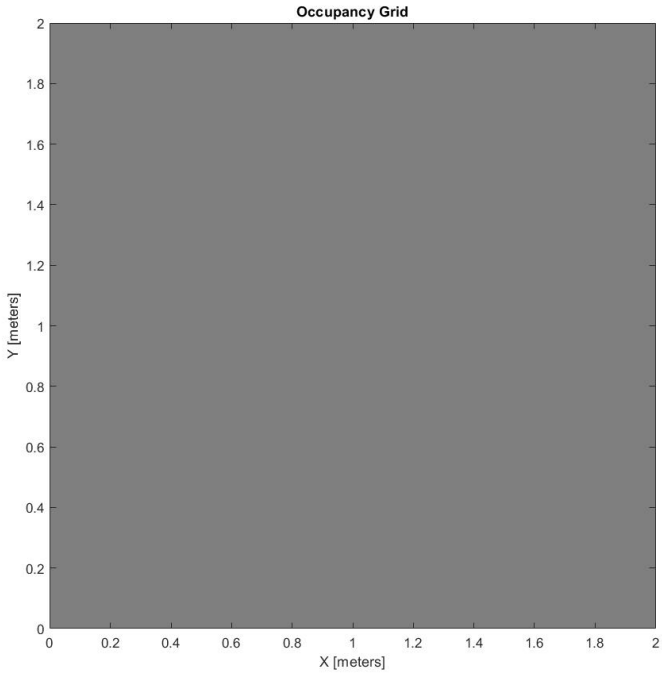



**Figura 17.** Mapa de 2x2 metros generado con occupancyMap.

Como se puede observar en la Figura 17, las cuadrículas del mapa tienen un color grisáceo que se debe a que su valor de ocupación es de 0.7 por defecto. Para poder definir cuales espacios se encuentran vacíos y cuales ocupados se utiliza el comando *updateOccupancy*. En la Figura 18 se puede observar el mapa presentado en la Figura 17 con los espacios (0.1, 0.1) y (2, 0.1) declarados como espacios ocupado y vació respectivamente.

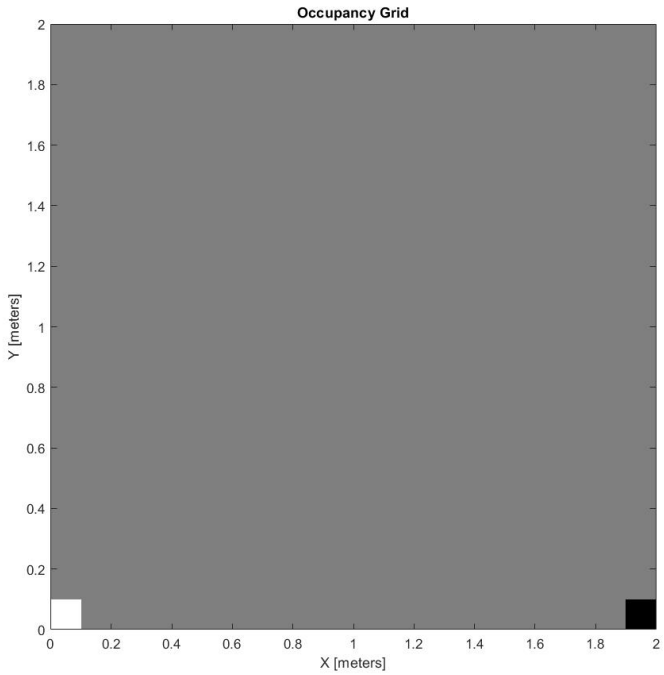



**Figura 18.** Mapa con valores de ocupación actualizados.

De esta forma se realizan dos distintos mapas. El primero de ellos es el que se puede ver en la Figura 19, el cual es un mapa cerrado sencillo de 2x2 metros con un obstáculo cuadrado de 40 cm de lado que será utilizado para comprobar el funcionamiento correcto de los algoritmos.

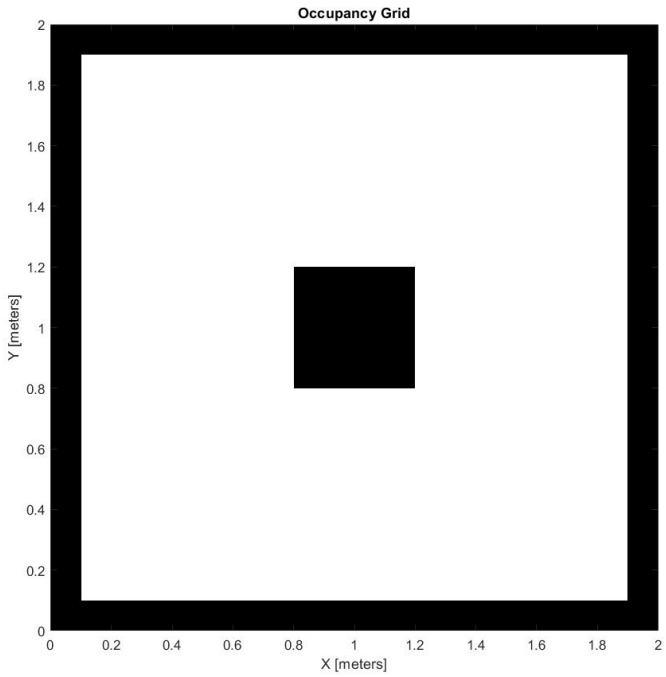



**Figura 19.** Mapa básico de comprobación de algoritmos.

En la Figura 20 se puede observar el segundo mapa, el cual es un mapa cerrado de 5x5 metros con varios obstáculos dentro de él que se utilizará para analizar el comportamiento de los algoritmos tras comprobar que funcionan correctamente en el primer mapa.

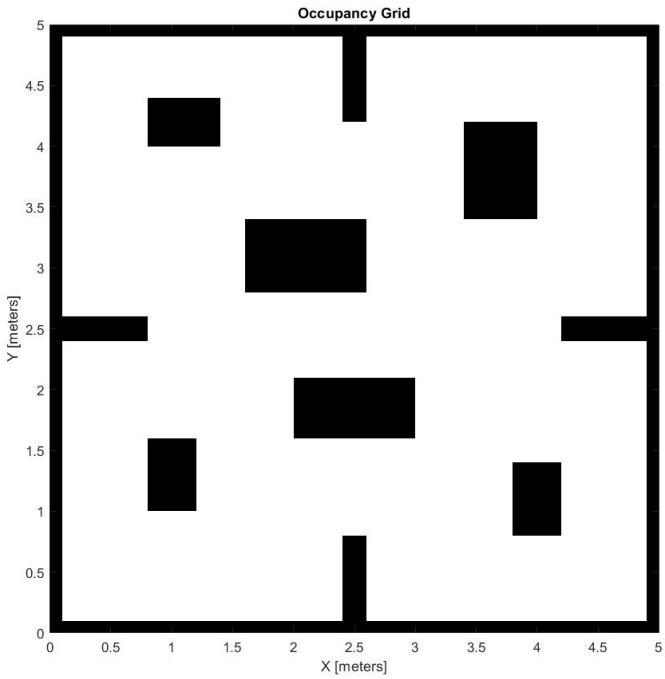



**Figura 20.** Mapa de prueba de comportamiento de algoritmos.

## 2.4.1. SIMULACIÓN DE MAPAS

Para la simulación de mapas, esta es más sencilla ya que se utiliza el mismo bloque *Robot Visualizer*, resultando en algo conveniente ya que se combina muy fácilmente con la simulación del robot móvil de una manera dinámica. El mapa se añade al bloque simplemente colocando su nombre en el mismo. En la Figura 21 se puede observar la parte donde se añade el mapa creado. Nótese que también se tiene campos para añadir el radio del robot (0.1 en este caso), detección de objetos y sensores LIDAR, los cuales son de vital importancia para el funcionamiento de los algoritmos.

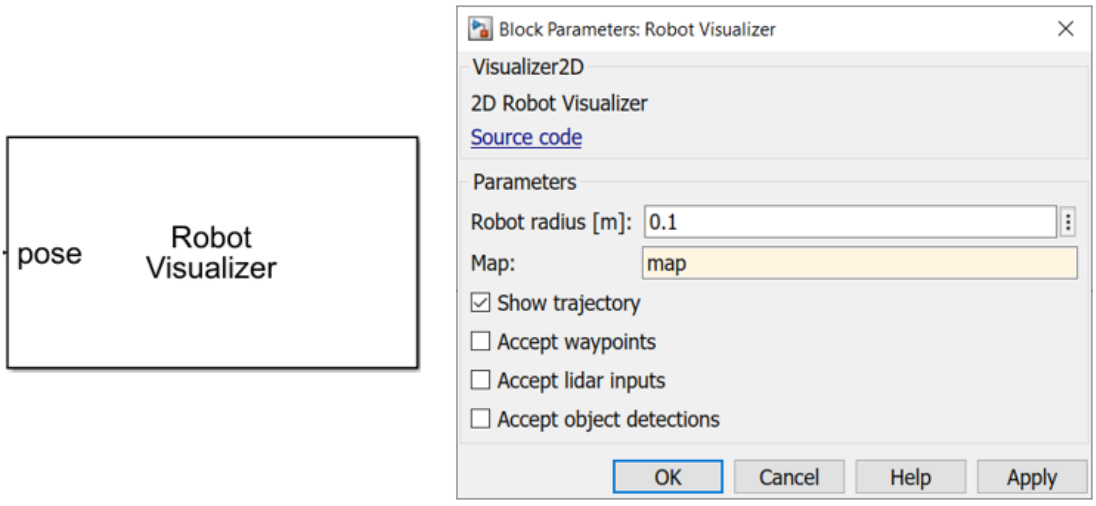



**Figura 21.** Bloque Robot Visualizer.

En la Figura 22 se puede observar al mismo robot de la Figura 16 dentro de los mapas de las Figuras 22 y 23 respectivamente.

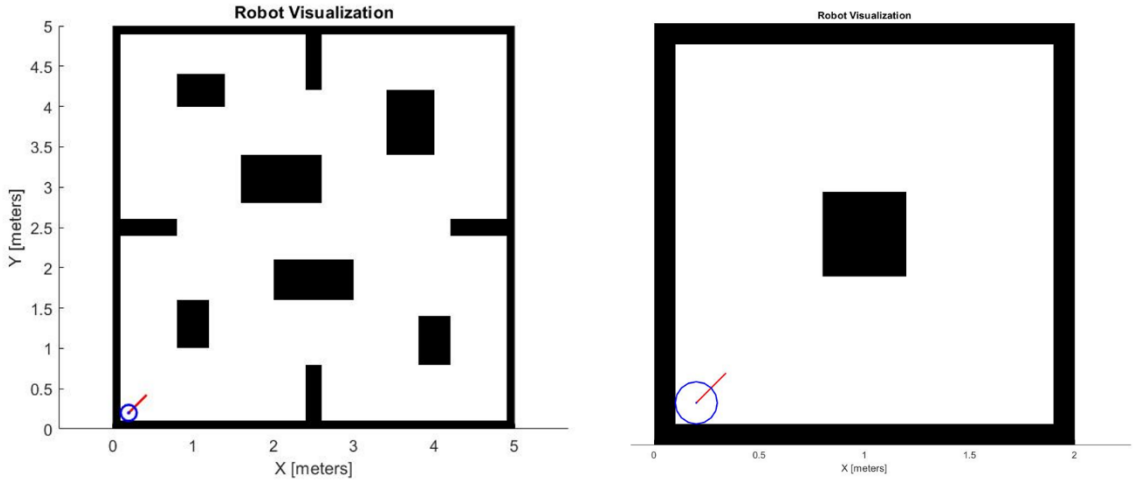



**Figura 22.** Robot móvil dentro de mapas de prueba.

## 2.5. PROGRAMACIÓN DE ALGORITMOS

Para el desarrollo de los algoritmos, primero se analiza la lógica de los mismos para después desarrollar un código que será introducido en la forma de una función de MatLab. Los algoritmos siempre tendrán como salidas la dirección deseada en la cual debe encaminarse el robot, y la velocidad a la que se desplazará el mismo.

### 2.5.1. RRT (RAPIDLY RANDOM TREE)

El algoritmo RRT es un algoritmo que emplea la aleatoriedad para definir un camino factible entre un punto inicial del mapa hasta un punto final del mismo. Esta aleatoriedad lleva a la unión de nodos a lo largo del mapa, encadenados uno tras de otro, de forma que la estructura que forma el algoritmo se asemeja a la de un árbol y sus ramas. (Wei & Ren, 2018)

Cabe destacar que, debido a la naturaleza aleatoria de este algoritmo, la exploración de mapa se produce de una manera uniforme y no objetiva, lo que produce que sea más lento el alcanzar el objetivo final. (Zhang et al., 2018)

### 2.5.1.1. Lógica RRT

La lógica del algoritmo RRT, como se ha mencionado, tiene como método de exploración la aleatoriedad.

El proceso se inicia siempre con el punto de partida del robot que se convierte en el nodo inicial. Luego de esto, se crea un punto aleatorio dentro del mapa, el cual busca el nodo existente más cercano y lo elige como nodo Padre, para el inicio, el nodo inicial es el más cercano ya que es el único creado, pero para siguientes nodos, se buscará siempre el más cercano. En la Figura 23 se puede ver un ejemplo de esto.

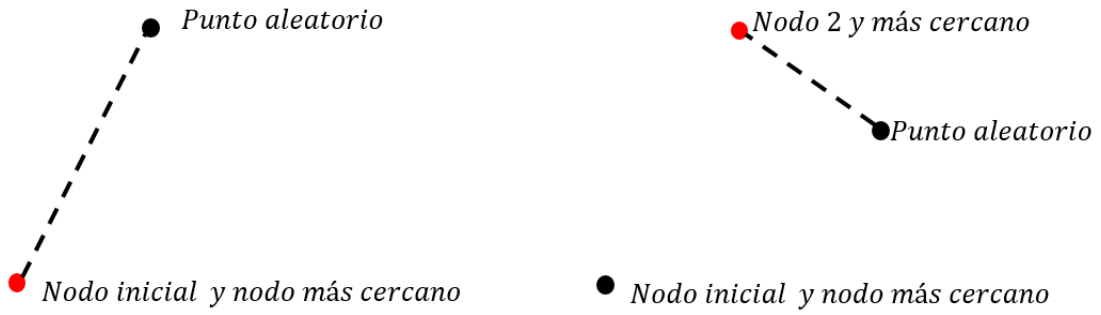



**Figura 23.** Elección del nodo cercano.

Dentro de una línea creada entre el punto aleatorio y el nodo Padre, se revisa si el punto aleatorio se encuentra dentro de una distancia máxima llamada Paso. Si se encuentra dentro de esta distancia, un nodo se crea donde se generó inicialmente el punto, sino, se coloca el nodo al final del Paso a lo largo de la línea creada. En la Figura 24 se puede ver un ejemplo de estos dos casos.

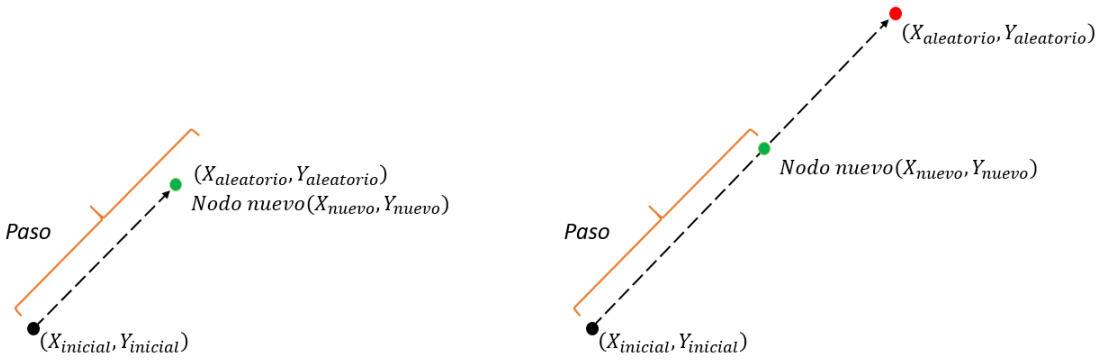



**Figura 24.** Creación de un nuevo nodo. Izquierda: cuando el nodo se encuentra a una menor o igual distancia del paso. Derecha: cuando el nodo se encuentra a una mayor distancia del Paso

El proceso se repite una y otra vez hasta que se llega al objetivo. El aumento progresivo de nodos hace que su estructura se asemeje a un árbol con sus ramas.

### 2.5.1.2. Implementación RRT

La lógica del algoritmo se puede ver implementada de una manera más clara en la Figura 25, en la cual se puede ver el diagrama de flujo mediante el cual se aplica el algoritmo RRT. Nótese que el algoritmo realiza una exploración del mapa mediante la lógica presentada anteriormente y, cuando se llega al objetivo, se empieza a declarar un vector solución uniendo cada nodo con su respectivo padre hasta llegar a la posición inicial.

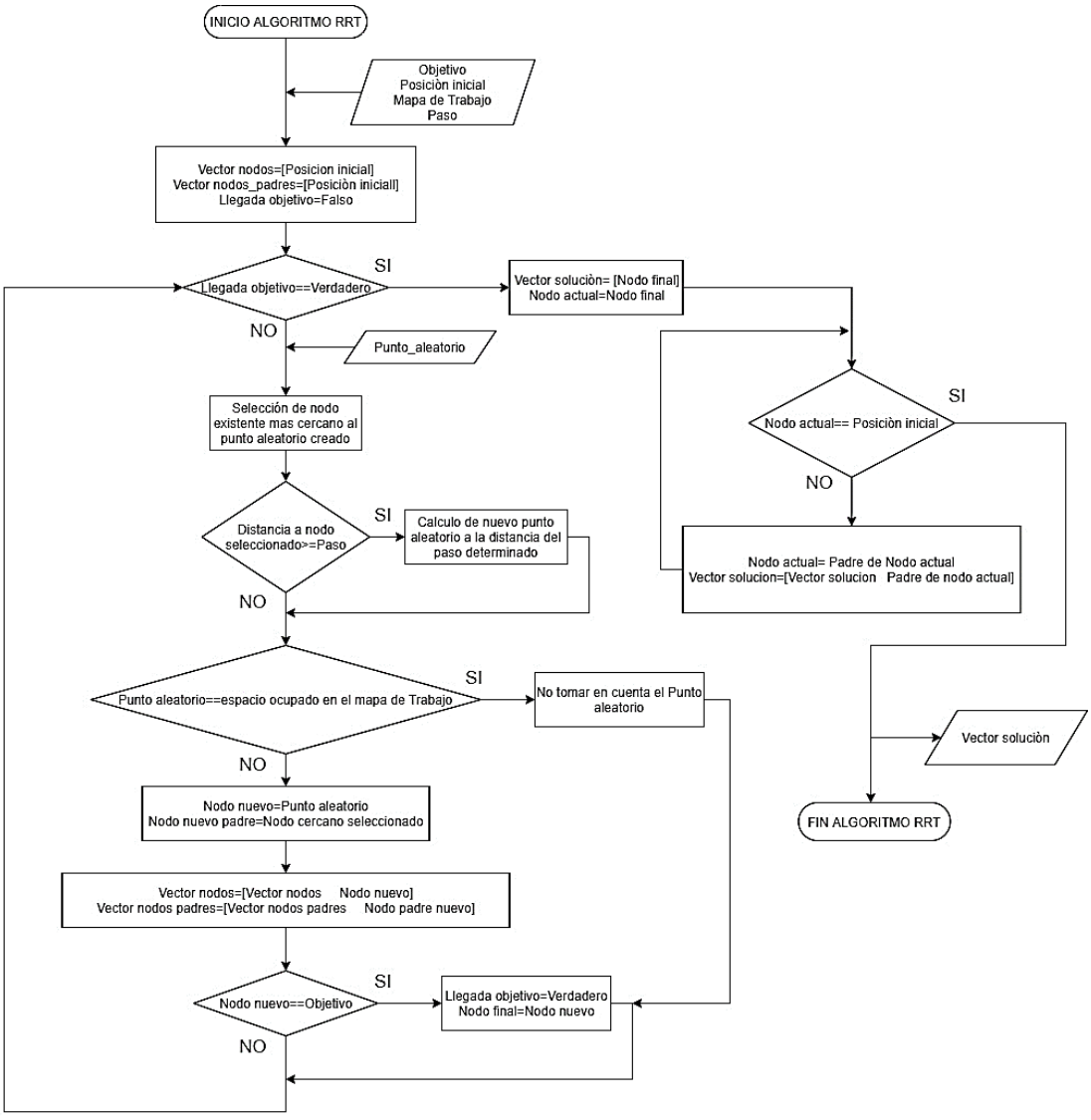



**Figura 25.** Diagrama de Flujo Algoritmo RRT.

Como se puede ver al final del diagrama de flujo de la Figura 25, como dato de salida se tiene al **vector solución**, el cual es en sí mismo la ruta planificada para el robot. El vector solución es usado posteriormente para guiar al modelo de robot móvil dentro de una simulación en Simulink. En la Figura 27 se puede ver la planificación realizada por el algoritmo dentro del mapa mostrado en la Figura 19. Cabe destacar que, con el objetivo de evitar choques, la exploración del algoritmo se realiza tomando en cuenta el radio de 0.1 m del robot, de forma que los puntos generados dentro de un área que pudiera ocasionar un choque no son tomados en cuenta por el algoritmo. Esto se logra creando un mapa adicional de trabajo que es donde verdaderamente se realiza la exploración. En la Figura 26 se puede ver la comparación entre estos dos mapas.

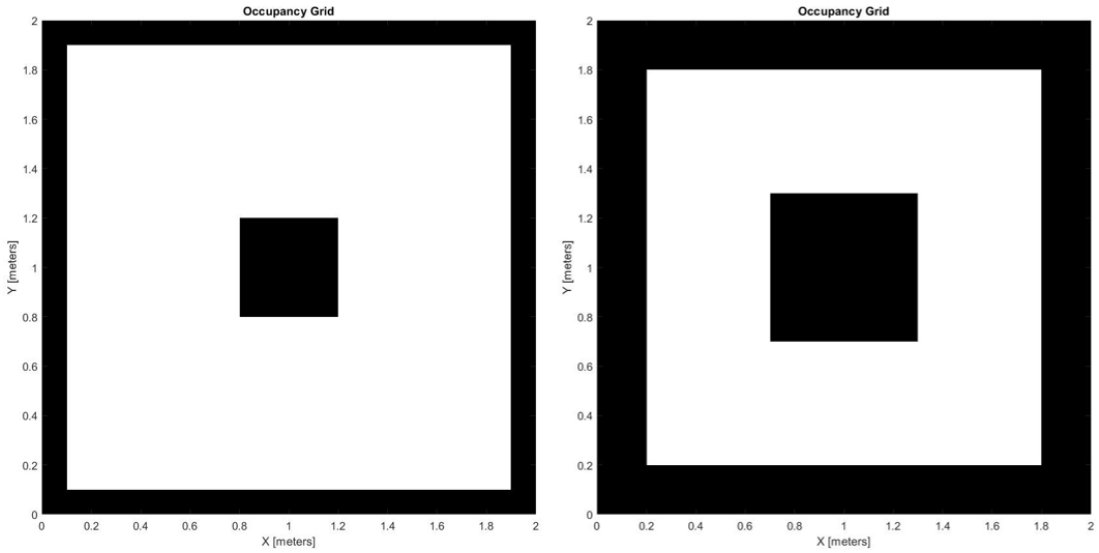



**Figura 26.** Mapa real VS Mapa de trabajo

Como se puede ver, el mapa de trabajo toma en cuenta la restricción del radio del robot, lo que permite que la exploración realizada por el algoritmo sea correcta y no pase por puntos que pudiesen ocasionar choques.

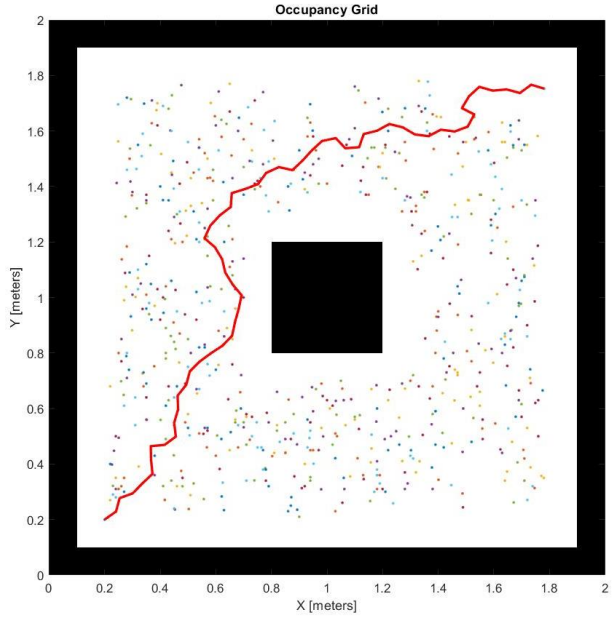



**Figura 27.** Planificación de ruta realizada por algoritmo RRT en mapa básico de comprobación.

### 2.5.1.3. Simulación RRT

Para la simulación del algoritmo RRT, se usa como ayuda adicional otro bloque de la librería *Mobile Robotics Simulation Toolbox*, el cual se llama *Object Detector*. Este bloque es utilizado debido a su conveniente funcionamiento, ya que al darle la posición actual del robot junto con la posición de un objeto x, el bloque automáticamente devuelve la distancia y posición angular respecto al robot a la que se encuentra el objeto.

Usando este principio y tomando como recurso las posiciones del ya declarado vector solución, se procede a guiar al robot móvil desde la posición inicial, a través de todas las posiciones del vector, hasta llegar al objetivo final.

En la Figura 28 se puede ver el bloque *Object Detector*. Nótese que en este bloque se debe definir el mapa en el cual se moverá el robot móvil, este mapa siempre debe ser el mismo que se utiliza en el bloque *Robot Visualizer*.

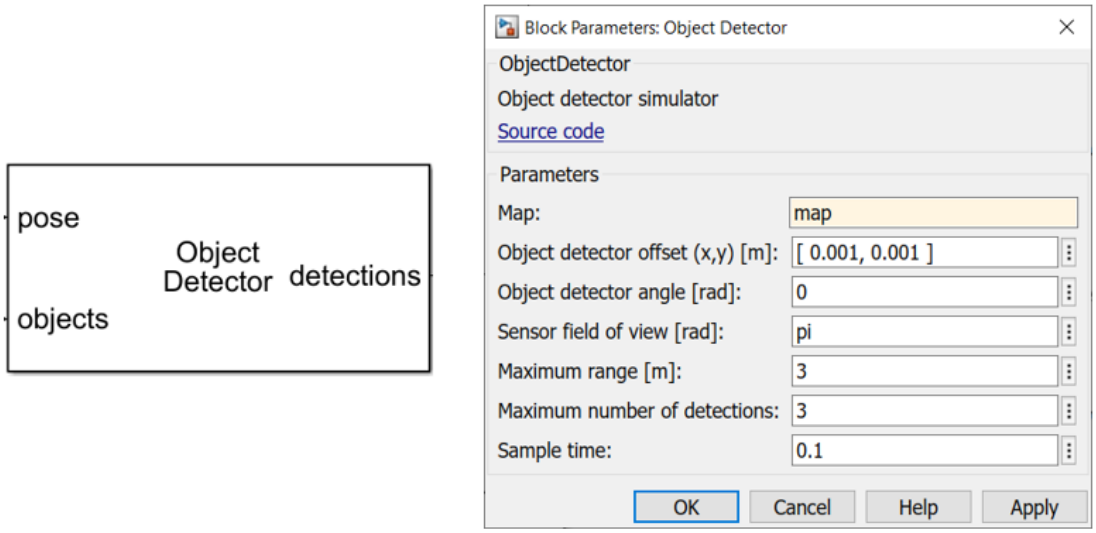



**Figura 28.** Bloque Object Detector.

El movimiento desde una posición hasta otra se realiza mediante una máquina de estados, que calcula simultáneamente velocidad lineal y velocidad angular deseadas para el robot, las cuales pasan a ser las entradas para la simulación del robot móvil como se puede ver en la Figura 15. Las velocidades lineal y angular son calculadas mediante las siguientes condicionales, las cuales aseguran una navegación precisa junto con una aproximación final al objetivo final suave y controlada.

$$V = velocidad\ lineal = \begin{cases} 0.08\left[\frac{m}{s}\right], & d \geq 0.025 \\ d * 0.04\left[\frac{m}{s}\right], & 0.025 > d \geq 0.005 \\ d * 0.001\left[\frac{m}{s}\right], & d > 0.005 \end{cases} \quad [8]$$

$$W = velocidad\ angular = \begin{cases} 2 * \theta\ \left[\frac{grados}{s}\right], & \theta = 0 \\ 0\ \left[\frac{grados}{s}\right], & \theta \neq 0 \end{cases} \quad [9]$$

Donde (para casos 8 y 9):

$V$ : Velocidad lineal del robot en [m/s]
$W$: Velocidad angular del robot en [grados/s]
$d$ : Distancia desde la posición actual del robot al objetivo o posición deseada en [m]
$\theta$ : Posición angular del objetivo o posición deseada respecto al robot en [grados]

Al integrar todo, se tiene un sistema de bloques como el mostrado en la Figura 29. Para que funcione correctamente se añade una pequeña función secuencial entre el vector solución y el bloque *Object Detector*, esta función detecta cuando el robot móvil ha llegado a una posición deseada del vector solución, y define al próximo elemento del vector solución como la nueva posición deseada.

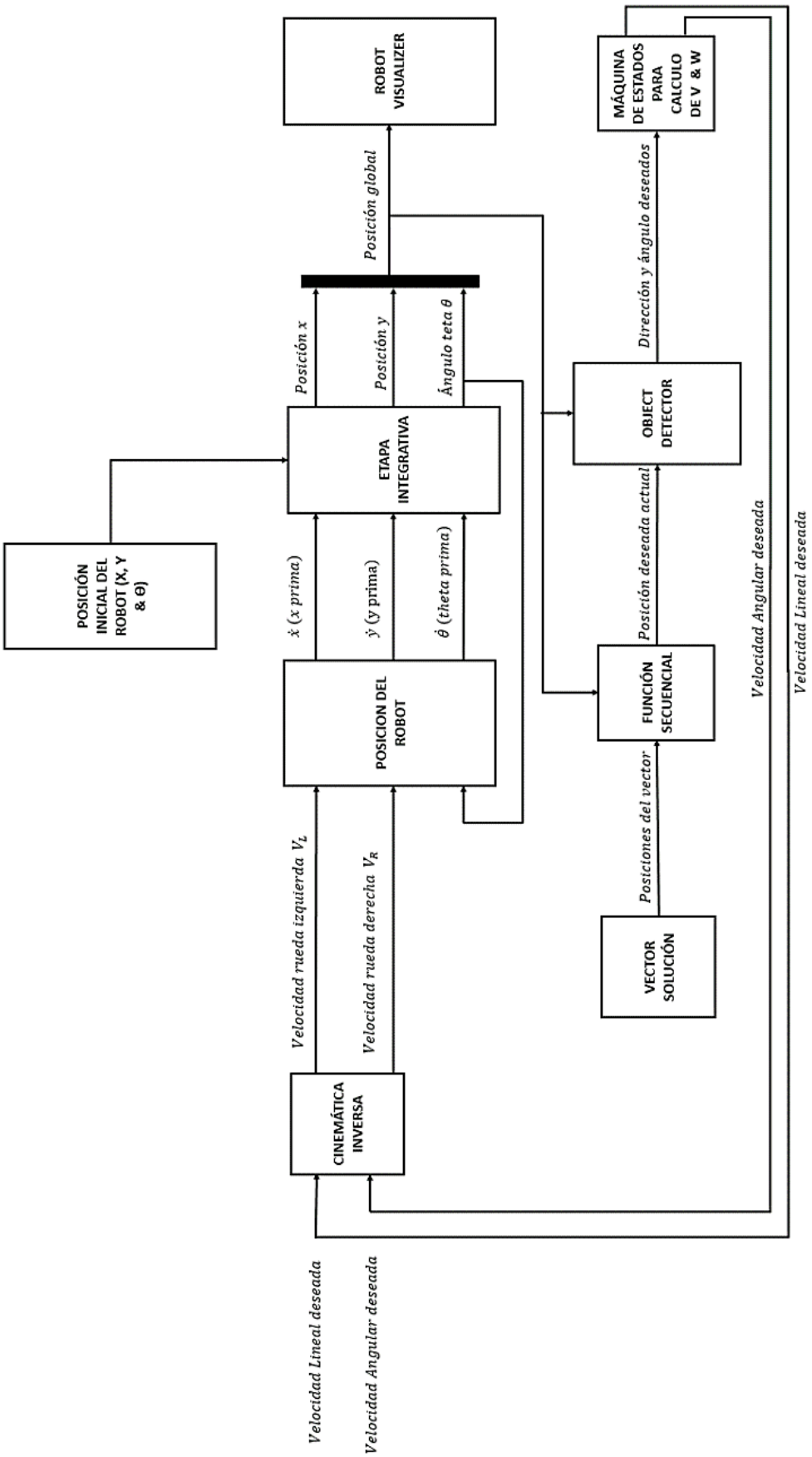



**Figura 29.** Sistema de bloques para Simulación de Algoritmo RRT.

En la Figura 30 se puede observar la simulación final del robot móvil guiado por el algoritmo RRT. Nótese que el camino dejado a su paso por el robot corresponde a la ruta planificada presentada en la Figura 27.

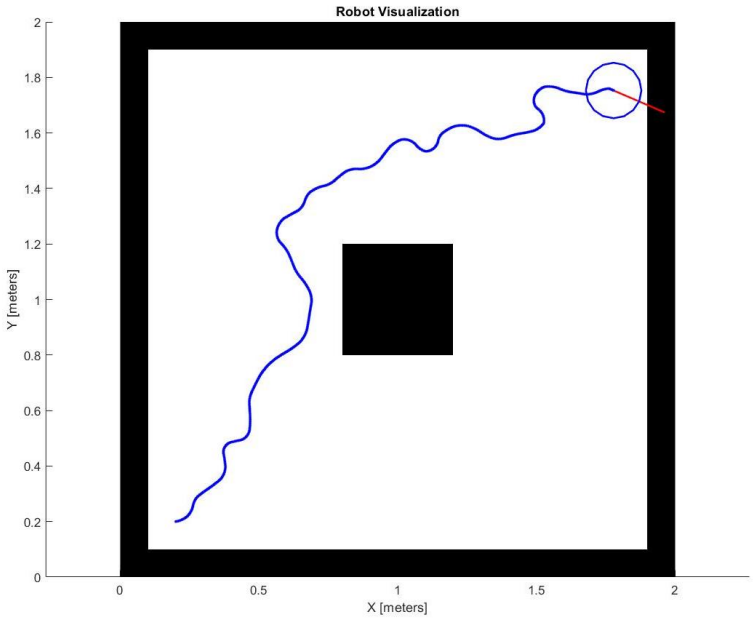



**Figura 30.** Simulación de robot móvil guiado por algoritmo RRT.

## 2.5.2. RRT* (RAPIDLY RANDOM TREE STAR)

El algoritmo RRT*, es una modificación del algoritmo RRT del cual se habló en la subsección anterior. Ya que el algoritmo RRT es su base, se puede evitar ciertas explicaciones y profundizar en los cambios que trae consigo este nuevo algoritmo.

El primer cambio es la adición de un “área vecinal o área de búsqueda” que se forma a cada nodo cuando es creado. El segundo cambio es un registro del costo de cada nodo creado, este costo se refiere a la distancia recorrida desde el nodo inicial hasta el nuevo nodo creado. El último cambio es la posibilidad de conectar un nodo ya existente al nuevo nodo creado de forma que se reduzca el costo del nodo existente. (Adiyatov & Varol, 2017)

### 2.5.2.1. Lógica RRT*

La lógica del algoritmo RRT* comienza con el mismo método de creación de nuevos nodos que se predecesor, en el cual se generaba un punto aleatorio

en el mapa y se creaba una línea en la cual se colocaba el nodo nuevo a una cierta distancia predefinida.

La diferencia comienza en este punto, y esta comienza con la adición de un registro de los costos de cada nodo creado, el cual servirá para crear un camino más eficiente para llegar al objetivo final. En el algoritmo RRT* cuando se crea un nodo este no se conecta directamente al nodo más cercano, sino que hace una búsqueda alrededor de sí mismo dentro de un rango predeterminado hasta encontrar el nodo más eficiente al que unirse y el cual le garantice al nuevo nodo el menor costo posible. Luego de encontrar al nodo más eficiente, se lo designa como nodo padre e inmediatamente se lo conecta al nodo nuevo. En la Figura 31 se puede observar cómo se selecciona al nodo que proporciona una mayor eficiencia como nodo padre a pesar de que haya un nodo más cercano. Nótese que a pesar que el nodo C tiene un menor costo, no se lo selecciona ya que no le proporcionaría la mayor eficiencia al nodo nuevo creado.

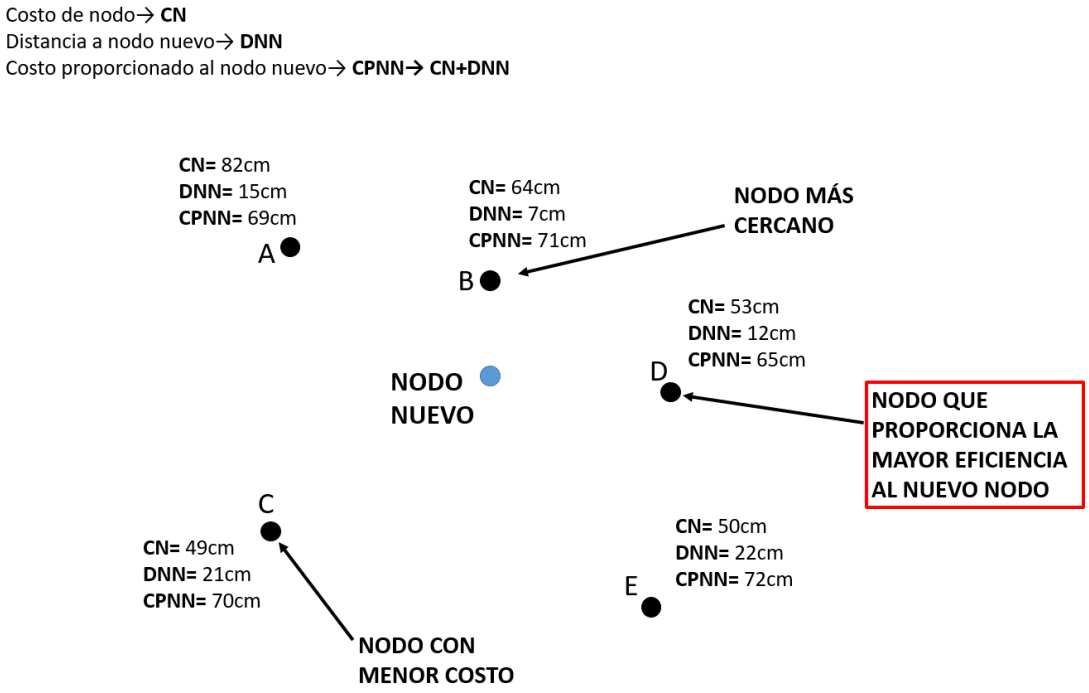



**Figura 31**. Selección de Nodo Padre Algoritmo RRT*

Luego de conectado el nodo nuevo, se pone en marcha un análisis de los nodos vecinos (con excepción del nodo padre) respecto al nuevo nodo. En este análisis se verifica si uno de los nodos vecinos podría disminuir su costo al conectarse como nodo hijo al nuevo nodo creado. Si el costo del nodo se disminuye, este es reconectado directamente al nodo nuevo, el cual pasa a ser su nodo padre. En la Figura 32 se puede ver un ejemplo de esto siguiendo

el mismo grupo de nodos planteados en la Figura 31, en este caso, el nodo A puede reducir su costo total al unirse al nodo nuevo, por lo que es reconectado a este.

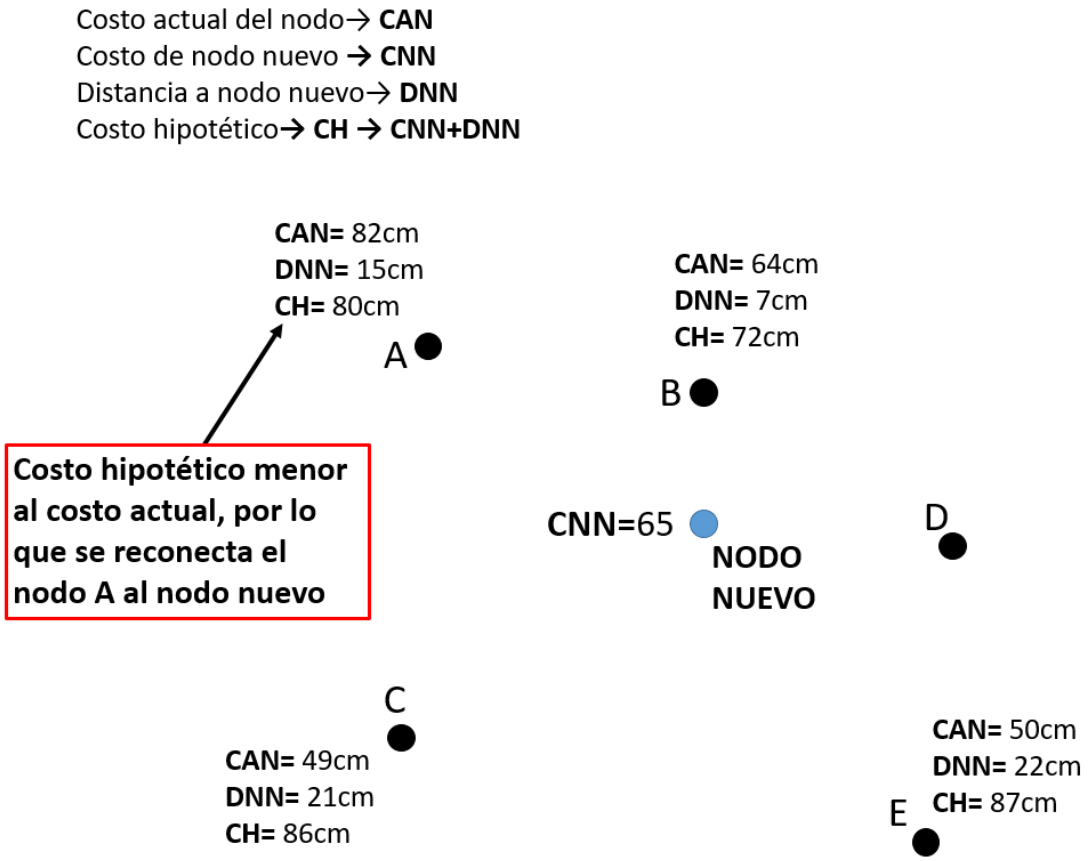



**Figura 32.** Reconexión de nodo existente a nodo nuevo para disminuir su costo.

Estos análisis extra mencionados son los cambios que incluye el algoritmo RRT* con respecto a su predecesor, los cuales tienen el fin de minimizar la longitud de la ruta que debe seguir el robot móvil.

### 2.5.2.2. Implementación RRT*

El algoritmo RRT* se implementa de forma idéntica al RRT, con la inclusión de las modificaciones ya mencionadas. En la Figura 33 se puede observar el diagrama de flujo del algoritmo RRT*, como se puede observar, su dato de salida final es un vector de solución de solución que será usado para la simulación de este algoritmo.

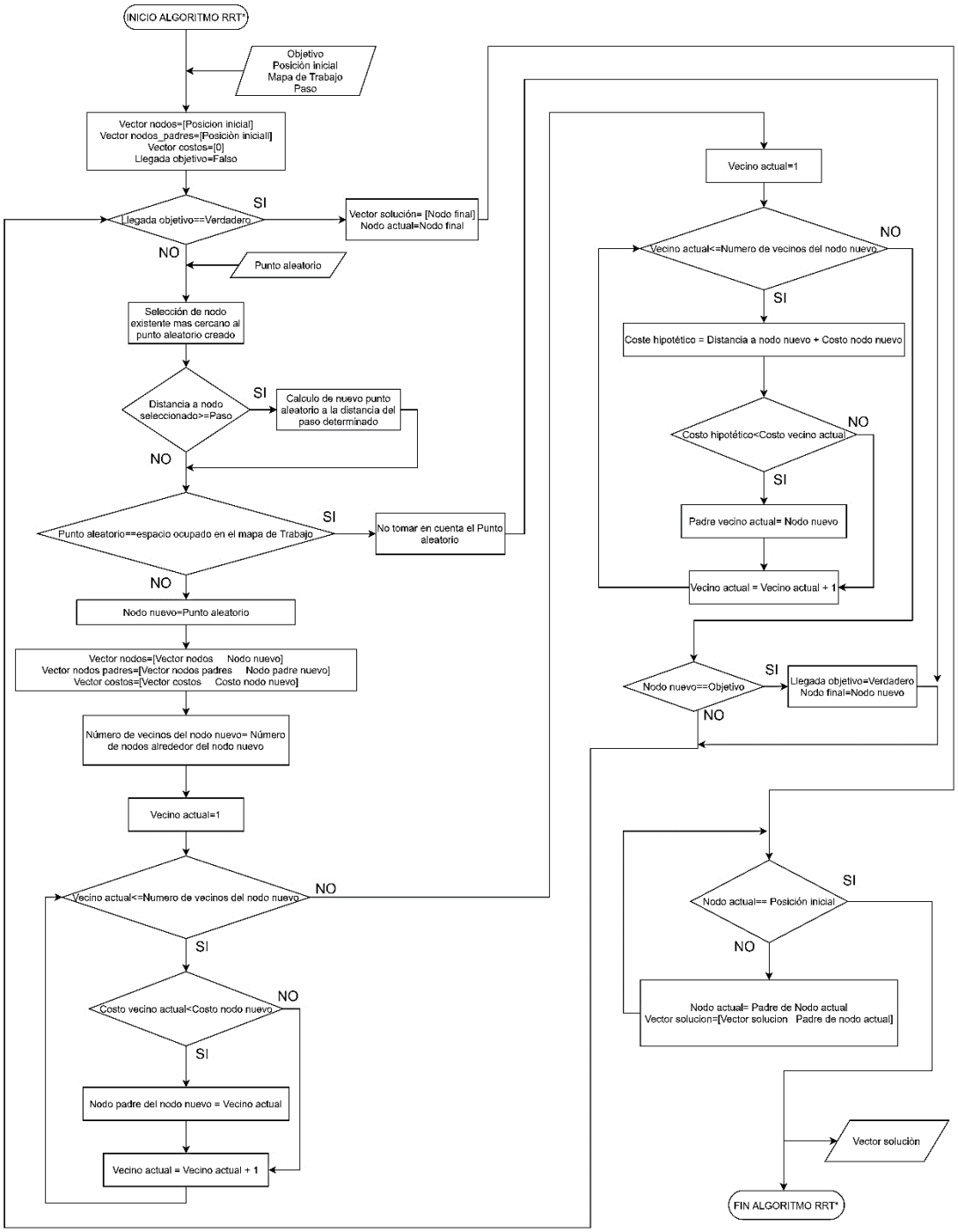



**Figura 33.** Diagrama de Flujo Algoritmo RRT*.

Cabe destacar que, al igual que con el algoritmo RRT, se toma a consideración el radio del robot (en este caso 0.1m) al momento de realizar la exploración del mapa. Por lo tanto, para este algoritmo también se utiliza un mapa de trabajo creado a partir del mapa original y que es en el cual se realiza la exploración.

En la Figura 34 se puede observar el **vector solución** generado por el algoritmo RRT*. A pesar que la generación de puntos se realiza de la misma forma que en el algoritmo RRT, se puede notar a simple vista los cambios que

el algoritmo RRT* conlleva en cuanto a la interconexiones de los puntos creados. El vector solución se nota más corto y estable, contrastando mucho con el vector solución presentado en la Figura 27, el cual presenta una forma más dentada e irregular.

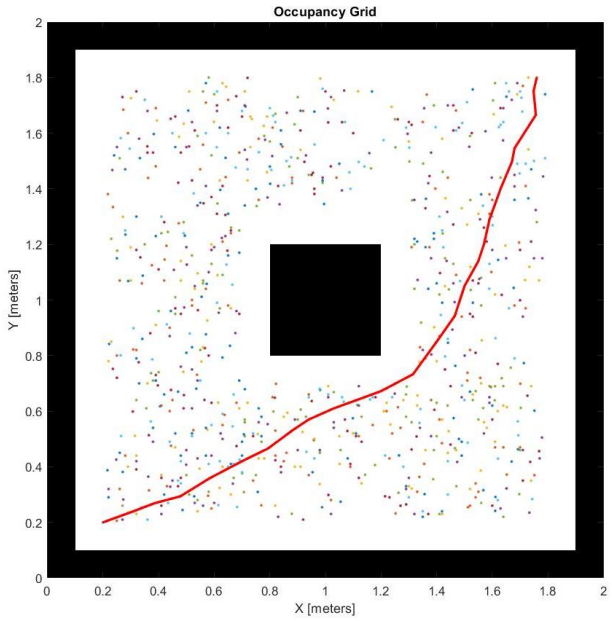



**Figura 34.** Planificación de ruta realizada por algoritmo RRT* en mapa básico de comprobación.

### 2.5.2.3. Simulación RRT*

La simulación del algoritmo RRT* se realiza de la misma que para con el algoritmo RRT, esto ya que ambos utilizan un vector solución como dato para guiar al robot móvil hasta su objetivo.

En la Figura 35 se puede observar la simulación del robot móvil guiado por el algoritmo RRT*, nótese que la ruta seguida por el robot móvil es la misma que la presentada en la Figura 34.

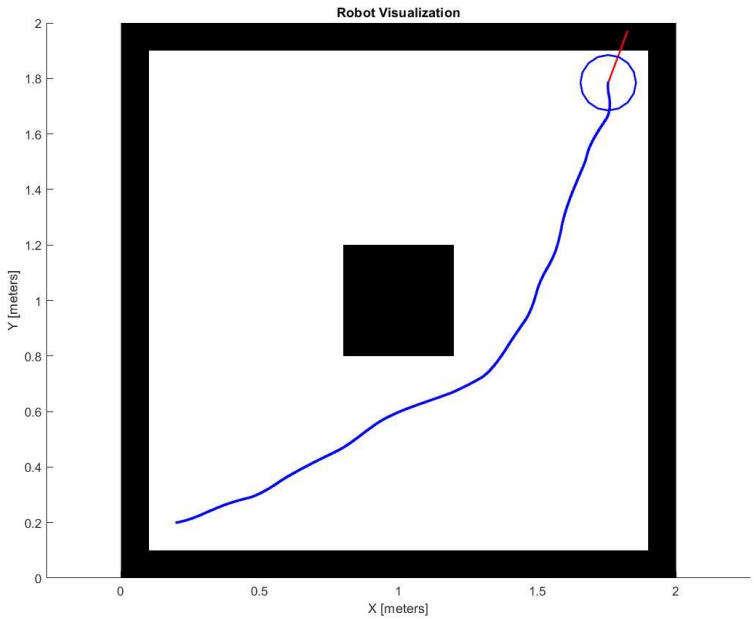



**Figura 35.** Simulación de robot móvil guiado por algoritmo RRT*.

## 2.5.3. POTENTIAL FIELD PATH PLANNING

El Potential Field Path Planning trata de establecer un campo potencial de atracción alrededor del objetivo al que se quiere llegar a la par que se establece un campo potencial repulsivo alrededor de los obstáculos a evitar. De esta forma, se usan fuerzas atractivas y repulsivas para guiar a un robot móvil hacia una meta específica. (Orozco-Rosas et al., 2019)

El algoritmo PFPP utiliza el mapa predeterminado donde se moverá el robot para crear un campo potencial cuyo punto mínimo estará posicionado sobre el objetivo o meta a donde el robot móvil deberá llegar. La magnitud del potencial de un punto X del mapa será directamente proporcional a la distancia entre el punto y el objetivo, e inversamente proporcional a la distancia entre el punto y cualquier obstáculo cercano que pudiera existir. El robot, como si de una partícula en una pendiente se tratase, se desplaza de manera descendente desde un mayor a un menor potencial de manera repetitiva hasta que se llega al punto mínimo del campo, que es donde se encuentra el objetivo. (Bounini et al., 2017)

### 2.5.3.1. Lógica Potential Field Path Planning

La lógica del algoritmo PFPP se trata de un proceso repetitivo. Este proceso repetitivo pasa por cada una de las celdas del mapa (en este caso cada celda

mide 0.1 m de lado) y calcula su potencial dependiendo de su distancia al objetivo y su distancia a obstáculos cercanos. De esta forma, podemos definir el potencial de una celda (x, y) como la suma del potencial atractivo con el potencial repulsivo de la misma celda.

$$U = U_{att} + U_{rep} \quad [10]$$

Donde:

$U$ : Potencial total de una celda (x, y) del mapa
$U_{att}$ : Potencial atractivo de una celda (x, y) del mapa
$U_{rep}$ : Potencial repulsivo de una celda (x, y) del mapa

El cálculo del potencial atractivo es realizado mediante la combinación de dos tipos de potenciales atractivos, uno cónico y otro parabólico. Cabe destacar que, además del potencial, también es calculada la gradiente de este potencial, que sirve para definir la velocidad con la que se mueve el robot móvil a lo largo de su trayecto.

El potencial atractivo cónico $U_{att}$ y su gradiente $\nabla U_{att}$ son calculados mediante las siguientes expresiones:

$$U_{att} = \zeta d(pos_{actual}, pos_{objetivo}) \quad [11]$$

$$\nabla U_{att} = \frac{\zeta}{d(pos_{actual}, pos_{objetivo})} \left(pos_{actual} - pos_{objetivo}\right) \quad [12]$$

Donde (para 11 y 12):

$U_{att}$ : Potencial de un punto X del mapa
$pos_{actual}$: Posición de un punto X del mapa (x, y)
$pos_{objetivo}$: Posición del objetivo final o meta (x, y)
$d(pos_{actual}, pos_{objetivo})$: Distancia entre la posición actual y la posición objetivo en metros [m]
$\nabla U_{att}$: Gradiente negativa del potencial
$\zeta$: Constante de escalamiento (en este caso 1)

Las expresiones anteriores conllevan a la creación de un potencial atractivo cónico y una gradiente similar a lo que se puede ver en la Figura 36. A pesar de que el potencial cumple con proporcionar un potencial mínimo en el objetivo, se puede ver que la gradiente, la cual será la que defina la velocidad del robot móvil, tiene un cambio drástico en su magnitud que provocaría

complicaciones al momento de la aproximación final al objetivo. Es por este motivo que se añade un potencial atractivo parabólico que se activa al traspasar un umbral predeterminado durante la aproximación final al objetivo, este umbral es colocado a una distancia X alrededor del objetivo.

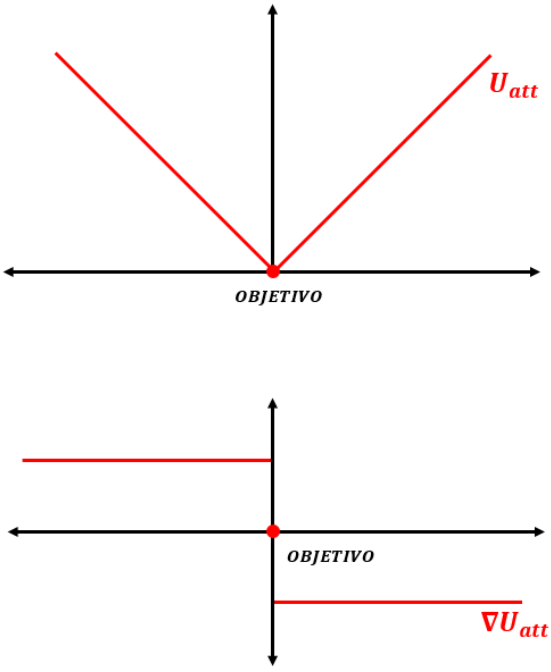



**Figura 36.** Potencial atractivo cónico Uatt y gradiente negativa ∇Uatt del mismo.

El potencial atractivo parabólico y su gradiente son calculados mediante las siguientes expresiones:

$$U_{att} = \frac{1}{2}\zeta d^2\left(pos_{actual}, pos_{objetivo}\right) \quad [13]$$

$$\nabla U_{att} = \nabla\left(\frac{1}{2}\zeta d^2\left(pos_{actual}, pos_{objetivo}\right)\right)$$

$$\nabla U_{att} = \frac{1}{2}\zeta \nabla \mathrm{d}^2\left(pos_{actual}, pos_{objetivo}\right)$$

$$\nabla U_{att} = \zeta\left(pos_{actual} - pos_{objetivo}\right) \quad [14]$$

Las ecuaciones 13 y 14 comparten los mismos datos que las ecuaciones 11 y 12.

En la Figura 37 se puede observar una representación del potencial atractivo parabólico junto con su gradiente, la cual se destaca debido a que presenta

un cambio progresivo en su magnitud, lo que permite que la aproximación final del robot móvil a su objetivo sea suave y controlada.

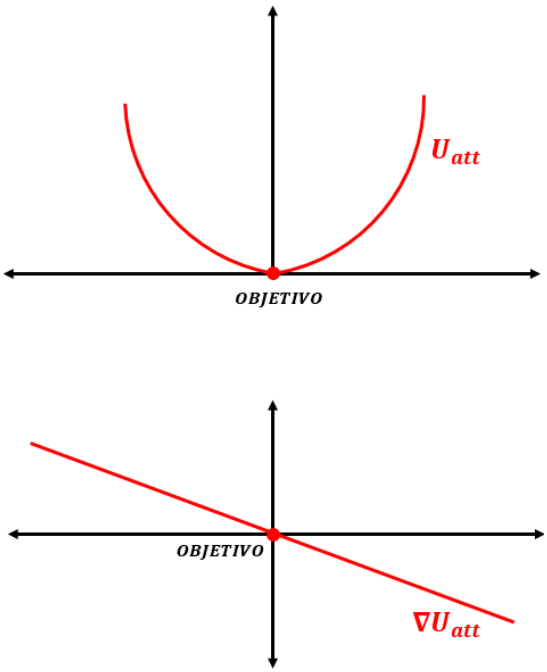



**Figura 37.** Potencial atractivo parabólico Uatt y gradiente negativa ∇Uatt del mismo.

Aunque el potencial atractivo parabólico y su gradiente son ideales para la aproximación final al objetivo, estos no pueden aplicarse más allá ya que tienden a generar magnitudes demasiado grandes cuando una celda se encuentra a gran distancia del objetivo. Por lo tanto, a grandes distancias, la gradiente provocaría que el robot móvil tenga una velocidad muy alta, lo que con certeza provocaría alta inestabilidad, poca maniobrabilidad y una alta probabilidad de chocar contra obstáculos.

Con el potencial cónico y parabólico como recursos, se plantea la solución final para tener un trayecto así como una aproximación final estables, esta solución es la combinación de estos dos tipos de potencial atractivo.

La combinación de ambos potenciales se realiza aplicando el potencial cónico a las celdas que se encuentran más allá de una distancia $D_{objetivo}$ con respecto al objetivo final, y aplicando el potencial parabólico para las celdas que se encuentren a igual o menor distancia que la distancia $D_{objetivo}$.

$$U_{att} = \begin{cases} \zeta d(pos_{actual}, pos_{objet}), & d(pos_{actual}, pos_{objet}) > D_{objet} \\ \frac{1}{2}\zeta d^2(pos_{actual}, pos_{objet}), & d(pos_{actual}, pos_{objet}) \leq D_{objet} \end{cases}$$

$$\nabla U_{att} = \begin{cases} \frac{\zeta}{d(pos_{actual}, pos_{objet})}(pos_{actual} - pos_{objet}) & d(pos_{actual}, pos_{objet}) > D_{objet} \\ \zeta(pos_{actual} - pos_{objet}) & d(pos_{actual}, pos_{objet}) \leq D_{objet} \end{cases}$$

Con estas expresiones, se obtiene un potencial atractivo y gradiente como los que se pueden observar en la Figura 38, nótese que con la combinación de ambos potenciales se logra una navegación óptima y una aproximación suave y precisa.

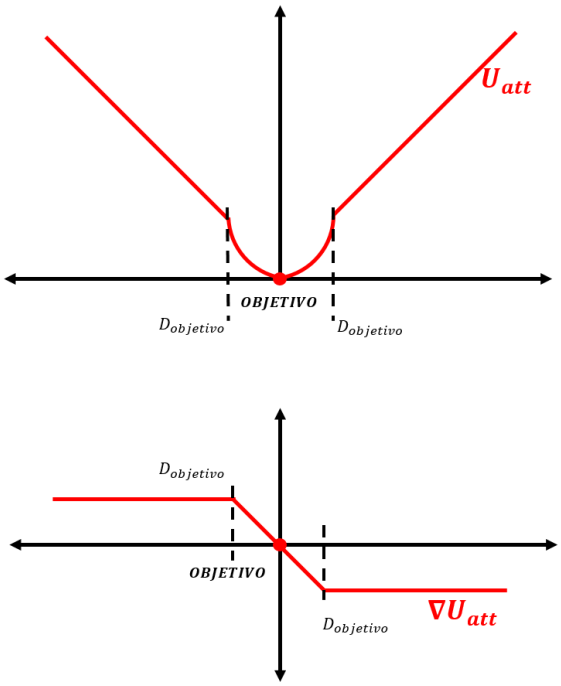



**Figura 38.** Potencial atractivo combinado (cónico+parabólico) Uatt y gradiente negativa ∇Uatt del mismo.

Habiendo definido el potencial atractivo generado por el objetivo final, el siguiente paso es definir el potencial repulsivo generado por los obstáculos.

Para calcular el potencial repulsivo de una celda se emplea una lógica similar a la que se utiliza en el potencial atractivo ya que aquí también se define una distancia $D_{obstáculo}$ que servirá de umbral para decidir si una celda tendrá un potencial repulsivo o no. Si una celda se encuentra a una distancia con

respecto al obstáculo mayor a la distancia $D_{obstáculo}$, está no tendrá un potencial repulsivo. Mientras que, si la celda se encuentra a un distancia con respecto al obstáculo menor o igual que la distancia $D_{obstáculo}$. Esta tendrá un potencial repulsivo que será inversamente proporcional a la magnitud de la distancia de la celda con respecto al obstáculo. Cabe destacar que, en el caso en que una celda esté cerca de dos obstáculo o más, su potencial repulsivo se calculará basado en la distancia al obstáculo más cercano.

$$Considerando: d(pos_{actual}, pos_{obstáculo}) = \boldsymbol{d(Q)}$$

$$U_{rep} = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d(Q)} - \frac{1}{D_{obstáculo}}\right)^2, & d(\mathrm{Q}) \leq D_{obstáculo} \\ 0, & d(\mathrm{Q}) > D_{obstáculo} \end{cases} \quad [15]$$

$$\nabla U_{rep} = \begin{cases} \eta\left(\frac{1}{D_{obstáculo}} - \frac{1}{d(Q)}\right)\frac{1}{d^2(Q)}\nabla d(Q), & d(Q) \leq D_{obstáculo} \\ 0, & d(Q) > D_{obstáculo} \end{cases} \quad [16]$$

Donde (para 15 y 16):

$U_{rep}$ : Potencial repulsivo de una celda (x, y) del mapa.
$\nabla U_{rep}$: Gradiente del potencial repulsivo de una celda (x, y) del mapa.
$\eta$: Constante de escalamiento.
$d(Q)$: Distancia de una celda (x, y) hasta el obstáculo más cercano [m].
$D_{obstáculo}$: Distancia de umbral predeterminada [m].

Con estas expresiones, se obtiene un potencial repulsivo y gradiente similares a lo que muestra en la Figura 39. Se puede notar a simple vista las asíntotas que se forman en los obstáculos, lo que repelerá al robot inmediatamente ya que este siempre escogerá seguir el menor potencial.

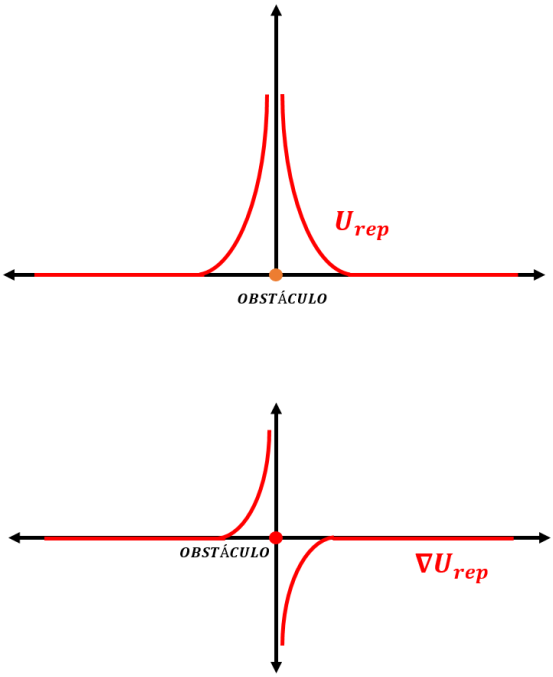



**Figura 39.** Potencial repulsivo Urep y gradiente negativa ∇Urep del mismo.

La suma del potencial atractivo con el potencial repulsivo da como resultado el potencial total, el cual finalmente guiará al robot a lo largo del mapa hasta su meta.

### 2.5.3.2. Implemetación Potential Field Path Planning

La lógica del algoritmo PFPP se puede ver implementada de mejor forma en la Figura 40. El proceso, en general, se trata de pasar por cada una de las celdas del mapa y asignarles su potencial calculado usando las ecuaciones previamente mostradas.

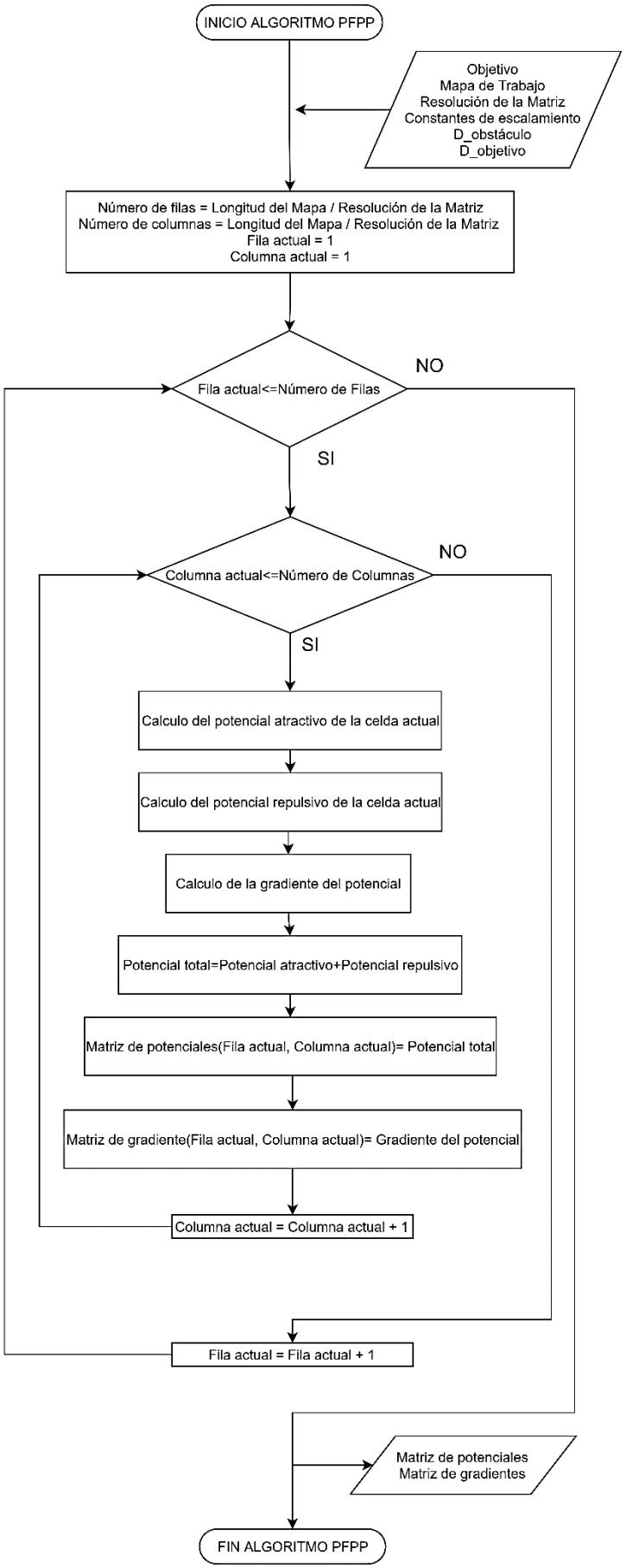



**Figura 40.** Diagrama de flujo algoritmo PFPP.

El resultado final de la aplicación del algoritmo PFPP se puede ver en las Figuras 41 y 42, donde se muestra el campo potencial y gradiente generados tomando como base el mapa de comprobación mostrado en la Figura 19 y tomando como posición objetivo al punto (1.8, 1.8). De estos campos generados se debe destacar dos puntos.

El primero de estos puntos es que, debido a que la gradiente repulsiva puede generar cierta inestabilidad dentro de la navegación del robot, esta no es considerada para la implementación final. El segundo punto es que en este algoritmo, al igual que en el RRT y RRT*, es necesario tomar en cuenta el radio de robot móvil (en este caso 0.1m), por lo que el cálculo de los potenciales es de hecho realizado sobre el mapa de trabajo mostrado en la Figura 26.

El campo generado en la Figura 41 se presenta estable y correctamente escalado gracias a la constante de escalamiento correcta y distancia de umbral adecuadas.

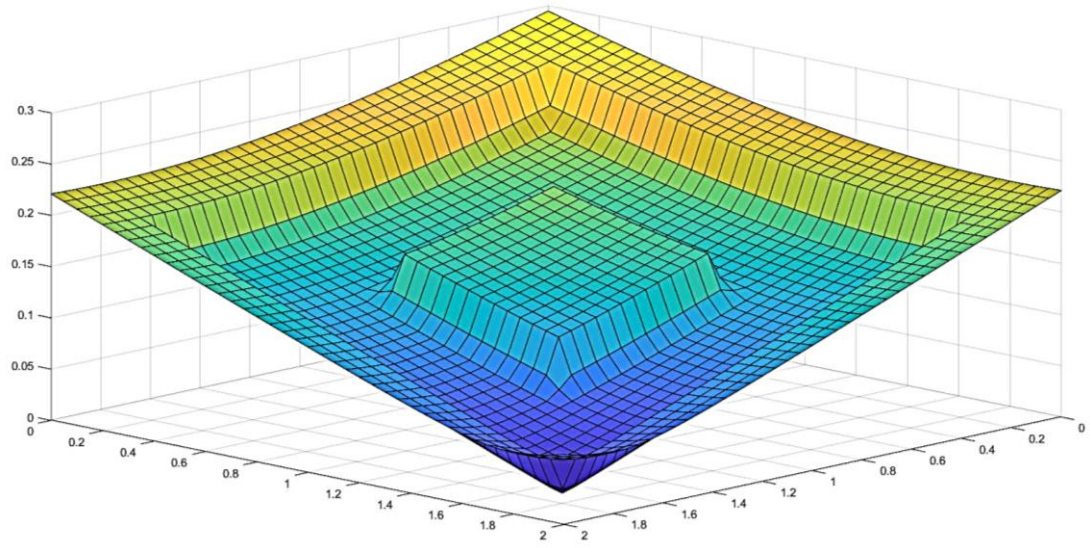



**Figura 41.** Campo potencial generado en base a Mapa de Comprobación.

La gradiente de velocidad que se puede observar en la Figura 42 presenta una velocidad estable en casi la totalidad del mapa, y esta solo disminuye de forma proporcional a la distancia con el objetivo final.

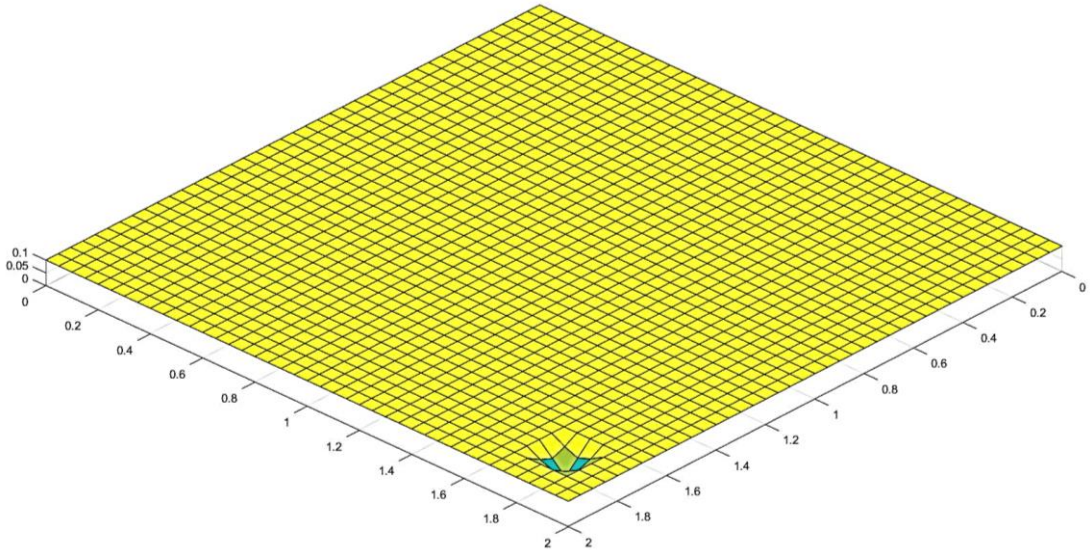



**Figura 42.** Gradiente generada en base a Mapa de comprobación.

### 2.5.3.3. Simulación Potential Field Path Planning

Para la simulación del algoritmo PFPP, se aplica bloques muy similares a los utilizados para la simulación de los algoritmos RRT y RRT*. En este caso, se vuelve a utilizar el bloque ***Object Detector*** presentado en la Figura 28 y ***Robot Visualizer*** presentado en la Figura 21. El segundo de ellos se aplica para la visualización del robot.

Para que el robot móvil siga una ruta desde un potencial mayor hasta un potencial menor, se aplica una función de MatLab la cual tiene como entradas la matriz de potenciales, la matriz de gradientes, la posición actual del robot, la resolución de la matriz y el objetivo final.

Con estos datos, la función de MatLab recibe la posición actual del robot y lo localiza dentro de la matriz usando la resolución del matriz, a continuación busca de entre los vecinos de la posición actual al que tenga el menor potencial y lo pone como el objetivo actual al que se deberá dirigir el robot. Este proceso se repite una y otra vez hasta llegar al punto mínimo del campo potencial que es donde se posiciona el objetivo final. La velocidad del robot móvil se define dependiendo de su posición y el valor de esta posición dentro de la matriz de gradiente. Adicional, se agrega una condición dentro de la función de MatLab la cual indica que la velocidad del robot móvil sea 0 cuando este se encuentre a menos de 0.01 m de su objetivo, esto solo se aplica dentro de esta simulación para evitar que el retardo por los tiempos de carga haga que el robot sobrepase el objetivo, tarde en reconocer que ha llegado, y por el retraso continúe avanzando. En la Figura 43 se puede observar el sistema de bloques que se aplican para la simulación del algoritmo PFPP.

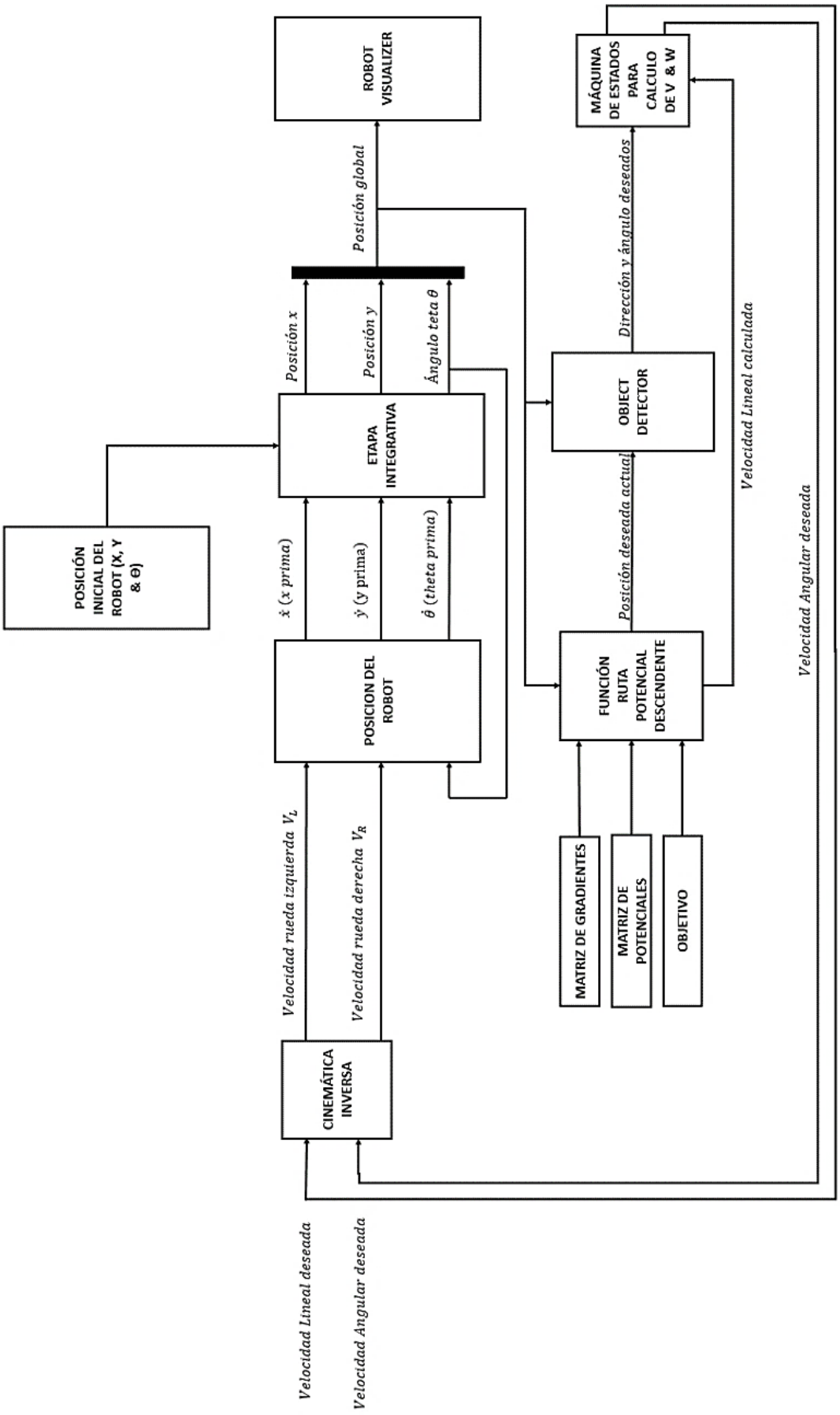



**Figura 43.** Sistema de bloques para Simulación de Algoritmo PFPP.

Como se puede observar, también se utiliza la misma máquina de estados para el cálculo de la velocidad angular W del robot móvil y para dar un escalamiento final a la velocidad provista por la matriz de gradientes, este escalamiento solo se realiza como una herramienta para aumentar o disminuir la velocidad del robot móvil con el objetivo de aumentar la eficiencia del robot móvil.

El resultado final de la simulación del algoritmo en Simulink se puede observar en la Figura 44, al igual que con el algoritmo RRT y RRT*, el robot se mueve desde su posición inicial en (0.2, 0.2) hasta su objetivo final en (1.8, 1.8).

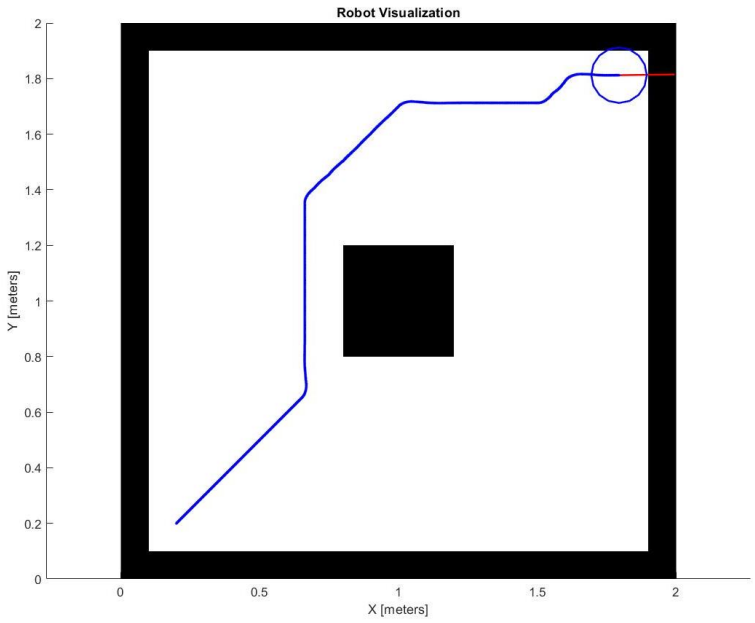



**Figura 44.** Simulación de robot móvil guiado por algoritmo PFPP.

## 2.5.4. VFF (VIRTUAL FORCE FIELD)

El Virtual Force Field es un algoritmo que usa fuerzas repulsivas ejercidas por obstáculos junto con una fuerza atractiva individual ejercida por el objetivo final para guiar a un robot móvil hacia su meta a la vez que se esquiva cualquiera obstáculo que pudiera presentarse en el camino. (Nia et al., 2011)

El VFF es de planificación local, ya que no cuenta con ningún tipo de información previa sobre el mapa donde se desplazará el robot móvil y solo se basa en la información recogida por sensores de distancia colocados alrededor del mismo, los cuales verifican los obstáculos en las inmediaciones del robot. La confiabilidad de las lecturas de los sensores, junto con un correcto posicionamiento global del robot móvil en el mapa, son las claves

para el cálculo de la fuerza resultante que guía al robot hasta su objetivo final. (Nia et al., 2011)

#### 2.5.4.1. Lógica VFF

La lógica del algoritmo VFF se trata del cálculo continuo y repetitivo de una fuerza resultante que indica la dirección en la que se debe mover el robot móvil. Esta fuerza se compone de dos partes, una atractiva y una repulsiva.

$$\overline{F_t} = \overline{F_a} + \overline{F_r} \qquad [17]$$

Donde:

$F_t$ : Fuerza resultante o Fuerza total ejercida sobre el robot móvil.
$F_r$ : Fuerza repulsiva total ejercida sobre el robot móvil.
$F_a$ : Fuerza atractiva ejercida sobre el robot móvil.

La fuerza repulsiva generada es de hecho una suma de varias fuerzas repulsivas menores cuya cantidad es igual al número de sensores colocados alrededor del robot móvil. La magnitud de cada una de las fuerzas menores es calculada mediante la siguiente expresión y es inversamente proporcional a la distancia entre el robot móvil y el obstáculo detectado por el sensor. Nótese el dato $C_{ij,}$ este dato es definido como la confiabilidad del sensor y puede tener un valor entre 0 y 1, siendo 0 una confiabilidad nula en la lectura del sensor y 1 una confiabilidad completa en que la lectura del sensor es correcta.

$$\overline{F_{rep}} = \frac{F_{cr} \cdot C_{ij}}{d^2(robot,obs)} \left( \frac{X_{obs} - X_{robot}}{d(robot,obs)} \hat{x} + \frac{Y_{obs} - Y_{robot}}{d(robot,obs)} \hat{y} \right) \qquad [18]$$

Donde:

$F_{rep}$: Fuerza repulsiva.
$F_{cr}$ : Constante de atracción.
$C_{ij}$ : Confiabilidad del sensor (1 en este caso ya que es una simulación).
$d(robot, obs)$: Distancia entre el robot móvil y el obstáculo detectado por el sensor.
$X_{obs}$ : Posición X del obstáculo detectado.
$X_{robot}$ : Posición X del robot.
$Y_{obs}$ : Posición Y del obstáculo detectado.
$Y_{robot}$ : Posición Y del robot.

Como se puede notar, las fuerzas tienen una componente en X y una componente en Y que luego se sumarán a las componentes de la fuerza atractiva, para al final calcular el ángulo o dirección en la que se moverá el robot móvil.

Las fuerzas repulsivas, como se puede ver en la Figura 45, son contrarías u opuestas a la dirección del obstáculo, por lo que siempre se deben de cambiar el signo de estas al momento de realizar el cálculo final de la fuerza repulsiva total, tal y como se ve en la siguiente expresión.

$$\overline{Fr} = -\sum_0^n \overline{F_{rep1}} + \overline{F_{rep2}} + \overline{F_{rep3}} + \cdots + \overline{F_{repn}} \quad [19]$$

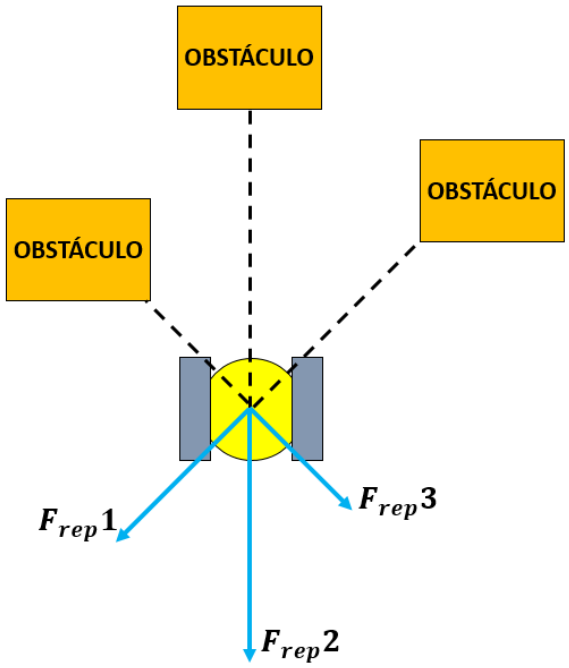



**Figura 45.** Fuerzas repulsivas producidas por obstáculos sobre el robot móvil

La fuerza atractiva individual que atrae al robot móvil hacia el objetivo es calculada mediante la siguiente expresión.

$$\overline{F_a} = F_{ca}\left(\frac{X_{obj}-X_{robot}}{d(robot,obj)}\hat{x}\,\frac{Y_{obj}-Y_{robot}}{d(robot,obj)}\hat{y}\right) \quad [20]$$

Donde:

$F_a$: Fuerza repulsiva.
$F_{ca}$ : Constante de atracción.

$d(robot, obj)$: Distancia entre el robot y el objetivo final.
$X_{obs}$ : Posición X del objetivo.
$X_{robot}$ : Posición X del robot.
$Y_{obs}$ : Posición Y del objetivo.
$Y_{robot}$ : Posición Y del robot.

Debido a que esta es una fuerza atractiva, la dirección de esta es igual a la dirección en la que se encuentra el objetivo, tal y como se ve en la Figura 46. Por lo tanto, no se cambia el signo de esta fuerza al momento de calcular la Fuerza resultante.

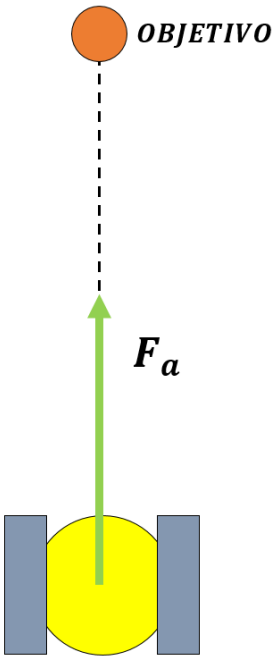



**Figura 46.** Fuerza atractiva producidas por obstáculos sobre el robot móvil.

La suma de la fuerza atractiva con la fuerza repulsiva total da como resultado la fuerza resultante, en la Figura 47 se puede observar una representación de esta fuerza resultante indicando la dirección en la que debe dirigirse el robot móvil.

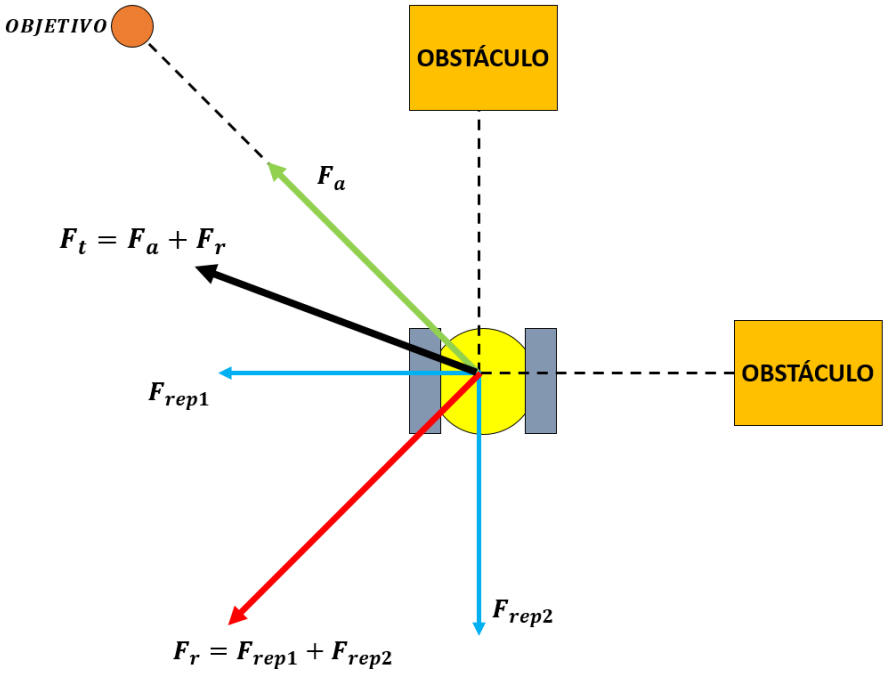



**Figura 47.** Fuerza resultante ejercida sobre el robot móvil.

### 2.5.4.2. Implementación VFF

Para implementar el algoritmo VFF es necesario tomar en consideración que, a diferencia de los anteriores algoritmos, este tiene una planificación local en tiempo real, por lo que su lógica debe ser implementada dentro de la misma simulación.

Para implementar sensores de distancia en el modelo de robot móvil diferencial, se utiliza otro bloque de la librería ***Mobile Robotics Simulation Toolbox***, el cual se llama ***LidarSensor***. Este bloque tiene dos características especiales. La primera de ellas es su capacidad de acoplarse fácilmente al robot móvil simplemente dando como entrada la posición actual del robot móvil y la segunda es el poder declarar e implementar cuantos sensores LIDAR se necesiten. Para este caso, se declararon 9 sensores LIDAR con un espaciado de 22.5° entre ellos, como se puede ver en la Tabla 1.

**Tabla 1.** Posiciones de sensores LIDAR para implementación de algoritmo VFF.

| Sensor | Posición Angular |
|---|---|
| **1** | -90°→270° |
| **2** | -67.5°→292.5° |
| **3** | -45°→315° |
| **4** | -22.5°→337.5° |
| **5** | 0 |
| **6** | 22.5° |
| **7** | 45° |
| **8** | 67.5° |
| **9** | 90° |

En la Figura 48 se puede observar el bloque de Simulink llamado LidarSensor. El bloque requiere para su funcionamiento la declaración del mapa en el cuál se moverá el robot, la posición de los sensores de distancia y el rango máximo de funcionamiento de estos sensores.

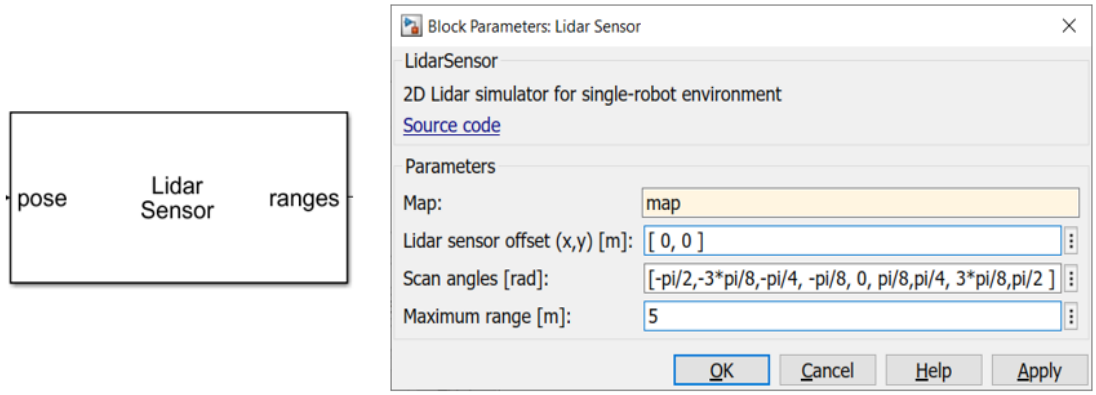



**Figura 48.** Bloque LidarSensor

Los sensores LIDAR declarados indican las distancias a las que se encuentran obstáculos o paredes. Con los datos de las distancias, ángulos de los sensores y posición actual del robot móvil, se calcula la posición de los obstáculos para así obtener todos los datos necesarios y calcular las fuerzas repulsivas generadas.

La Fuerza atractiva generada se calcula tomando en cuenta la posición del objetivo y la posición actual del robot. Como se puede ver en la Figura 48, las distancias calculadas de los sensores LIDAR vienen como un solo dato de salida llamado *Ranges*, para descomponer este dato se emplea una máquina de estados la cual tiene un funcionamiento en paralelo y separa cada dato de distancia que será usado posteriormente. Teniendo el resultado de las fuerzas atractiva y repulsiva, se procede a hacer el cálculo de la fuerza resultante y su ángulo, el cual indica la dirección en la que se debe mover el robot móvil. La

velocidad de desplazamiento tiene un control condicional y proporcional a la distancia en el robot móvil y su objetivo.

En la Figura 49 se puede observar la conexión entre los bloques mencionados. Nótese la máquina de estados conectada al final de estos, la cual cumple con la función de comparar la posición angular actual del robot con la posición angular deseada calculada y, en caso de que estas no sean iguales, corregir la posición angular del robot mediante un cambio en la velocidad angular W.

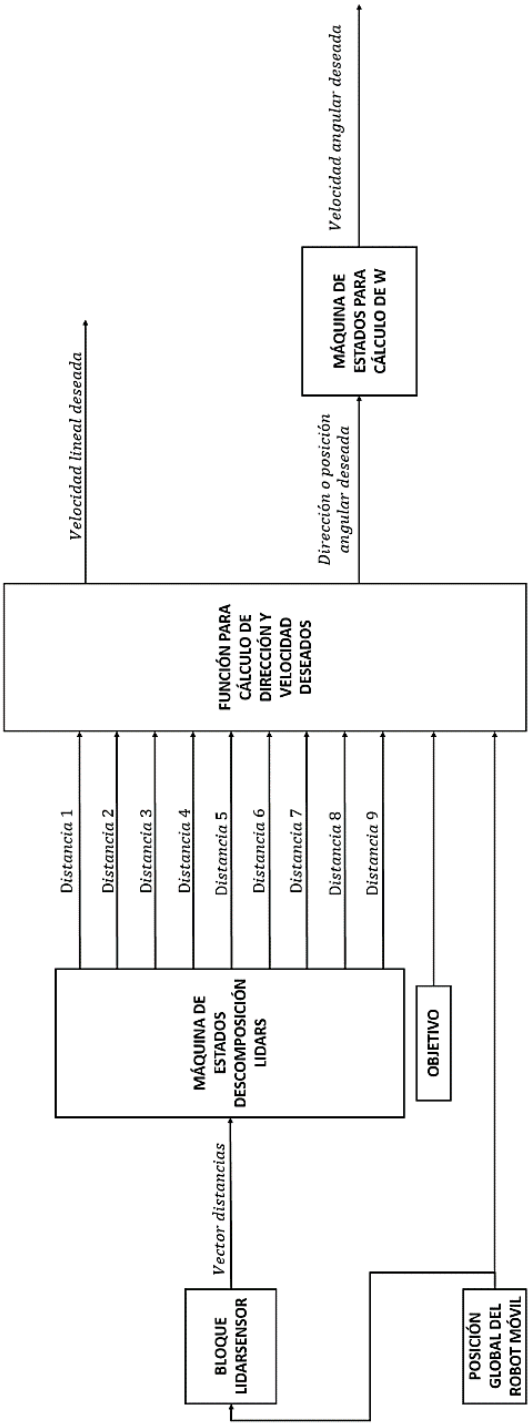



**Figura 49.** Sistema de bloques para implementación de algoritmo VFF.

Dado este sistema de bloques, la lógica del algoritmo VFF que es llevada a cabo se puede ver representada en la Figura 50.

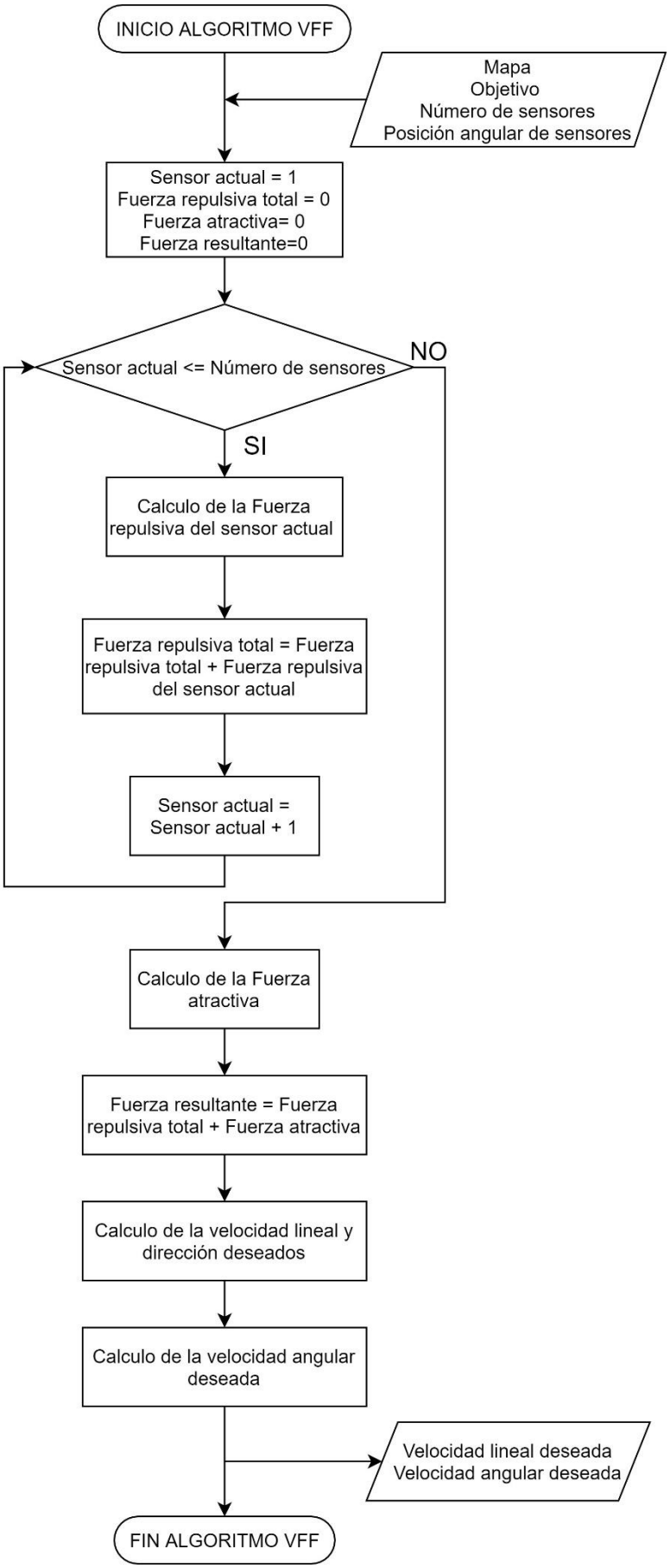



**Figura 50.** Diagrama de flujo algoritmo VFF.

### 2.5.4.3. Simulación VFF

Para la simulación del algoritmo VFF, simplemente se añade los bloques del sistema de navegación del robot móvil a los bloques de implementación del algoritmo. El sistema de bloques completo para la simulación del algoritmo VFF se puede ver en la Figura 51.

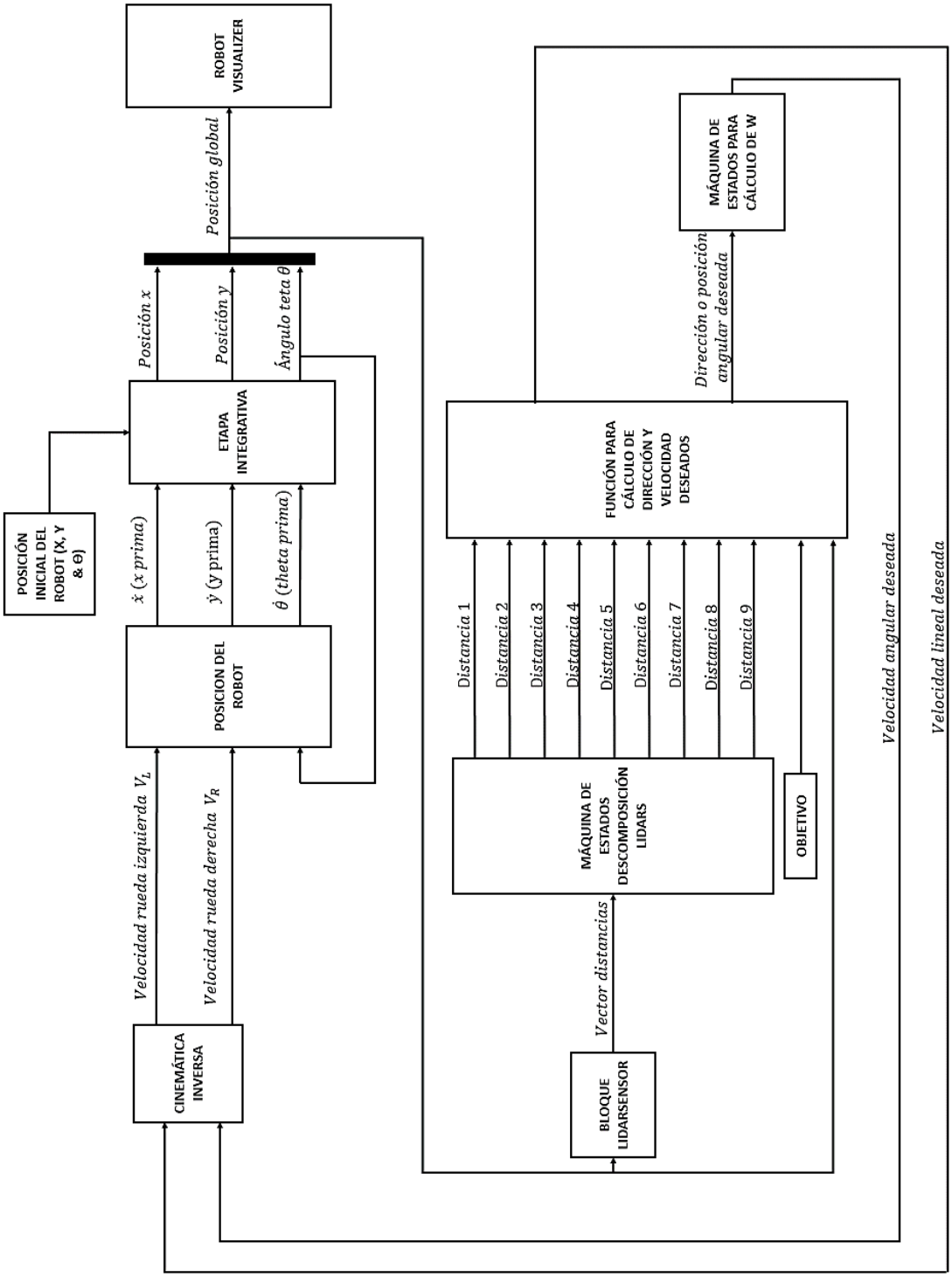



**Figura 51.** Sistema de bloques para Simulación de Algoritmo VFF.

El resultado final de este sistema de bloques se pude observar en la Figura 52. Como en los pasados algoritmos, el robot móvil es ordenado a moverse desde la posición (0.2, 0.2) a la posición (1.8, 1.8) dentro del mapa de comprobación mostrado en la Figura 19.

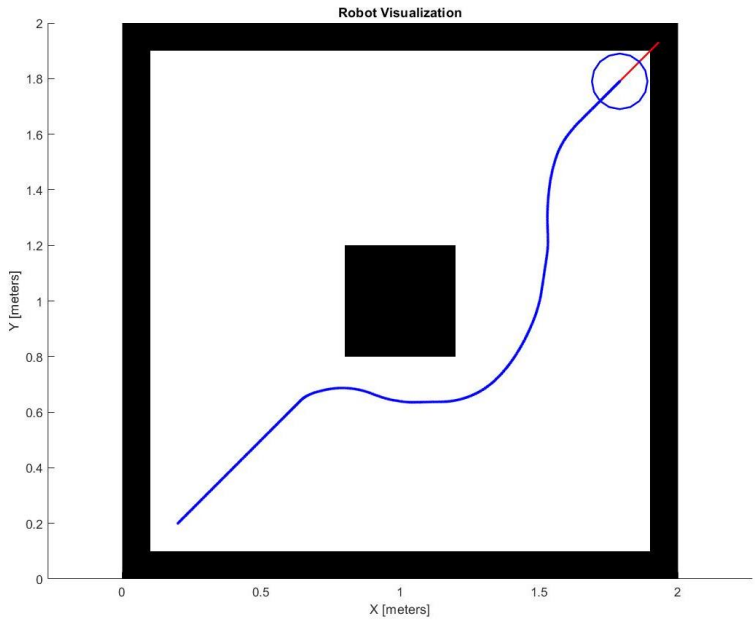



**Figura 52.** Simulación de robot móvil guiado por algoritmo VFF.

## 2.5.5. VFH (VECTOR FIELD HISTOGRAM)

El Vector Field Histogram es un algoritmo de planificación local, y de hecho puede considerarsele una evolución del algoritmo VFF antes presentado. En el VFH, se aplica una cantidad mayor de sensores con el objetivo de cubrir todos los alrededores del robot móvil y, debido al gran número de sensores, los datos recibidos por estos deben pasar por un proceso de reducción y acondicionamiento antes de poder usarlos para calcular la dirección por la que se debe mover el robot móvil. (Alagic et al., 2019)

El algoritmo VFH reduce la información del ambiente del robot en un procedimiento dividido en dos pasos. El primero paso es la transformación del mapa cartesiano donde desplaza el robot móvil en un histograma polar 1D (una dimensión), el segundo paso es la discretización de este histograma para proceder a dividir el ambiente del robot en sectores que pueden ser considerados como “ocupados” o libres”. Tras la determinación del estado de todos los sectores, se realiza una evaluación para seleccionar la dirección

óptima, dentro de uno de los sectores libres, en la que se dirigirá el robot móvil. (Li et al., 2016)

#### 2.5.5.1. Lógica VFH

La lógica del algoritmo VFH comienza con la declaración de una cantidad determinada de sensores de distancia alrededor del robot móvil, sea cual sea la cantidad que se escoja, estos deberán cubrir todos los alrededores del robot de forma uniforme. En la Figura 53 se puede observar una representación del robot móvil con un total de 12 sensores de distancia, estos 12 sensores dividen los alrededores del robot 12 sectores, cada sector con un rango radial de 30°.

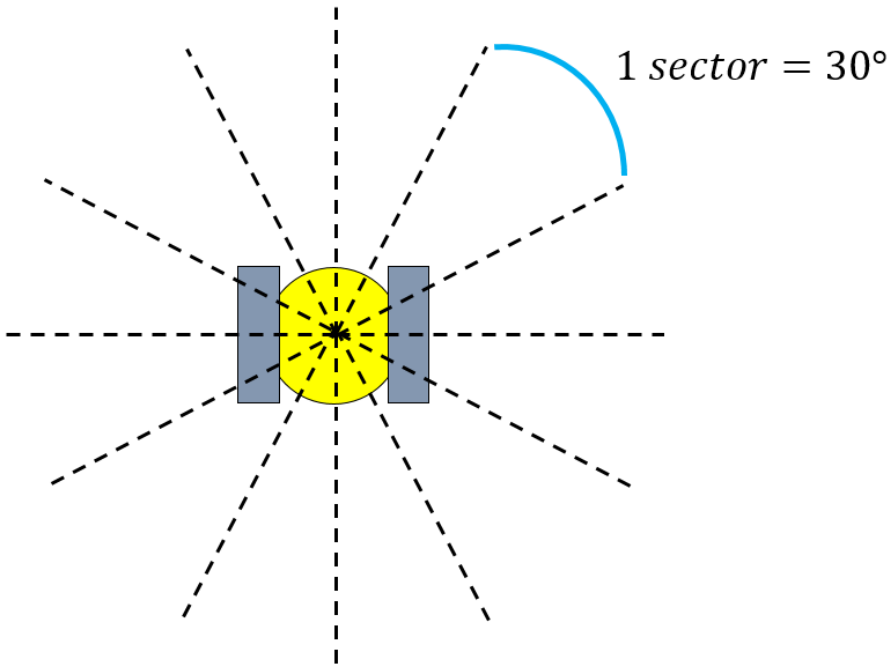



**Figura 53.** Distribución de sensores de distancia para algoritmo VFH.

Tras de declararse los sensores, se calcula la magnitud de ocupación de cada uno de estos mediante la siguiente expresión. De esta forma, se consigue el valor de ocupación de cada sector alrededor del robot móvil.

$$h_k = M_n = \left(C_{ij}\right)^2 \left(a - b.d_{ij}\right) \quad [21]$$

Donde:

$M_n,\ h_k$ : Magnitud del sector n.
$C_{ij}$ : Confiabilidad de la lectura del sensor (en este caso 1).
$a, b$ : Constantes positivas.
$d_{ij}$ : Distancia entre el robot y el obstáculo detectado por el sensor n.

Teniendo ya la magnitud de cada sector, es necesario tomar en cuenta que podría suceder el tipo de situación como la que se puede ver en la Figura 54, en la que uno de los sectores detecte un obstáculo cercano pero uno de sus sectores vecinos colindantes no lo detecte, y por consiguiente ocurra una colisión ya que el robot móvil detecta dicho sector como libre.

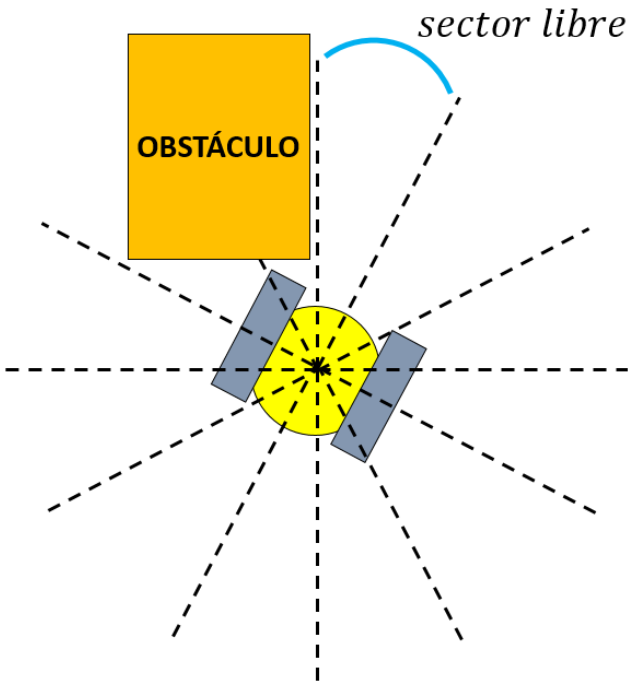



**Figura 54.** Situación de colisión por sector libre con interferencia.

Por el tipo de situación mostrada anteriormente, es que se establece la siguiente condición, en la que se recalcula la magnitud de ocupación de cada sector tomando en cuenta las magnitudes de ocupación de sus vecinos. Este cálculo se realiza mediante la siguiente expresión y garantiza evitar cualquier tipo de colisión accidental debido a una lectura de sector libre con interferencia.

$$h'_k = \frac{h_{k-l}+2h_{k-l+1}+\cdots+lh_k+\cdots+2h_{k+l-1}+h_{k+l}}{2l+1} \quad [22]$$

Donde:

$h'_k$: Magnitud recalculada del sector n.
$l$: Número de vecinos a tomar en cuenta para el recálculo de la magnitud.

Con los valores recalculados, se tiene las magnitudes definitivas de cada sector, las cuales pueden ser representadas en un histograma polar 1D. En la Figura 55 se puede observar la situación de un robot móvil frente a un par de obstáculos y a continuación dos histogramas polares 1D, el primero de ellos compuesto por las magnitudes de ocupación originales y el segundo con las magnitudes recalculadas. Nótese que, en el segundo histograma (2), un sector con magnitud de ocupación cero en el primer histograma (ángulo 60°), y que se encuentra peligrosamente cerca de un obstáculo, (1) es ahora considerado un sector con magnitud de ocupación alta debido a la influencia de los sectores colindantes al mismo al momento de la recalculación.

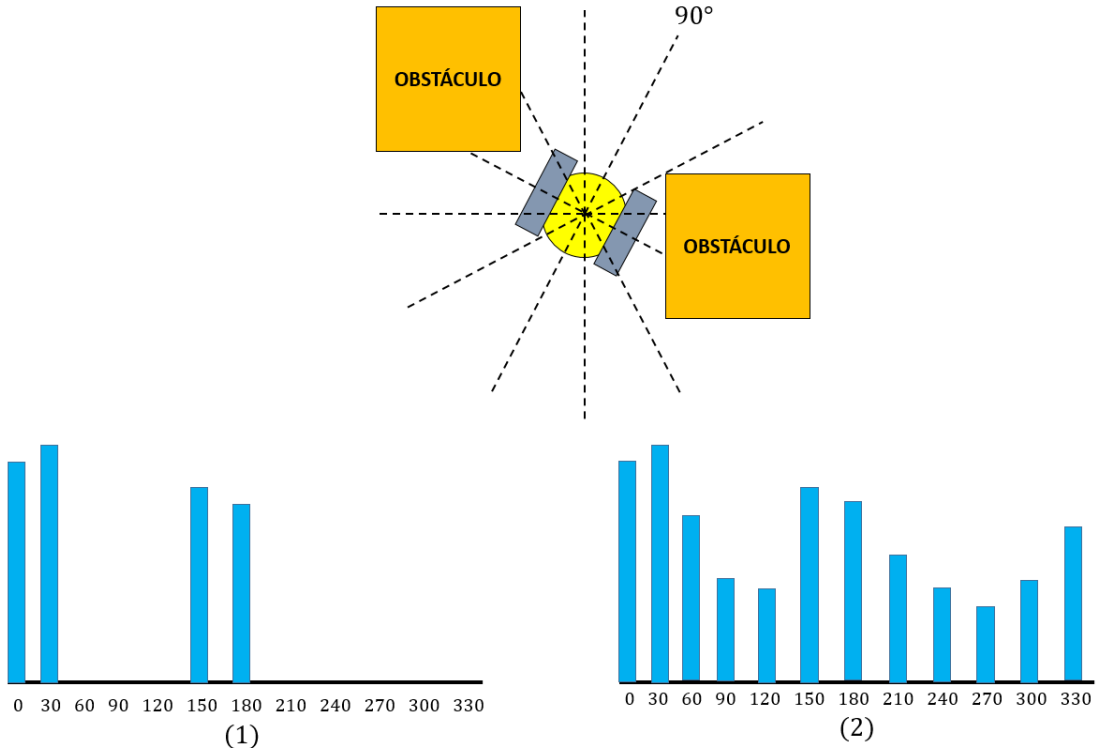



**Figura 55.** Histogramas polares 1D basados en magnitud originales (1) y magnitudes recalculadas (2).

Una vez calculada la magnitud de ocupación de cada sector, se inicia el segundo paso en el proceso de reducción, el cual es la discretización de las magnitudes. La discretización se realiza tomando en cuenta un umbral predeterminado, de esta forma, cualquier sector cuya magnitud se encuentre

por debajo de este umbral, será reconocido como un sector “libre” para circulación.

En la Figura 56 se puede observar la discretización del segundo histograma de la Figura 56 bajo un umbral de valor “X”. Se le asigna el valor de 1 a un sector considerado “libre” y el valor de 0 a uno considerado “ocupado”.

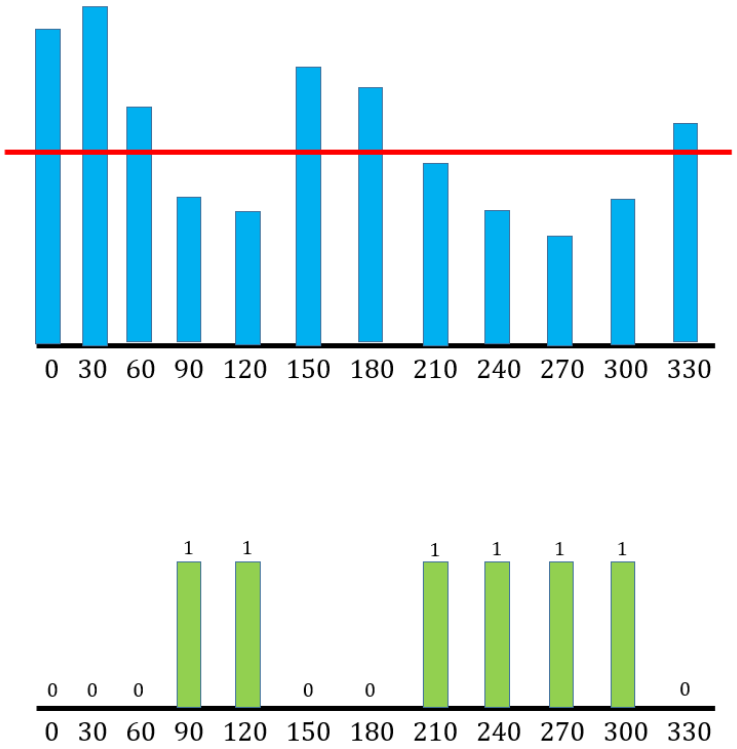



**Figura 56.** Discretización de magnitudes de ocupación de cada sector.

Tras la discretización, el siguiente paso es la detección de espacios lo suficientemente grandes como para permitir el paso del robot móvil, a estos espacios se los llama “valles”. Un valle está compuesto por la unión de dos o más sectores, el número mínimo de sectores que deberá de tener un valle dependerá del tamaño del robot y del tamaño de los sectores que conforman el valle. Por lo tanto, si un sector libre se encuentra solo, no puede ser considerado un valle y por consiguiente tampoco es considerado un camino viable para el robot móvil. En la Figura 57 puede observarse ejemplos de valles, y sectores libres individuales los cuales no son considerados como valles. Nótese que es considerado como valle a la unión entre el sector de 330° y 0°, esto debido a que, aunque no se aprecie en el histograma, estos dos son sectores contiguos.

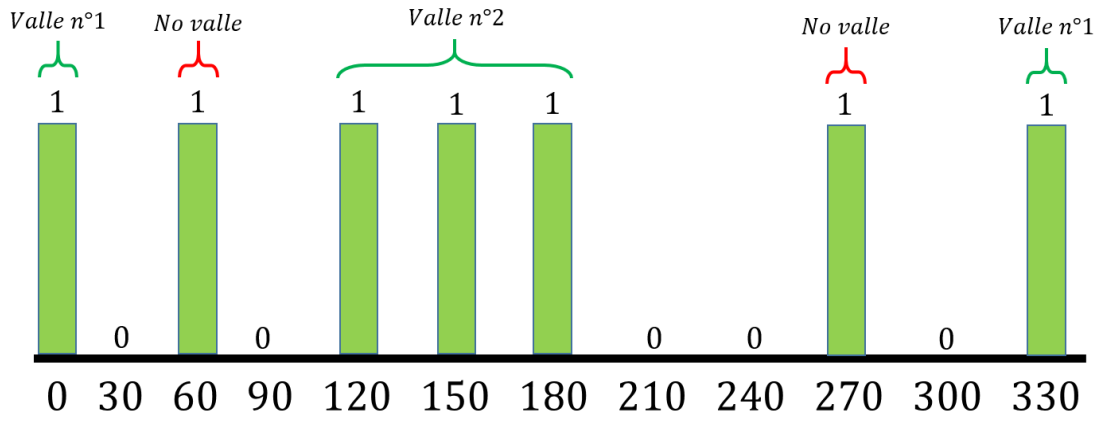



**Figura 57.** Identificación de valles tras discretización de sectores.

Los valles identificados tienen dos clasificaciones separadas entre sí por un umbral, el cual es un rango angular predeterminado. Si un valle tiene un rango angular mayor o igual a dicho umbral, este es considerado como un valle “amplio”. Por otro lado, si un valle tiene un rango menor al umbral, este es considerado como un valle “estrecho”.

En la Figura 58 se puede observar la clasificación de varios valles, para lo cual se define un umbral de 90°, por lo que cualquier valle mayor o igual a dicho umbral es considerado como un valle “amplio”. Cabe destacar que el umbral puede tener distintos valores dependiendo de la situación, pero para este caso se usa 90°.

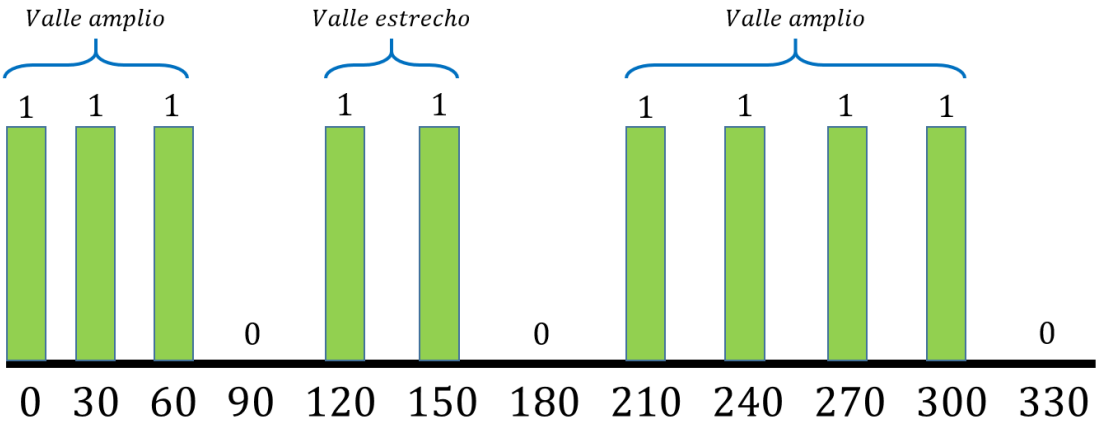



**Figura 58.** Clasificación de valles identificados.

Con la identificación de los valles, lo cual se traduce en la identificación de los camino viables para el robot móvil, el paso siguiente identificar la dirección en la que se encuentra el objetivo y la dirección que seguirá el robot para llegar a dicho objetivo. La elección del camino a seguir por el robot móvil, es elegida dependiendo de tres posibles casos.

El primero de estos casos sucede la dirección del objetivo se encuentra dentro de uno de los valles identificados. En dicho caso, el robot móvil se dirigirá directamente en la dirección del objetivo, como se puede en la Figura 59.

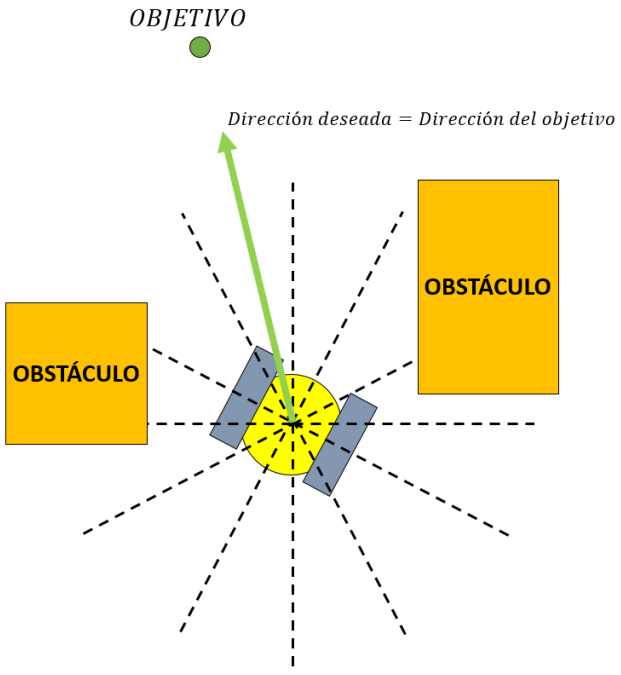



**Figura 59.** Cálculo de dirección VFH caso 1.

El segundo caso ocurre cuando el objetivo no se encuentra en una dirección dentro de uno de los valles identificados mientras el robot transita por un espacio estrecho y por tanto un valle estrecho. En este caso, se selecciona el valle más cercano a la dirección del objetivo y se calcula la dirección a seguir mediante la siguiente expresión. En la Figura 60 se puede observar un ejemplo de este segundo caso.

$$\theta = \frac{K_n + K_f}{2} \qquad [23]$$

Donde:

$\theta$: Dirección angular deseada para el robot móvil.
$K_n$: Ángulo inicial del sector seleccionado.
$K_f$: Ángulo final del sector seleccionado.

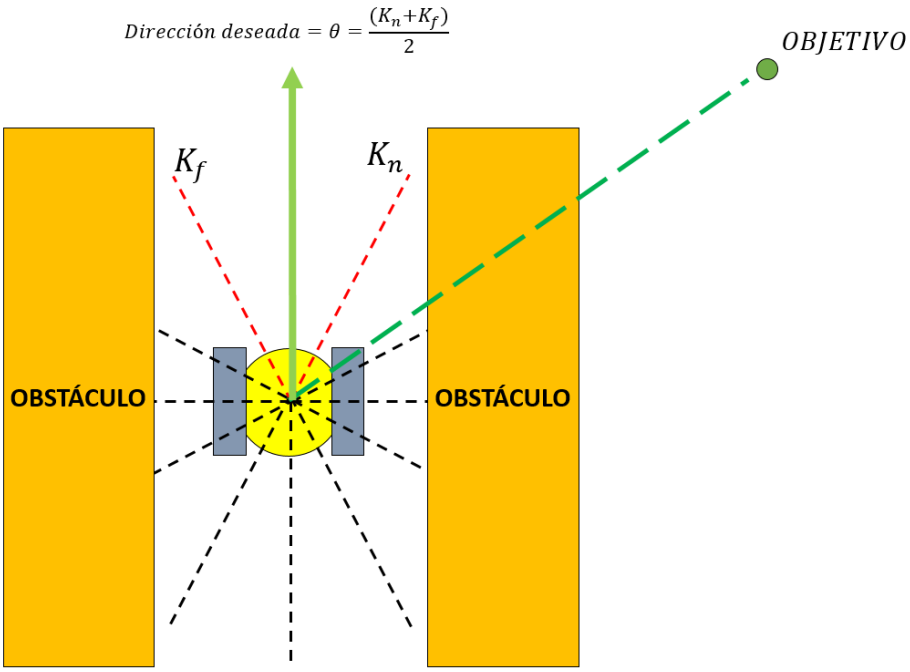



**Figura 60.** Cálculo de dirección VFH caso 2.

El último caso se suscita cuando el objetivo no se encuentra dentro de uno de los valles identificados mientras que el robot transita por un espacio amplio y por tanto un valle amplio. En este caso, la dirección es calculada mediante la misma ecuación [23] con la única diferencia que, ya que el valle es bastante amplio, el ángulo final $K_f$ ya no será el ángulo final del valle, sino la suma del ángulo inicial del valle más un valor predeterminado $S_{max}$. En la Figura 61 se puede observar un ejemplo de ese tercer caso.

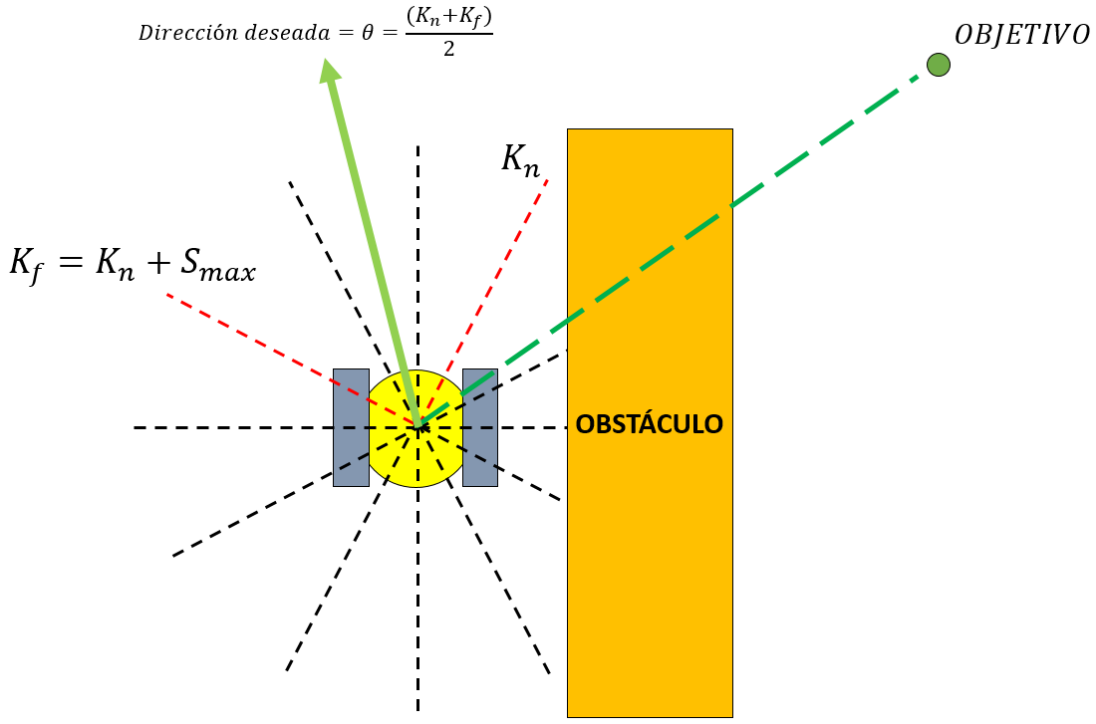



**Figura 61.** Cálculo de dirección VFH caso 3.

### 2.5.5.2. Implementación VFH

Para la implementación del algoritmo VFH, es necesario considerar que este es un algoritmo de planificación local, por lo que la implementación del mismo debe ser realizada dentro de la simulación.

Se utiliza nuevamente el bloque ***LidarSensor***. Para el caso de este proyecto, se declaran 24 sensores alrededor de todo el robot, de manera que cada sensor representa un sector con un rango 15°. En la Tabla 2 se puede ver a detalle los sensores declarados junto con sus respectivos rangos.

**Tabla 2.** Posiciones de sensores LIDAR para implementación de algoritmo VFH.

| Sensor | Posición Angular |
|---|---|
| **1** | 0° |
| **2** | 15° |
| **3** | 30° |
| **4** | 45° |
| **5** | 60° |
| **6** | 75° |
| **7** | 90° |
| **8** | 105° |
| **9** | 120° |
| **10** | 135° |
| **11** | 150° |
| **12** | 165° |
| **13** | 180° |
| **14** | 195° |
| **15** | 210° |
| **16** | 225° |
| **17** | 240° |
| **18** | 255° |
| **19** | 270° |
| **20** | 285° |
| **21** | 300° |
| **22** | 315° |
| **23** | 330° |
| **24** | 345° |

Tras la declaración de los sensores, se aplica una máquina de estados para la descomposición del vector en el que vienen las lecturas, tal y como se hizo en el algoritmo VFF. Tras obtener las lecturas de cada sensor, se aplican dos funciones de MatLab consecutivas, la primera de ellas realiza el cálculo y recálculo de las magnitudes de ocupación mientras que la segunda realiza la discretización de los sectores según su magnitud de ocupación. Con estos bloques que se pueden observar en la Figura 62, se completa el proceso de reducción de datos mencionado anteriormente.

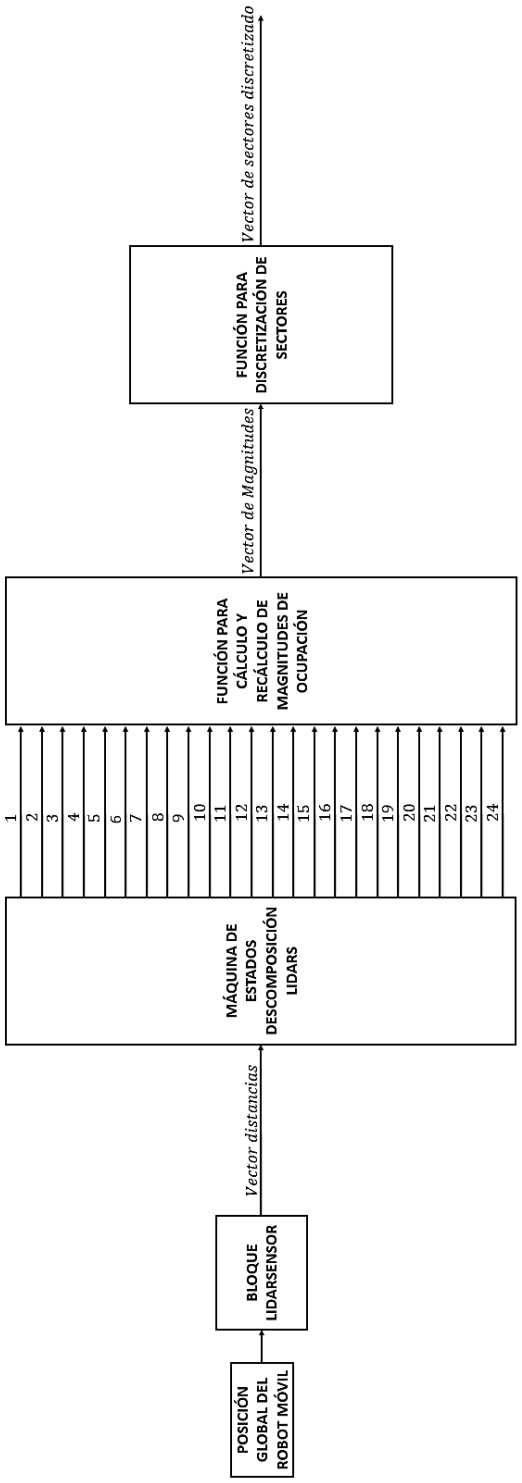



**Figura 62.** Bloques para reducción de datos en algoritmo VFH.

Tras la discretización de los sectores, se aplican otras tres funciones de MatLab consecutivas que pueden observarse en la Figura 63. La primera de ellas realiza una segregación de los sectores libres a ser evaluados, la segunda evalúa los sectores libres y los ordena en forma de valles, y la tercera identifica el inicio, final y extensión de cada valle (información de los valles).

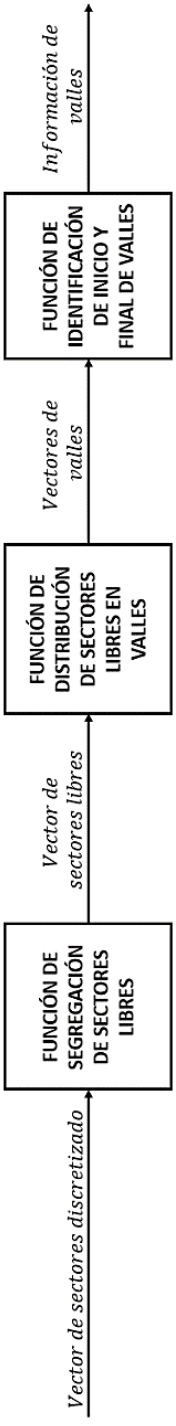



**Figura 63.** Bloques para diferenciación de valles.

La información de los valles es utilizada dentro de una función de MatLab que coteja la dirección angular del objetivo con los valles disponibles para circulación, determina el caso de direccionamiento, y calcula la dirección angular deseada para el robot móvil.

Las velocidades lineal y angular deseadas para el robot móvil son calculadas mediante una función de MatLab y una máquina de estados respectivamente, las cuales se pueden observar en la Figura 64. Adicional, se agrega una función de acondicionamiento de ángulos ya que, como se puede notar en la Figura 55, la perspectiva angular del robot móvil es diferente a su posición angular global dentro del mapa y necesita ser acondicionada para tener una correcta navegación. Para este acondicionamiento se utiliza la posición global del robot móvil junto con la posición del objetivo para calcular el ángulo en el que se encuentra el objetivo tanto desde la perspectiva global como desde la perspectiva del robot.

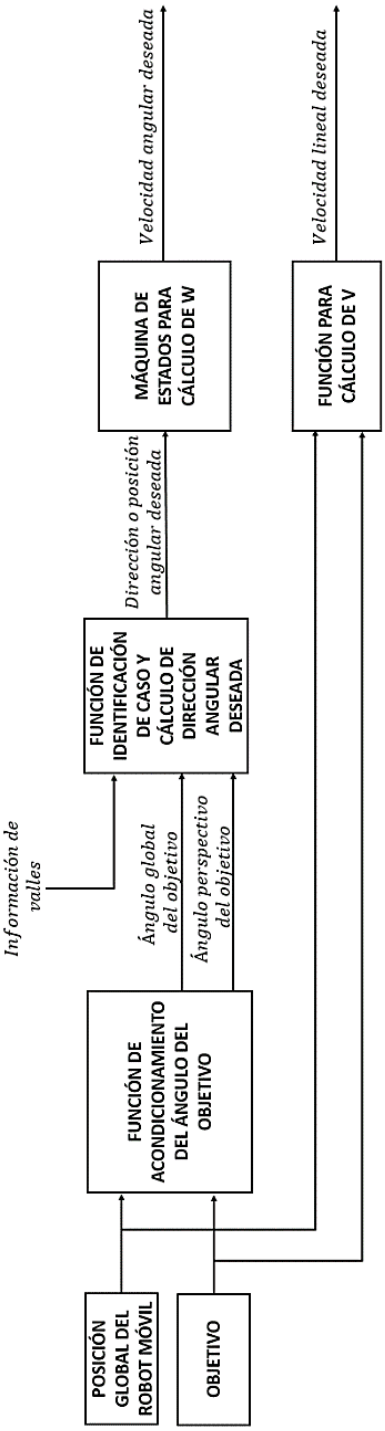



**Figura 64.** Bloques para cálculo de Velocidades lineal y angular deseadas para el robot móvil.

La integración de los bloques presentados en las últimas 3 Figuras da como resultado el sistema de bloques presentado en la Figura 65, el cual contiene dentro de sí todo lo requerido para la correcta implementación del algoritmo VFH.

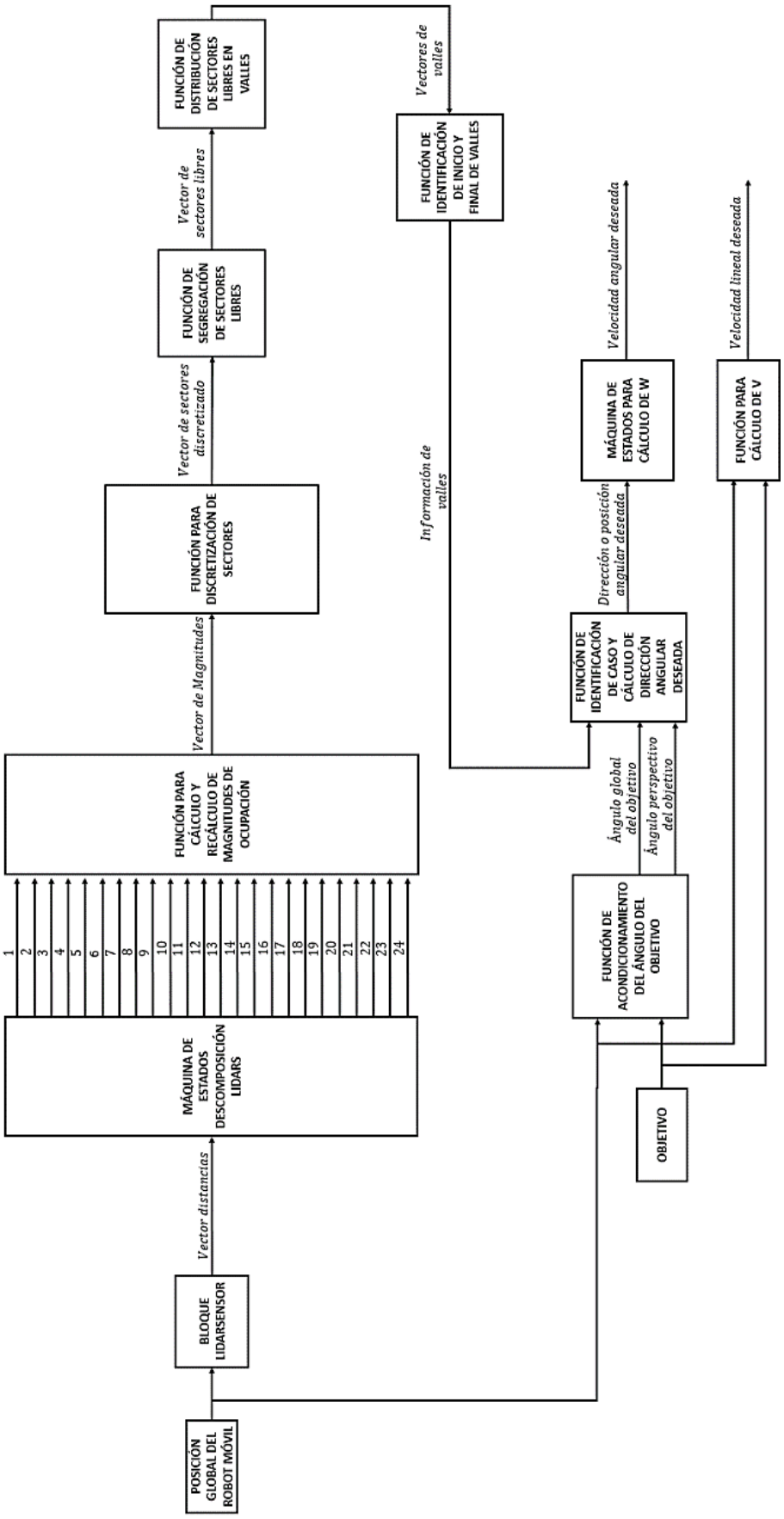



**Figura 65.** Sistema de bloques para implementación de algoritmo VFH.

Dado este sistema de bloques, la lógica del algoritmo VFH que es llevada a cabo se puede ver representada en la Figura 66.

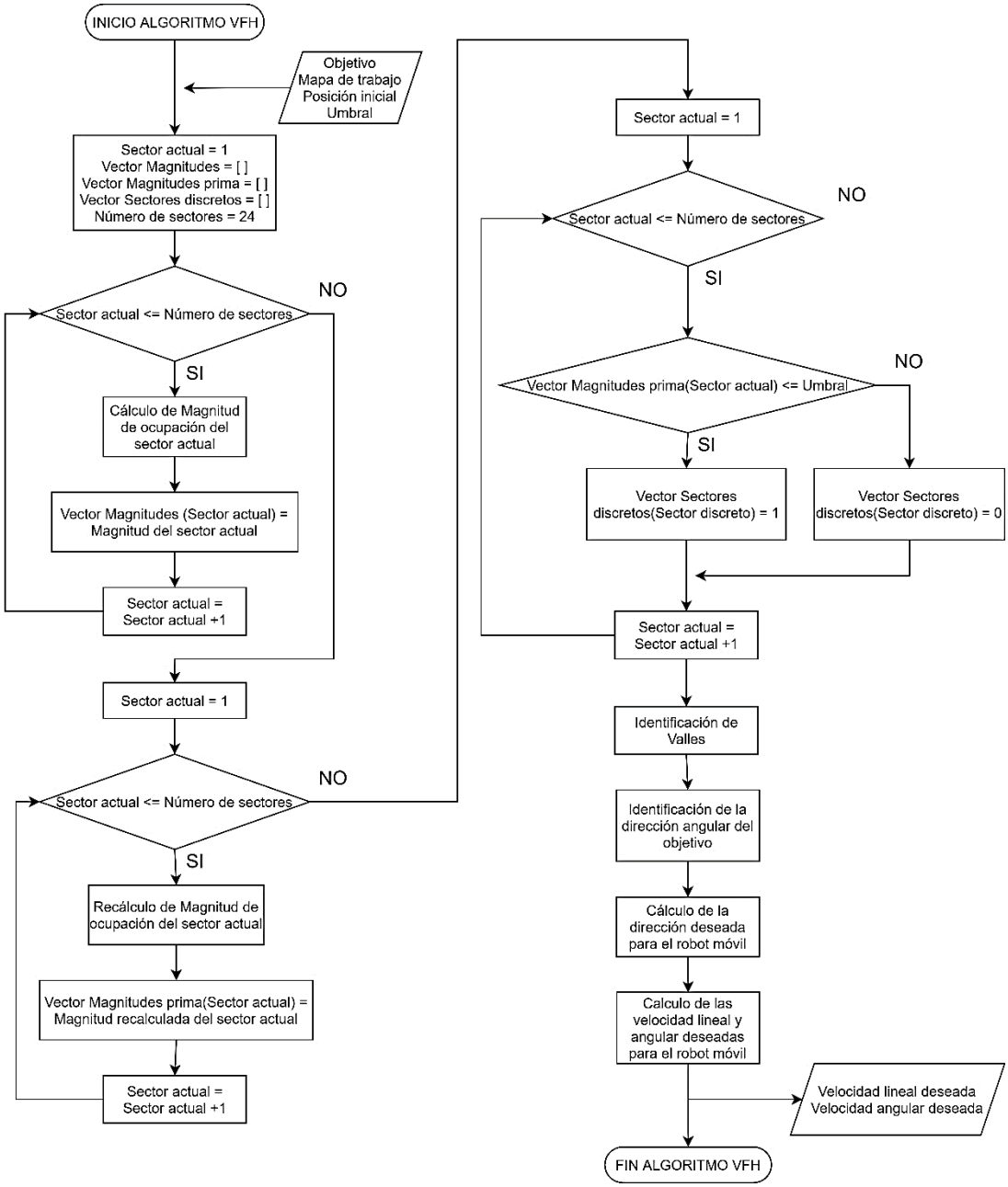



**Figura 66.** Diagrama de flujo algoritmo VFH.

### 2.5.5.3. Simulación VFH

Para la simulación del algoritmo VFH, simplemente se añade los bloques del sistema de navegación del robot móvil a los bloques de implementación del algoritmo. El sistema de bloques completo para la simulación del algoritmo VFH se puede ver en la Figura 67.

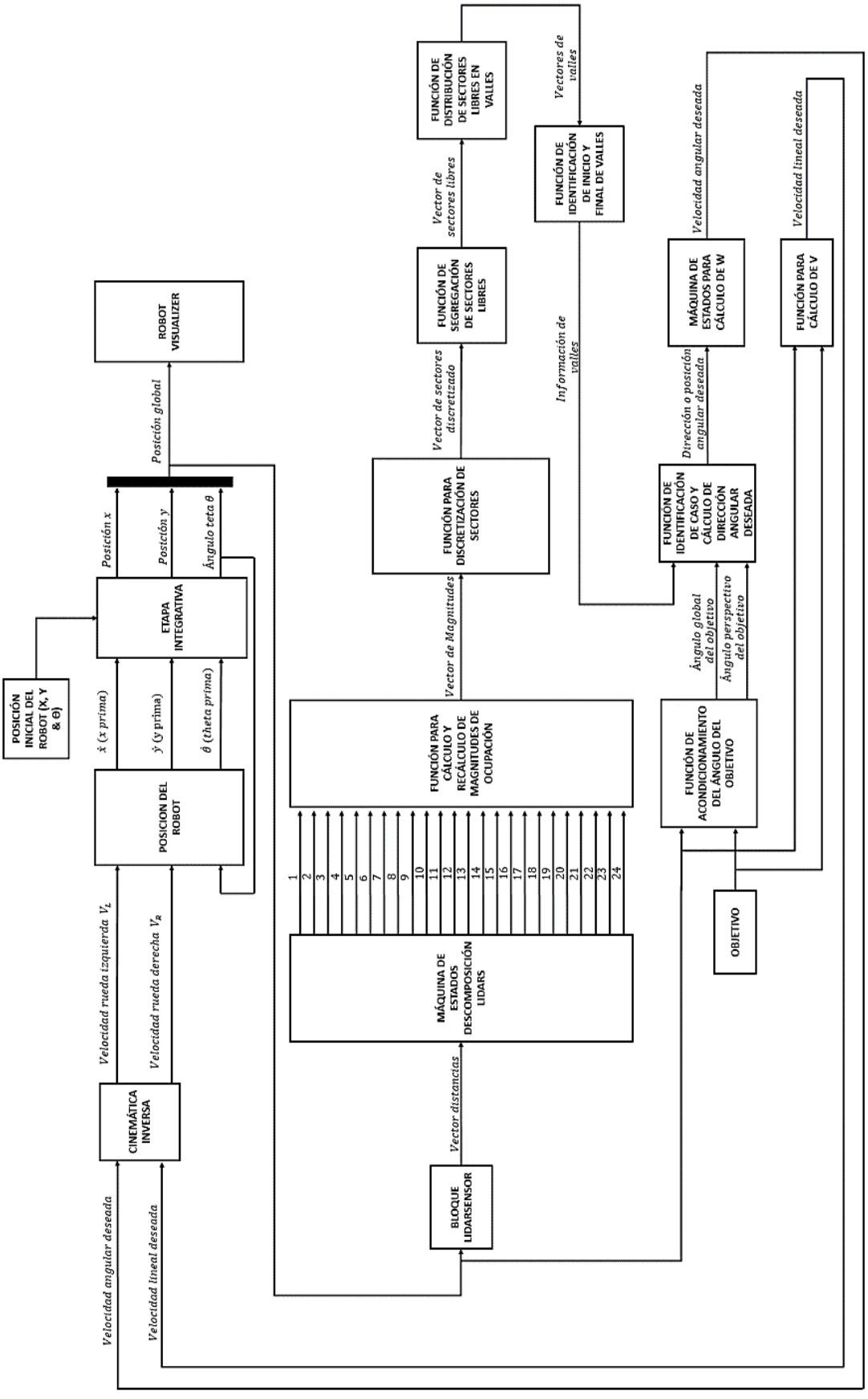



**Figura 67.** Sistema de bloques para Simulación de Algoritmo VFF.

El resultado final de este sistema de bloques se pude observar en la Figura 68. Como en los pasados algoritmos, el robot móvil es ordenado a moverse desde la posición (0.2, 0.2) a la posición (1.8, 1.8) dentro del mapa de comprobación mostrado en la Figura 19.

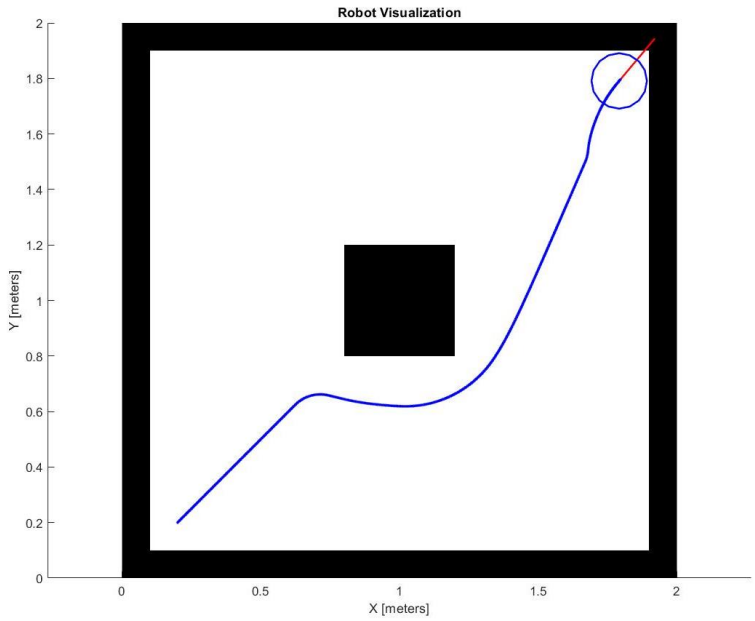



**Figura 68.** Simulación de robot móvil guiado por algoritmo VFH.

## 2.6. RUTAS DE PRUEBA

Para la comprobación del comportamiento y rendimiento de los distintos algoritmos de evasión de obstáculos, se definen 10 rutas de prueba dentro del Mapa de prueba de comportamiento de algoritmos mostrado en la Figura 20. Cada ruta de prueba se conforma por una posición de salida o inicial y una posición objetivo o final.

En la Tabla 3 se pueden observar las 10 rutas de prueba definidas, nótese que cada ruta tiene una posición angular inicial específica la cual tiene el objetivo dar una orientación previa al robot móvil.

**Tabla 3.** Rutas de prueba.

| RUTA | POSICIÓN DE SALIDA | | | POSICIÓN OBJETIVO | |
|---|---|---|---|---|---|
| | Posición inicial en X [m] | Posición inicial en Y [m] | Ángulo inicial Θ [grados] | Posición final en X [m] | Posición final en Y [m] |
| 1 | 0,3 | 0,3 | 0 | 4,5 | 4,5 |
| 2 | 4,5 | 0,5 | 180 | 0,4 | 4,6 |
| 3 | 4,7 | 4,7 | 225 | 1 | 0,5 |
| 4 | 0,5 | 4,7 | 315 | 4,6 | 0,4 |
| 5 | 2,5 | 1,1 | 0 | 2,5 | 3,9 |
| 6 | 3,8 | 2 | 225 | 1 | 0,7 |
| 7 | 0,5 | 2 | 0 | 4,6 | 0,5 |
| 8 | 0,5 | 2,8 | 0 | 4,6 | 2,1 |
| 9 | 2,1 | 3,7 | 0 | 2,5 | 1,3 |
| 10 | 4 | 0,4 | 180 | 1,4 | 3 |

El rendimiento de los algoritmos de evasión de obstáculos es medido mediante dos sencillos bloques de MatLab que son añadidos a la simulación de cada algoritmo. El primero de estos bloques es un Integrador el cual es conectado a la salida de la Velocidad lineal del robot, con esto se logra una medición precisa de la distancia recorrida por el robot móvil. El segundo bloque consiste de una máquina de estados que tiene la función de cronometrar el tiempo que demora en llegar el robot móvil desde su posición inicial hasta su posición final. Esta máquina de estados cronometra por décimas de segundo el tiempo transcurrido y se detiene cuando la posición actual del robot móvil se encuentra a una distancia menor o igual a un rango de llegada predeterminado que para este caso es 0.01 [m]. En la Figura 69 se pueden observar los dos bloques mencionados, cabe destacar que también se contabiliza el tiempo de planificación de los algoritmos de planificación global (RRT, RTT* & PFPP), esto mediante un cronómetro externo.

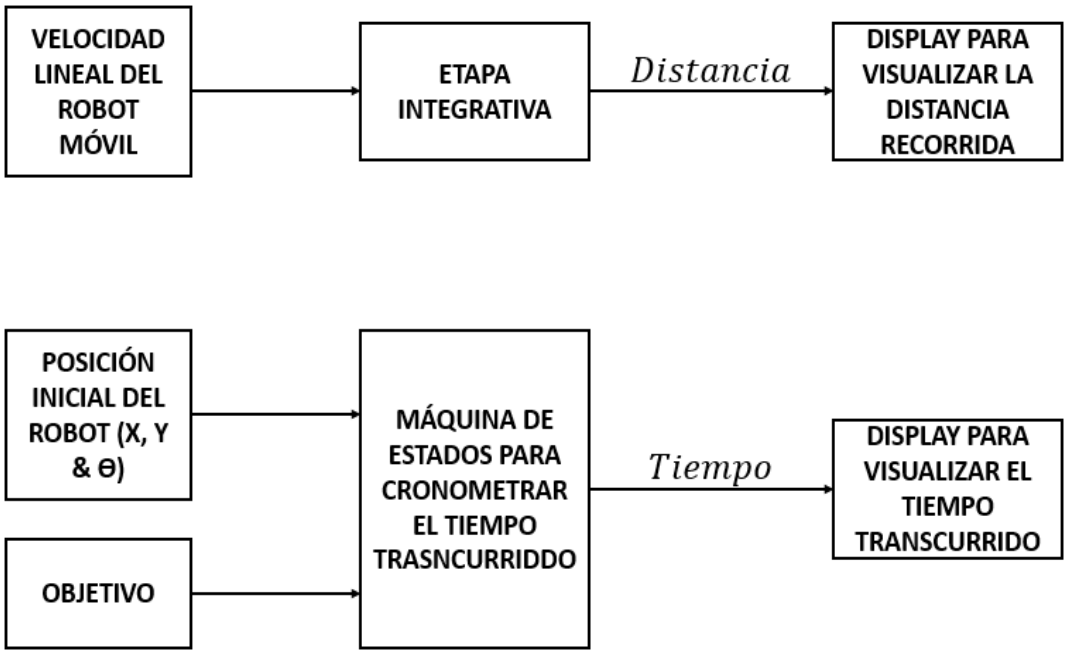



**Figura 69.** Sistema de bloques para medición de rendimiento de algoritmos de evasión de obstáculos.

# 3. RESULTADOS Y DISCUSIÓN

## 3.1. RENDIMIENTO

Es necesario mencionar que, para el presente proyecto, se empleó una computadora con las siguientes características para la planificación y simulación de los algoritmos. Por lo que los tiempos de planificación de ruta, para los algoritmos de tipo planificación global, podrían variar en otro equipo.

- Procesador: Intel ® Core ™ i7-7700HQ CPU @ 2.8GHz
- Memoria RAM: 16GB DDR4, 2667 MHz, doble canal
- Disco duro: SATA 1000GB
- Memoria Gráfica: GeForce GTX 1060 6GB
- Versión de Windows: Windows 10

Cabe destacar que, durante las pruebas, el programa MatLab tuvo un uso promedio de la CPU de un 27% y un uso promedio de 2.1 GB de la memoria RAM.

### 3.1.1. RRT

El rendimiento del algoritmo RRT dentro de las rutas predefinidas se puede observar en la Tabla 4.

**Tabla 4.** Tabla de rendimiento de tiempo y distancia de algoritmo RRT.

| Rapidly Random Tree | | | | |
|---|---|---|---|---|
| **RUTA** | **Tiempo (Planificación) [s]** | **Tiempo (Simulación) [s]** | **Tiempo total [s]** | **Distancia [m]** |
| **1** | 134,50 | 112,40 | 246,90 | 8,075 |
| **2** | 106,53 | 107,80 | 214,33 | 7,423 |
| **3** | 35,57 | 100,50 | 136,07 | 7,049 |
| **4** | 129,89 | 112,60 | 242,49 | 7,755 |
| **5** | 137,60 | 72,00 | 209,60 | 4,016 |
| **6** | 136,48 | 92,20 | 228,68 | 6,174 |
| **7** | 396,09 | 90,00 | 486,09 | 6,344 |
| **8** | 94,71 | 80,00 | 174,71 | 5,629 |
| **9** | 102,27 | 72,70 | 174,97 | 5,182 |
| **10** | 117,69 | 76,60 | 194,29 | 5,102 |

La gran variación entre los tiempos de planificación de cada ruta predefinida. Debido a la naturaleza aleatoria de este algoritmo, el tiempo de planificación

requerido para llegar desde un punto A hasta un punto B puede variar en gran medida y de ninguna forma puede predecirse un tiempo aproximado. Esto provoca que la planificación de ruta para llegar desde un punto A hasta un punto B sea muy inestable, pudiendo demorar desde unos pocos segundos (Ruta 3), hasta varios minutos (Ruta 7). Adicional cabe destacar que el tiempo de planificación requerido es proporcional al tamaño del mapa, por lo que en mapas de gran amplitud, el algoritmo tardaría demasiado en encontrar un vector solución.

Por consecuencia, el algoritmo RRT, aunque efectivo y rápido en espacios pequeños, no tiene un buen rendimiento en espacios de gran amplitud, lo que lo hace poco atractivo para aplicaciones en la realidad.

### 3.1.2. RRT*

El rendimiento del algoritmo RRT* dentro de las rutas predefinidas se puede observar en la Tabla 5.

**Tabla 5.** Tabla de rendimiento de tiempo y distancia de algoritmo RRT*.

| Rapidly Random Tree Star | | | | |
|---|---|---|---|---|
| **RUTA** | **Tiempo (Planificación) [s]** | **Tiempo (Simulación) [s]** | **Tiempo total [s]** | **Distancia [m]** |
| **1** | 268,55 | 93,90 | 362,45 | 7,009 |
| **2** | 684,06 | 83,50 | 767,56 | 6,339 |
| **3** | 459,57 | 90,50 | 550,07 | 6,785 |
| **4** | 826,30 | 94,40 | 920,70 | 7,027 |
| **5** | 95,96 | 56,80 | 152,76 | 3,781 |
| **6** | 208,48 | 61,10 | 269,58 | 4,167 |
| **7** | 286,91 | 75,10 | 362,01 | 5,62 |
| **8** | 88,85 | 64,90 | 153,75 | 4,761 |
| **9** | 84,97 | 61,00 | 145,97 | 4,598 |
| **10** | 229,51 | 58,70 | 288,21 | 4,35 |

Al igual que con el algoritmo RRT, el algoritmo RRT* destaca por la gran variación que existe entre los tiempos de planificación de cada ruta, siendo que uno de ellos (ruta 9) se resuelve en poco más de un minuto mientras que otro (ruta 4) se resuelve en poco menos de 14 minutos. La abismal diferencia se debe a que el algoritmo RRT* realiza una comprobación de sus alrededores cada vez que se crea un nuevo nodo, por lo que con el incremento de nodos también se incrementa la cantidad de nodos vecinos que deben analizarse cada vez que se añade uno nuevo.

A pesar de que las distancias recorridas en cada ruta se reducen con respecto al algoritmo RRT, el aumento de los tiempos de planificación hace que el algoritmo RRT* no tenga un buen rendimiento y por lo tanto no sea un buen candidato para aplicaciones en la realidad.

### 3.1.3. POTENTIAL FIELD PATH PLANNING

El rendimiento del algoritmo PFPP dentro de las rutas predefinidas se puede observar en la Tabla 6.

**Tabla 6.** Tabla de rendimiento de tiempo y distancia de algoritmo PFPP.

| Potential Field Path Planning | | | | |
|---|---|---|---|---|
| **RUTA** | **Tiempo (Planificación) [s]** | **Tiempo (Simulación) [s]** | **Tiempo total [s]** | **Distancia [m]** |
| **1** | 292,74 | 86,40 o | 379,14 | 7,036 |
| **2 (no logrado)** | 295,19 | 59,90 | 355,09 | 4,776 |
| **3** | 300,89 | 79,00 | 379,89 | 6,276 |
| **4** | 318,84 | 83,90 | 402,74 | 6,856 |
| **5(no logrado)** | 307,01 | 14,70 | 321,71 | 0,536 |
| **6** | 291,74 | 40,60 | 332,34 | 3,2 |
| **7 (no logrado)** | 354,26 | 38,20 | 392,46 | 3,04 |
| **8** | 300,06 | 55,80 | 355,86 | 4,608 |
| **9 (no logrado)** | 330,06 | 32,00 | 362,06 | 2,28 |
| **10** | 308,19 | 50,60 | 358,79 | 4,136 |

El algoritmo PFPP muestra un tiempo de planificación estable que es proporcional al área del mapa analizado. Para este proyecto, un mapa de 50x50 cuadrículas tuvo un tiempo de planificación promedio de 5 minutos (300 segundos). A pesar de ser un tiempo estable, 5 minutos no puede considerarse como un tiempo óptimo para la planificación de una solución. Sin embargo, este tiempo podría reducirse focalizando la capacidad del procesador a dicha tarea.

La naturaleza del algoritmo PFPP hace que en ciertos puntos el algoritmo llegue a lo que se conoce como una situación de “mínimo local”. En esta situación, a pesar de no haber llegado al punto mínimo de todo el mapa, el robot móvil se encuentra atrapado en un punto donde todos los puntos alrededor de sí mismo tienen un potencial mayor al potencial del punto actual, lo que provoca que el robot no pueda escapar de su posición actual. Es esta

situación la que provocó que ciertas rutas de prueba no pudieran ser completadas con este algoritmo.

Por consecuencia, y a pesar de que el tiempo de planificación y distancia recorrida sean menores en las rutas completadas por este algoritmo en comparación con los algoritmos anteriores, el algoritmo PFPP no llega a tener un rendimiento óptimo e incluso puede fallar frente a ciertas condiciones. Lo cual lo descarta como candidato para aplicaciones en la realidad.

### 3.1.4. VFF

El rendimiento del algoritmo PFPP dentro de las rutas predefinidas se puede observar en la Tabla 7.

**Tabla 7.** Tabla de rendimiento de tiempo y distancia de algoritmo VFF.

| Virtual Force Field | | |
|---|---|---|
| **RUTA** | **Tiempo [s]** | **Distancia [m]** |
| **1** | 83,50 | 6,729 |
| **2** | 83,30 | 6,57 |
| **3** | 78,30 | 6,337 |
| **4** | 90,90 | 7,119 |
| **5** | 42,30 | 3,425 |
| **6** | 38,60 | 3,19 |
| **7** | 55,80 | 4,565 |
| **8** | 52,60 | 4,313 |
| **9** | 53,80 | 4,315 |
| **10** | 51,20 | 4,216 |

El algoritmo VFF, al ser un algoritmo de planificación local, no tiene la necesidad de planificar su ruta previamente, sino que esta es planificada en tiempo real en medio de la simulación, lo cual le proporciona una ventaja inmensa sobre otros algoritmos ya que el tiempo necesario para pasar desde un punto A hasta un punto B se acorta en gran medida.

La distancia recorrida por este algoritmo también es un punto atractivo del mismo. Dentro de las rutas definidas, el algoritmo VFF demostró poder completar todas las rutas con una distancia recorrida menor o igual a algoritmos anteriores.

Otro punto a destacar es la capacidad de reacción del algoritmo ante obstáculos imprevistos y el hecho de que este carece de la necesidad de tener información previa del mapa donde se desplazará el robot móvil. Sin embargo, cabe señalar que, para emplear este algoritmo, es mandatorio emplear cinco

o más sensores de distancia dependiendo de la precisión de navegación que se quiera conseguir. Para este proyecto, el número de sensores empleados fue 9 repartidos en un rango de 180° en la parte frontal del robot móvil.

El algoritmo VFF, presenta un rendimiento óptimo y puede ser considerado como un buen candidato para aplicarse en la realidad. Sin embargo cabe destacar que la capacidad reactiva del algoritmo también puede llegar a representar una debilidad, ya que, si se llegaran a producir falsas lecturas de sensores, el robot móvil podría ser desviado bruscamente de su camino, lo cual sería causa de una potencial colisión.

### 3.1.5. VFH

El rendimiento del algoritmo PFPP dentro de las rutas predefinidas se puede observar en la Tabla 8.

**Tabla 8.** Tabla de rendimiento de tiempo y distancia de algoritmo VFH.

| Vector Field Histogram | | |
|---|---|---|
| **RUTA** | **Tiempo [s]** | **Distancia [m]** |
| **1** | 82,50 | 6,687 |
| **2** | 79,70 | 6,46 |
| **3** | 76,20 | 6,23 |
| **4** | 79,20 | 6,423 |
| **5** | 41,50 | 3,294 |
| **6** | 39,30 | 3,123 |
| **7** | 56,20 | 4,482 |
| **8** | 52,50 | 4,186 |
| **9** | 53,80 | 4,291 |
| **10** | 51,50 | 4,104 |

Al igual que el algoritmo VFF, el algoritmo VFH es de planificación local, por lo que no necesita información previa del mapa ni tampoco una planificación previa a la simulación, lo que le da una gran ventaja sobre los tres primeros algoritmos.

Los tiempos y la distancia recorrida en cada ruta son muy similares y en ciertos casos iguales a los del algoritmo VFF, por lo que su rendimiento también puede ser considerado como óptimo y apto para aplicaciones en la realidad. Sin embargo, es necesario aclarar que para la aplicación del algoritmo VFH se necesita un mínimo de 16 sensores de distancia repartidos alrededor de todo el robot móvil, esto con el objetivo de lograr una navegación precisa. En el caso de este proyecto se utilizaron 24 sensores.

El aumento de la cantidad de sensores trae consigo la necesidad de emplear un procesador más potente que el que se necesitaría para la ejecución algoritmos previos. De hecho, durante la ejecución de este proyecto se tuvo ciertos problemas con el procesamiento de la información recogida por lo sensores, esto debido a la gran cantidad de los mismos.

Como último punto a señalar está el hecho de que el algoritmo VFH cuenta con una característica de la cual su predecesor VFF carece, y esta es el considerar las lecturas de sensores vecinos antes de determinar el valor de ocupación de un sensor X. Esta nueva cualidad hace que el sistema de navegación sea más robusto, evita que falsas lecturas de sensores arruinen el trayecto del robot móvil hacia el objetivo final, y provoca que el robot móvil no sufra un cambio brusco de dirección frente a obstáculos emergentes.

## 3.2. COMPORTAMIENTO

En esta subsección se analiza el comportamiento del robot móvil cuando este es guiado por cada uno de los algoritmos de evasión de obstáculos. Este análisis se basa en las simulaciones a partir de las cuales se obtuvieron los resultados de la subsección anterior, dichas simulaciones pueden encontrarse al final de este documento en la sección de anexos.

### 3.2.1. RRT

En la Figura 70 se puede observar la simulación del algoritmo RRT en la ruta definida n°1. Como se puede observar en esta, la ruta seguida se caracteriza por ser sinuosa y bastante errática, esto debido a la naturaleza aleatoria del algoritmo. Aunque no llega a colisionar con ningún obstáculo, la ruta trazada por el algoritmo no es óptima e incluso puede llegar a ser difícil de seguir por el robot móvil debido a los giros tan cerrados que deben realizarse.

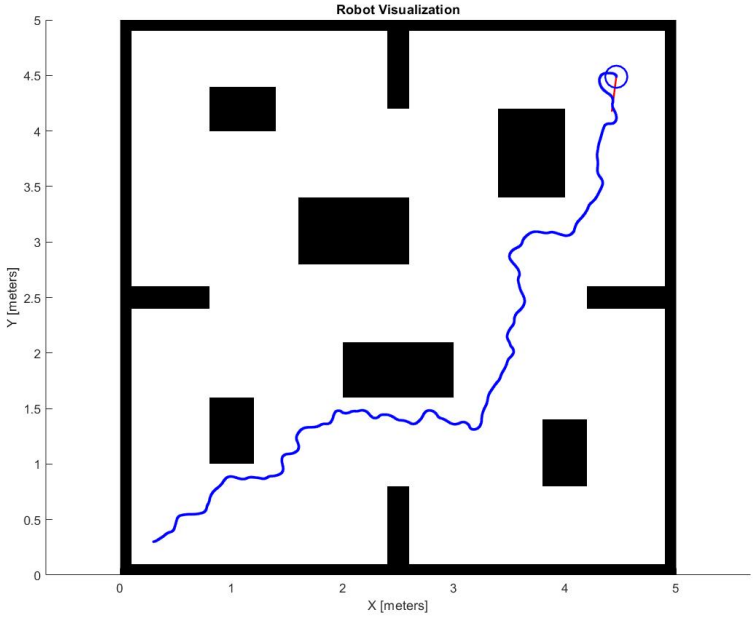



**Figura 70.** Simulación de robot móvil en ruta n°1 guiado por algoritmo RRT.

### 3.2.2. RRT*

En la Figura 71 se puede observar la simulación del algoritmo RRT* en la ruta definida n°2. Como se puede observar, la ruta seguida por el robot móvil es mucho más estable y lineal que la generada por el algoritmo RRT, esto debido a la estrategia de búsqueda del camino más corto que se añade al algoritmo. El comportamiento del robot móvil destaca por ser mucho más estable, siendo que apenas se tienen unas pequeñas fluctuaciones a lo largo de todo el camino. Adicional, a diferencia de la ruta generada por el algoritmo RRT, la ruta del algoritmo RRT* podría ser seguida por cualquier tipo de robot móvil sin importar su configuración.

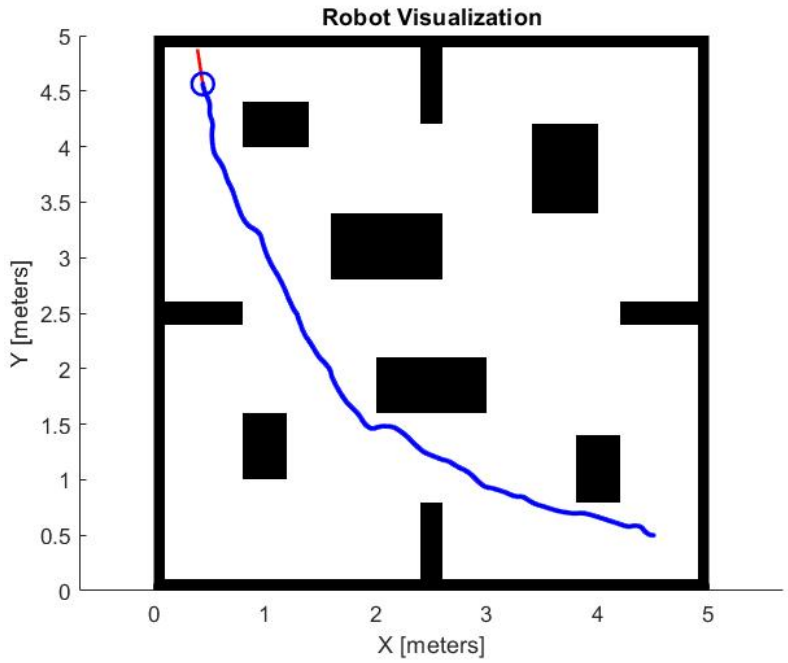



**Figura 71.** Simulación de robot móvil en ruta n°2 guiado por algoritmo RRT*.

### 3.2.3. POTENTIAL FIELD PATH PLANNING

En la Figura 72 se puede observar la simulación del algoritmo PFPP en la ruta definida n°3. Como se puede observar, la ruta seguida es impecablemente estable, llegando hasta el objetivo final sin tener ninguna fluctuación de rumbo en todo el camino. Sin lugar a dudas, el comportamiento del robot móvil guiado por el algoritmo PFPP es muy estable y lineal, por lo que se puede decir que cualquier configuración de robot móvil sería capaz de seguir la ruta trazada sin mayor dificultad.

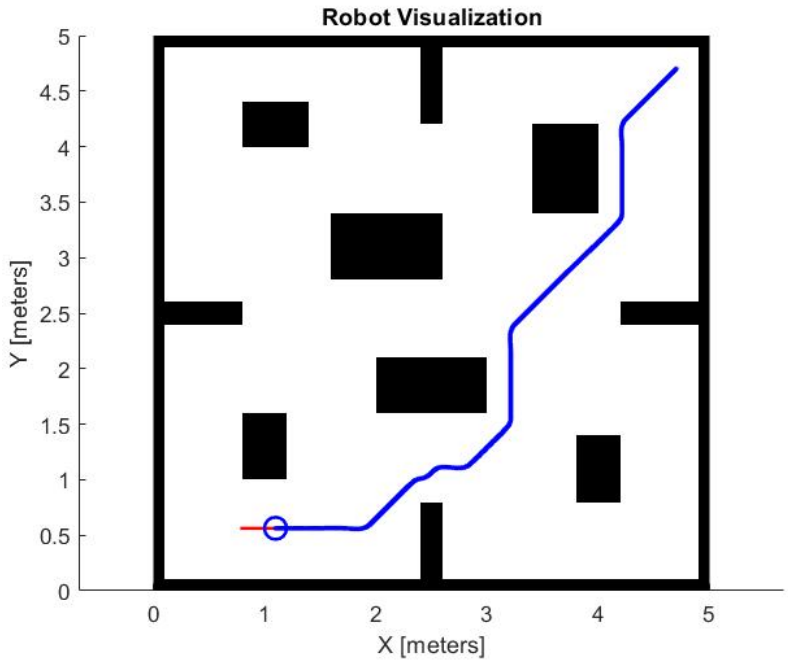



**Figura 72.** Simulación de robot móvil en ruta n°3 guiado por algoritmo PFPP.

Sin embargo, cabe destacar que el comportamiento producido por este algoritmo es estable siempre y cuando no exista la situación ya mencionada de “mínimo local”, situación en la cual el algoritmo no encontrará una solución para llegar su objetivo y por lo tanto fallará en su misión.

En la Figura 73 se puede observar un ejemplo de esta situación, en este caso, el objetivo final se encuentra exactamente al otro lado de un obstáculo (correspondiente a la ruta definida n°9), por lo que se forma un punto mínimo en el centro de uno de los lados de este obstáculo, lo que provoca que el robot móvil quede atrapado y de vueltas alrededor de dicho punto.

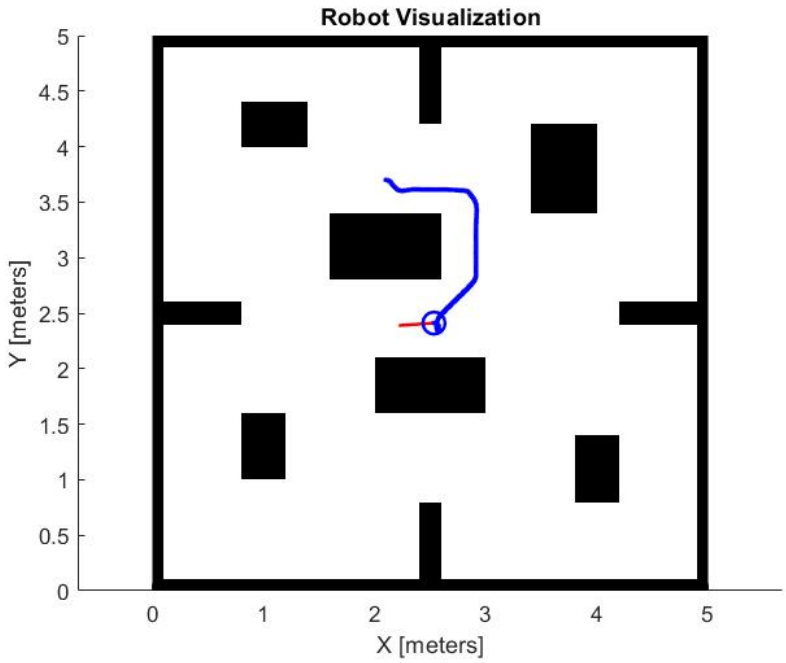



**Figura 73.** Simulación de robot móvil en ruta n°9 guiado por algoritmo PFPP.

### 3.2.4. VFF

En la Figura 74 se puede observar la simulación del algoritmo VFF en la ruta definida n°4. Como se puede observar, la ruta generada por este algoritmo es bastante estable, sin transiciones bruscas y con un esquive de obstáculos óptimo. El comportamiento del robot es muy fluido y puede decirse que la ruta generada podría ser seguida por cualquier configuración de robot móvil sin dificultad.

Sin embargo, es necesario señalar el movimiento parabólico realizado por el robot cada vez que se encuentra de frente con un obstáculo. Este movimiento parabólico, aunque eficaz para esquivar obstáculos, es ligeramente agresivo y podría ocasionar colisiones en situaciones en las que el robot móvil se desplazase por pasillos más estrechos. Esta respuesta ligeramente agresiva ya fue mencionada en la subsección anterior.

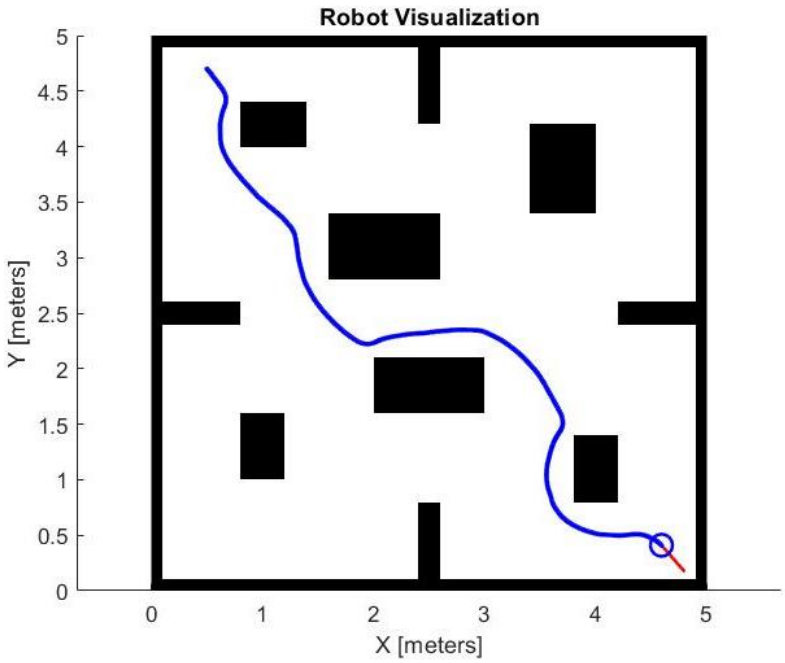



**Figura 74.** Simulación de robot móvil en ruta n°4 guiado por algoritmo VFF.

### 3.2.5. VFH

En la Figura 75 se puede observar la simulación del algoritmo VFH en la ruta definida n°3. Como se puede observar, la ruta generada por el algoritmo es bastante estable, muy similar a la generada por el algoritmo PFPP. El comportamiento del robot móvil guiado por el algoritmo es fluido y lineal, por lo que se puede decir que la ruta generada podría ser fácilmente seguida por cualquier configuración de robot móvil.

Nótese dos mejoras destacables de este algoritmo con respecto a su predecesor VFF. La primera de ellas es la linealidad de la ruta del robot, muy diferente a la generada por el algoritmo VFF, la cual era más curvada y por lo tanto menos eficiente. La segunda mejora es el cambio en la reacción del robot móvil frente a un obstáculo inminente, la cual era ligeramente agresiva en el algoritmo VFF y ahora es totalmente controlada, haciendo que el robot móvil pueda moverse de manera efectiva incluso a través de espacios muy estrechos.

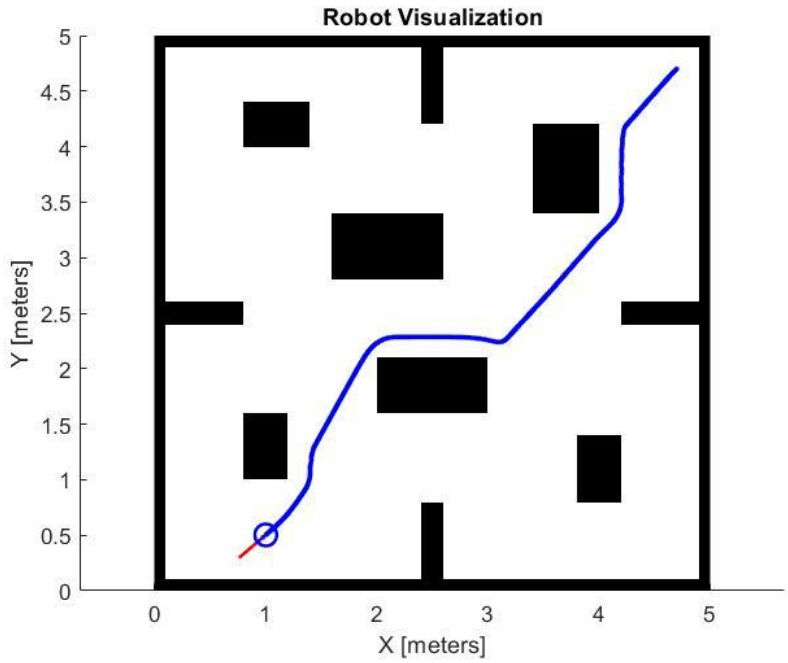



**Figura 75.** Simulación de robot móvil en ruta n°3 guiado por algoritmo VFH.

## 3.3. COMPARACIÓN DE ALGORITMOS.

En la Figura 76, se puede observar un gráfico de barras mostrando el tiempo transcurrido que necesitó cada algoritmo para llegar desde el punto inicial hasta el objetivo final en cada una de las rutas definidas. Se destacan los tiempos de los algoritmos VFF y VFH, los cuales son por mucho los más eficientes, siendo que en ninguna ruta necesitaron más de un minuto y medio para llegar desde el punto inicial hasta el objetivo final.

Los algoritmos RRT y RRT* se caracterizan por no tener un tiempo de planificación estable, el cual puede variar desde 3 minutos hasta poco más de 15 minutos. Por consecuencia, se puede decir que los algoritmos RRT y RRT* tienen un rendimiento de tiempo muy bajo en comparación de los algoritmos VFF y VFH. Además es necesario señalar que, ante espacios más amplios, los algoritmos RRT y en especial el RRT* tendrán tiempos de planificación más extensos e ineficientes.

El algoritmo PFPP ocupa un puesto intermedio entre estos dos grupos, a pesar de tener un tiempo de planificación estable de poco más de 5 minutos (en el caso de este proyecto), sigue sin poder considerarse óptimo para aplicaciones en la realidad y su rendimiento de tiempo puede ser considerado como regular.

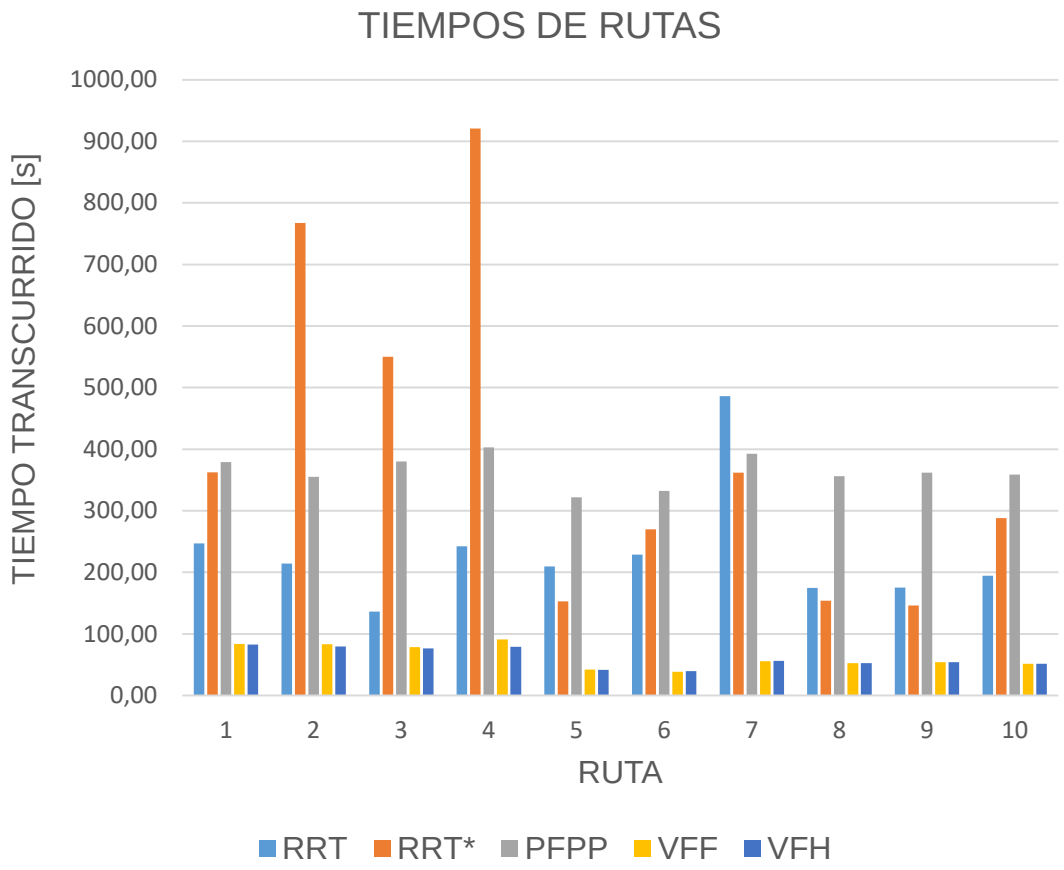



**Figura 76.** Gráfico de barras del tiempo de ruta de cada algoritmo en cada ruta de prueba.

En la Figura 77, se puede observar un gráfico de barras mostrando las distancias recorridas por cada uno de los algoritmos en cada una de las rutas definidas. Nuevamente, se destacan las distancias recorridas de los algoritmos VFF y VFH, las cuales son un poco menores a las distancias recorridas por los demás algoritmos. Sin embargo, es necesario señalar que el algoritmo VFH cuenta con un rendimiento ligeramente mejor al algoritmo VFF, esto debido a que, como se mencionó en la subsección anterior, el algoritmo VFH genera rutas más lineales y reacciones a obstáculos más controladas.

En cuanto a los algoritmo RRT y RRT*, se puede observar que el más eficiente entre estos dos es el RRT*, esto debido a la ya mencionada estrategia de búsqueda del camino más corto que se encuentra integrada al algoritmo. Adicional, se puede destacar que el algoritmo RRT* llega a producir rutas igual de eficientes que las producidas por el algoritmo VFF.

Finalmente, el algoritmo PFPP se destaca por llegar a producir rutas igualmente o incluso más eficientes que las producidas por todos los algoritmos anteriores. Sin embargo, su incapacidad ante las ya mencionadas

situaciones de “mínimo local” hace que este algoritmo no pueda ser considerado confiable.

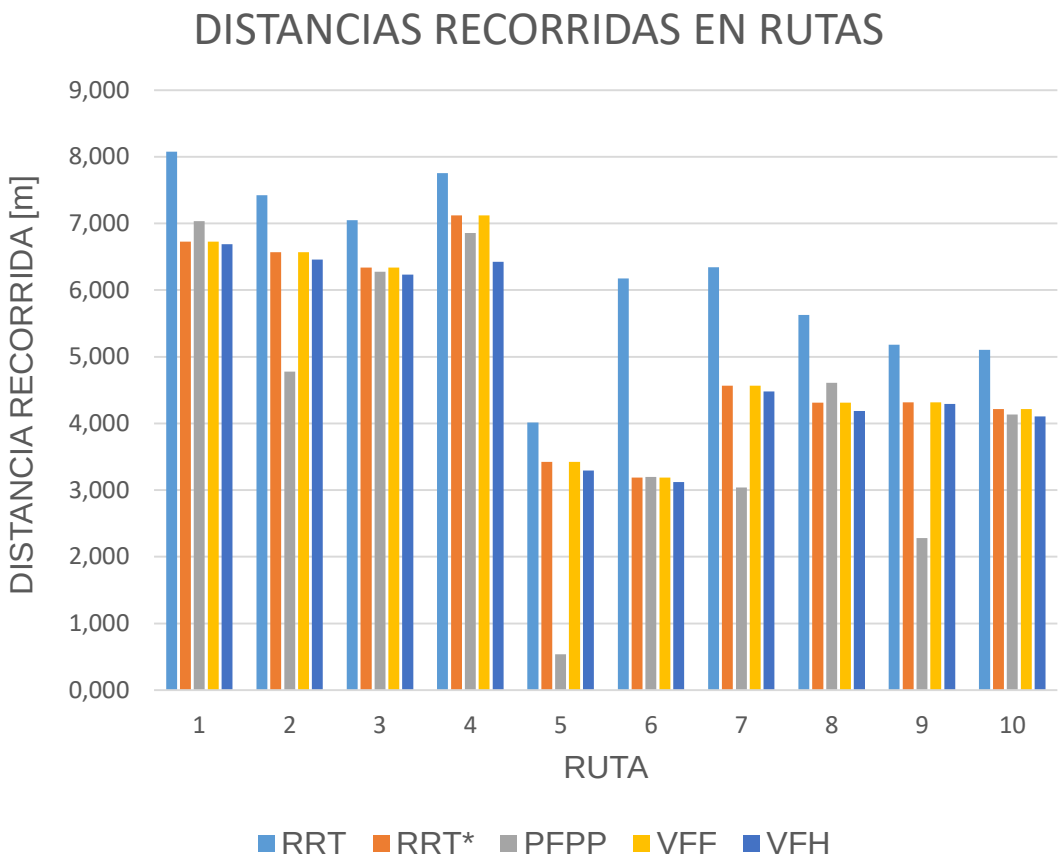



**Figura 77.** Gráfico de barras de la distancia recorrida por cada algoritmo en cada ruta de prueba.

# 4. CONCLUSIONES Y RECOMENDACIONES

## 4.1. CONCLUSIONES

Tras analizar los resultados obtenidos de las simulaciones, se concluye que los objetivos de este proyecto fueron logrados completamente. Siendo que se consiguió realizar un análisis completo del rendimiento y comportamiento de cada uno de los algoritmos de evasión de obstáculos, así como un análisis comparativo del desempeño de cada algoritmo.

Los algoritmos analizados en este proyecto se pueden dividir en dos grupos, algoritmos de planificación global y de planificación local. Los algoritmos del primer grupo se caracterizan por planificar la ruta a seguir por el robot móvil de manera previa al movimiento del mismo, es decir, el robot móvil no ejecuta ningún movimiento hasta que se haya encontrado una ruta solución con la que llegar al objetivo final. Los algoritmos del segundo grupo se caracterizan por planificar en tiempo real la ruta a seguir por el robot móvil, ruta la cual es calculada y recalculada iterativamente basándose en la información del ambiente exterior al robot que es recogida por los sensores.

Los algoritmos de planificación global como lo son el RRT, RRT* & PFPP, contienen una ventaja muy notable, y este es el hecho de que no requieren ningún tipo de sensor para su funcionamiento, basta con entregar información precisa del ambiente donde se moverá el robot móvil (mapa o croquis) para que los algoritmos encuentren una ruta solución efectiva hacia el objetivo. Por otro lado, el hecho de que estos algoritmos necesiten información completa del ambiente por donde se desplazará el robot móvil, implica muchas limitaciones de aplicación como el no poder usarlos en aplicaciones de exploración de lugares de cartografiados además del no tener una buena capacidad de reacción ante obstáculos imprevisto que no se encontraban detallados en el mapa.

Adicional se encuentra el hecho de que los algoritmos de planificación global necesitan un tiempo previo de planificación de ruta que, en ocasiones, puede llegar a ser excesivo y poco eficiente. Individualmente, se puede concluir diversos puntos acerca de cada algoritmo de este tipo.

El algoritmo RRT genera rutas efectivas hacia el objetivo final que se le plantea. Sin embargo, estas rutas generadas son bastante erráticas e innecesariamente largas, esto debido a la naturaleza aleatoria del algoritmo. Adicionalmente, es imposible determinar el tiempo que le tomará al algoritmo encontrar una ruta solución, siendo que podría demorar desde un par de minutos hasta más de media hora dependiendo de la extensión del mapa donde se desplazará el robot móvil. Por ende, se puede concluir que el algoritmo RRT, aunque efectivo, es errático y poco eficiente.

El algoritmo RRT* genera rutas efectivas, las cuales son más cortas y estables que las generadas por su predecesor RRT, esto gracias al aumento de una

estrategia de búsqueda de camino más corto que se añade en este nuevo algoritmo. Sin embargo, el tiempo requerido por el algoritmo RRT* para encontrar una ruta solución llega a ser igual o más largo que el tiempo empleado previamente por el algoritmo RRT, lo cual resulta bastante lógico tomando considerando el hecho de que se realiza un paso extra cada vez que se añade un nuevo nodo a la trama. Por consiguiente se puede concluir que, a pesar de que el algoritmo RRT* es efectivo y genera rutas cortas y estables, este es impredecible y poco eficiente con respecto al tiempo de planificación de ruta.

El algoritmo PFPP genera rutas estables, cortas y lineales, las cuales son generadas en un periodo de tiempo predecible que dependerá de la capacidad del procesador empleado para la ejecución del algoritmo. El punto débil de este algoritmo radica en la posibilidad de que las rutas generadas desemboquen en una situación de “mínimo local”, situación en la cual el robot móvil quedaría atrapado en un punto específico del mapa sin opción a salida. Sin embargo, cabría la posibilidad de agregar una estrategia auxiliar de escape para el algoritmo, la cual se active en el momento en el que se detectase una situación de “mínimo local”. En conclusión, el algoritmo PFPP puede ser considerado como estable, eficiente y también efectivo si es que se añadiese al mismo una estrategia auxiliar de escape para situaciones de “mínimo local”.

Los algoritmos de planificación local, conformados por el VFF y VFH, cuentan con una gran ventaja, la cual es el planificar la ruta deseada a seguir por el robot móvil en tiempo real basándose en información de los alrededores del robot móvil, información que es recogida por sensores de distancia. Esta cualidad le da un potencial muy grande a este tipo de algoritmos, ya que pueden ser utilizados para guiar robots móviles de exploración, los cuales pudieran ser empleados para cartografiar lugares desconocidos y/o peligrosos. Por otro lado, estos algoritmos, para su correcto funcionamiento, requieren de sensores altamente precisos y confiables que pudieran llegar a ser muy costosos.

Otra ventaja a destacar es el tiempo que necesitan estos algoritmos para conseguir llegar desde un punto A hasta un punto B, el cual se reduce mucho respecto a los algoritmos de planificación global por el hecho de no necesitar un tiempo de planificación de ruta previo. Adicional, los algoritmos de planificación local demuestran tener una capacidad de reacción muy buena ante obstáculos emergentes. Individualmente, se puede concluir diversos puntos acerca de cada algoritmo de este tipo.

El algoritmo VFF genera rutas efectivas, cortas y estables, las cuales guían al robot móvil hacia su objetivo en un tiempo relativamente corto. Sin embargo, la reacción de este algoritmo ante obstáculos frontales cercanos es ligeramente agresiva y podría ocasionar colisiones en el caso de tener dos

obstáculos muy cercanos el uno al otro. No obstante, el algoritmo VFF puede ser considerado como efectivo, eficiente y eficiente.

El algoritmo VFH genera rutas efectivas que son incluso más cortas y estables que las de su predecesor VFF, las cuales guían al robot móvil hacia su objetivo en un tiempo igual o menor que el tiempo empleado por el algoritmo VFF. Otra mejora que se puede mencionar es que el algoritmo VFH muestra una reacción controlada y estable frente a obstáculos frontales inminentes, rodeándolos con sutileza, lo cual le da una robustez adicional y lo hace el perfecto candidato para guiar a un robot móvil a través de ambientes con espacios muy estrechos. Por consiguiente, el algoritmo VFH puede considerarse como efectivo, eficiente, estable y preciso.

Existen 2 puntos importantes que pueden deducirse a partir de los comportamientos observados de cada algoritmo.

El primero punto es el hecho de que los algoritmos de planificación local tienen un desempeño superior a los algoritmos de planificación global, por lo que son los más aptos para aplicaciones en la realidad.

El segundo punto a mencionar es que una combinación correcta de un algoritmo de planificación global junto con un algoritmo de planificación local podría resultar en un sistema de navegación cuasi perfecto, el cual pudiera superar cualquier tipo de obstáculo y guiar a un robot móvil de manera eficiente y eficaz a través de cualquier tipo de ambiente sin importar que tan complicado sea.

## 4.2. RECOMENDACIONES

Los códigos fuente usados para implementar los algoritmos de este proyecto podrían ser depurados con el objetivo de disminuir la extensión de los mismos y por ende el tiempo necesario para su ejecución efectiva. Esta depuración incluso podría devenir en un ahorro económico, esto debido a que al depurar los códigos fuente se podría disminuir la capacidad de procesamiento necesaria para ejecutarlos y por ende se podría utilizar un procesador menos potente y costoso.

Este proyecto tiene el potencial de ser llevado a una aplicación en la realidad. En cuyo caso es necesario entender que para los algoritmos de planificación local existe un coeficiente de certeza de ocupación, el cual indica que tan confiable es la lectura dada por el sensor. Debido a que este proyecto se realizó mediante simulaciones, dicho valor fue tomado como 1 (100% de confiabilidad), pero para un proyecto en la realidad, este valor debería ser diferente y corresponder con la confiabilidad del sensor que se utilice.

En caso de llevarse este proyecto a la realidad, es recomendable dividir las actividades en dos zonas distintas mas no colocar el procesador, actuadores y sensores en el mismo robot móvil. Lo más efectivo sería realizar las actividades de procesamiento de información en un ordenador para luego transmitir las órdenes calculadas a los actuadores del robot móvil mediante una conexión Bluetooth o WiFi. De esta forma se aligeraría el peso del robot móvil, se tendría una velocidad de procesamiento mayor e incluso se podría monitorear, en tiempo real, la posición y estado del robot móvil durante todo el trayecto.

Para la simulación de este proyecto se utilizó la librería de uso gratuito de MatLab llamada Mobile Robotics Simulation Toolbox, la cual facilitó la implementación y simulación de los algoritmos. En caso de querer analizar otros tipos de algoritmos de evasión de obstáculos o hacer un análisis mucho más especializado de alguno de ellos, se recomienda usar la misma librería mas no intentar crear un nuevo sistema de simulación. Esto debido a que el propio sistema de simulación es bastante complejo y complicado de replicar. No obstante, cabe decir que el crear un nuevo y original sistema de navegación podría ser considerado como un proyecto en sí mismo el cual valdría mucho la pena.

En el caso de este proyecto, se usaron mapas con una resolución de 0.1 [m], es decir, cada cuadrícula del mapa tenia 10 cm por lado. Si se quisiera, podrían hacerse mapas de mayor resolución, los cuales permitirían definir obstáculos con formas más complejas. Sin embargo, cabe recalcar que la programación de dichos mapas sería más complicada y, en el caso del

algoritmo PFPP, el tiempo de planificación de ruta se elevaría exponencialmente.

En cuanto al modelo del robot móvil, se recomienda que sin importar la configuración de robot móvil a usarse, siempre se aplique la cinemática inversa debido a su sencillez y facilidad de uso.

# BIBLIOGRAFÍA

Zhao, J., Gao, J., Zhao, F., & Liu, Y. (2017). A Search-and-Rescue Robot System for Remotely Sensing the Underground Coal Mine Environment. Sensors 2017, 17(10), 2426; https://doi.org/10.3390/s17102426

Valverde, S. (2020). Implementación de sensores 3D para la navegación de robots móviles y vehículos autónomos. Tecnología en Marcha. Vol. 33, especial Movilidad estudiantil. Pág. 176-186. https://doi.org/10.18845/tm.v33i7.54921

Tamre, M., Hudjakov, R., Shvarts, D., Polder, A., Hiiemaa, M., & Juurma, M. (2019). Implementation of Integrated Wireless Network and MATLAB System to Control Autonomous Mobile Robot: Implementation of Integrated Wireless Network and MATLAB System to Control Autonomous Mobile Robot. International Journal of Innovative Technology and Interdisciplinary Sciences, 1(1), 18–25. https://doi.org/10.15157/IJITIS.2018.1.1.18-25

Velázquez, R., Orvañaños, M., & Macías, R. (2020). Diseño y Prototipado de un robot móvil autónomo y tele-operado. Pistas Educativas, 42(136),1029-1045.

Rodríguez, M. (2019). Navegación autónoma de un robot utilizando una guía visual. (Tesis de maestría, Centro de Investigación y Desarrollo de Tecnología Digital). http://mexculture.citedi.mx/handle/123456789/1418

Pérez, A. (2019). Estudio comparativo entre sensores de distancia, con diferente principio de medición para determinar las especificaciones en la implementación de un scanner tridimensional. (Tesis de pregrado). https://dspace.ups.edu.ec/bitstream/123456789/16981/1/UPS-ST003984.pdf

PCE-instruments. (2018). Telémetro láser. Recuperado de: https://www.pce-instruments.com/espanol/instrumento-medida/medidor/telemetro-laser-kat_70337.htm

Araneda, A. (2020). Simulaición y Análisis de algoritmos de planificación y rutas para robot móvil en Gazebo. (Tesis de pregrado). http://repositorio.unab.cl/xmlui/bitstream/handle/ria/17975/a131453_Araneda_A_Simulacion_y_Analisis_de_algoritmos_2020_tesis.pdf?sequence=1&isAllowed=y

Alvarez, G., & Flor, O. (2020). Desempeño en métodos de navegación autónoma para robots móviles. Minerva, 1(2), 19-29. https://doi.org/10.47460/minerva.v1i2.8

Hernandez, D., Eguibar, J., Cortés, C. & Reyes, J. (2017). Diseño, construcción y modelo dinámico de un robot móvil de tracción diferencial aplicado al seguimiento de trayectorias. Memorias del xxiii Congreso Internacional anual de la SOMIM. http://somim.org.mx/memorias/memorias2017/articulos/A3_189.pdf

Tiep, D., Lee, K., Im, D., Kwak, B. & Ryoo. (2018). Design of Fuzzy-PID controller for Path Tracking of Mobile Robot with Differential Drive. International Journal of Fuzzy Logic and Intelligent System, 18(3), 220-228. http://doi.org/10.5391/IJFIS.2018.18.3.220

Wei, K. & Ren, B. (2018). A Method on Dynamic Path Planning for Robotic Manipulator Autonomous Obstacle Avoidance Based on an Improved RRT Algorithm. Sensors 2018, 18(2), 571. https://doi.org/10.3390/s18020571

Zhang, H., Wang, Y., Zheng, J., & Yu, J. (2018). Path Planning of Industrial Robot Based on Improved RRT Algorithm in Complex Environments. IEEE Access, 1–1. doi:10.1109/access.2018.2871222

Adiyatov, O. & Varol, H. (2017). A Novel RRT*-Based Algorithm for Motion Planning in Dynamic. IEEE International Conference on Mechatronics and Automation (ICMA, pp. 1416-1421, doi: 10.1109/ICMA.2017.8016024.

Orozco-Rosas, U., Montiel, O., & Sepúlveda, R. (2019). Mobile robot path planning using membrane evolutionary artificial potential field. Applied Soft Computing, 77, 236–251. doi:10.1016/j.asoc.2019.01.036

Bounini, F., Gingras, D., Pollart, H., & Gruyer, D. (2017). Modified artificial potential field method for online path planning applications. 2017 IEEE Intelligent Vehicles Symposium (IV). doi:10.1109/ivs.2017.7995717

Nia, D. N., Tang, H. S., Karasfi, B., Motlagh, O. R. E., & Kit, A. C. (2011). Virtual force field algorithm for a behaviour-based autonomous robot in unknown environments. Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering, 225(1), 51–62. doi:10.1243/09596518jsce958

Alagic, E., Velagic, J., & Osmanovic, A. (2019). Design of Mobile Robot Motion Framework based on Modified Vector Field Histogram. 2019 International Symposium ELMAR. doi:10.1109/elmar.2019.8918891

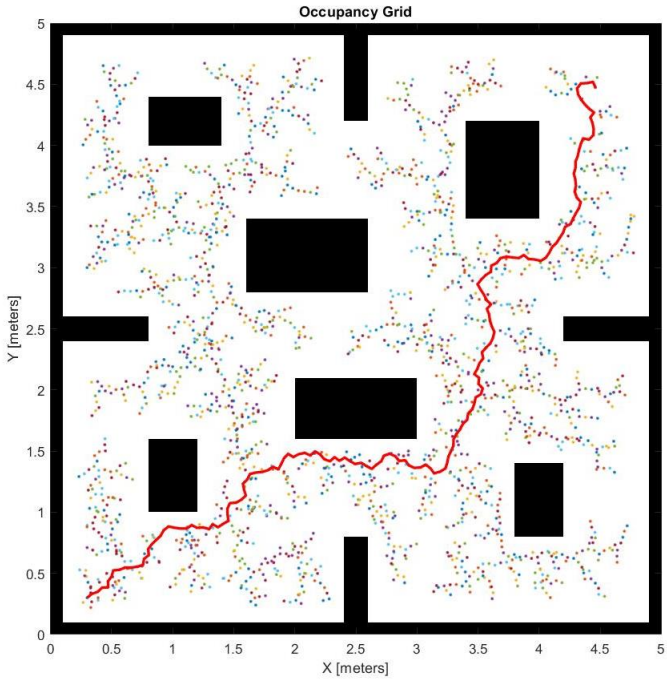
Li, J., Bao, H., Han, X., Pan, F., Pan, W., Zhang, F., & Wang, D. (2016). Real-time self-driving car navigation and obstacle avoidance using mobile 3D laser scanner and GNSS. Multimedia Tools and Applications, 76(21), 23017–23039. doi:10.1007/s11042-016-4211-7

Borenstein, J., & Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. IEEE Transactions on Robotics and Automation, 7(3), 278–288. doi:10.1109/70.88137

# ANEXOS

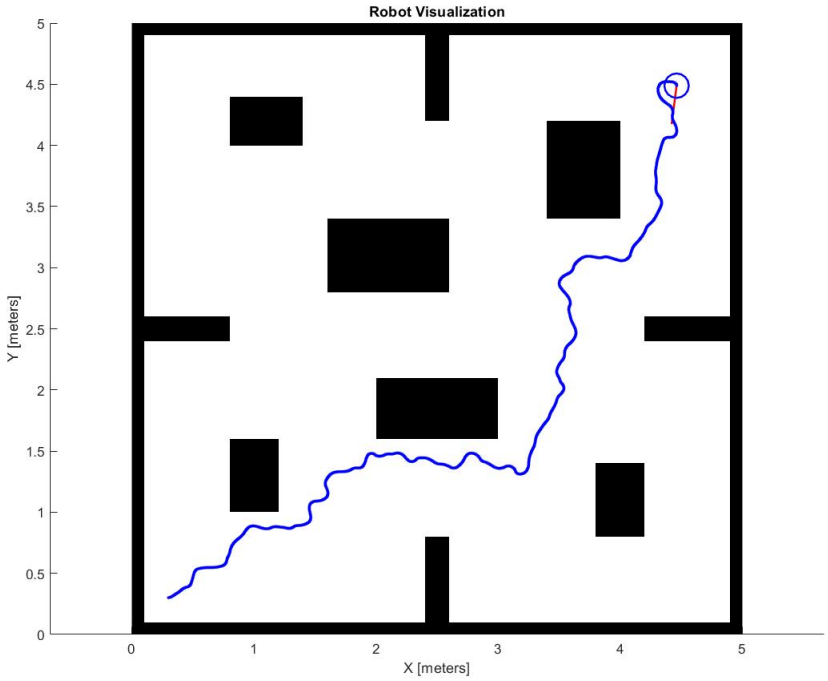
# ANEXO 1

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°1.



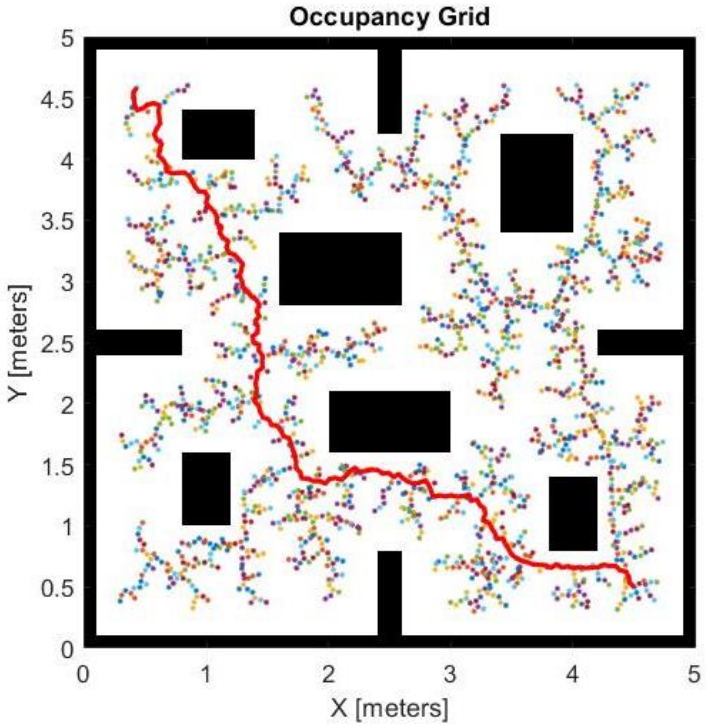

# ANEXO 2

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°2 GUIADO POR ALGORITMO RRT.



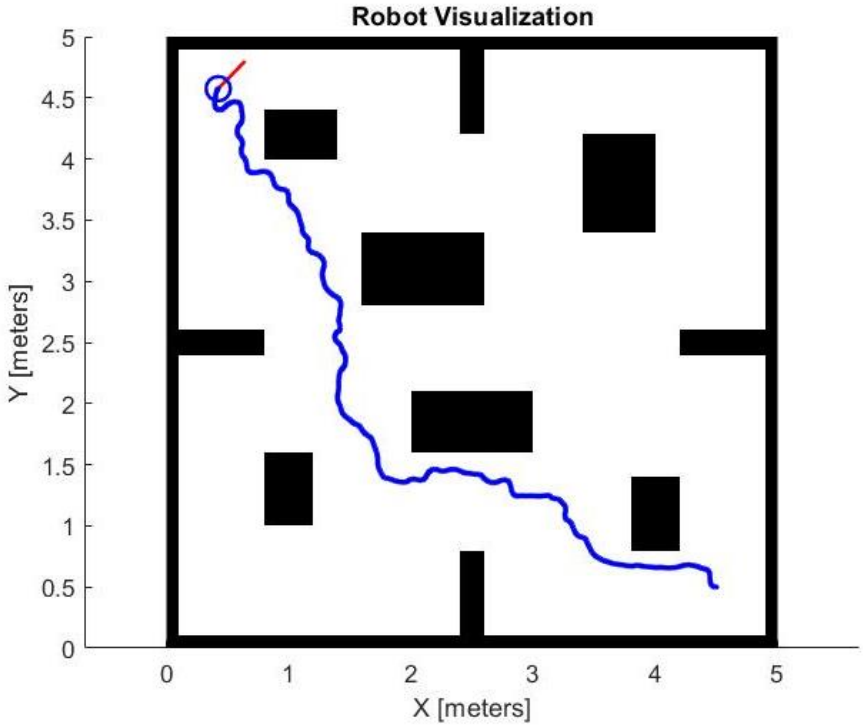
## ANEXO 3

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°2.



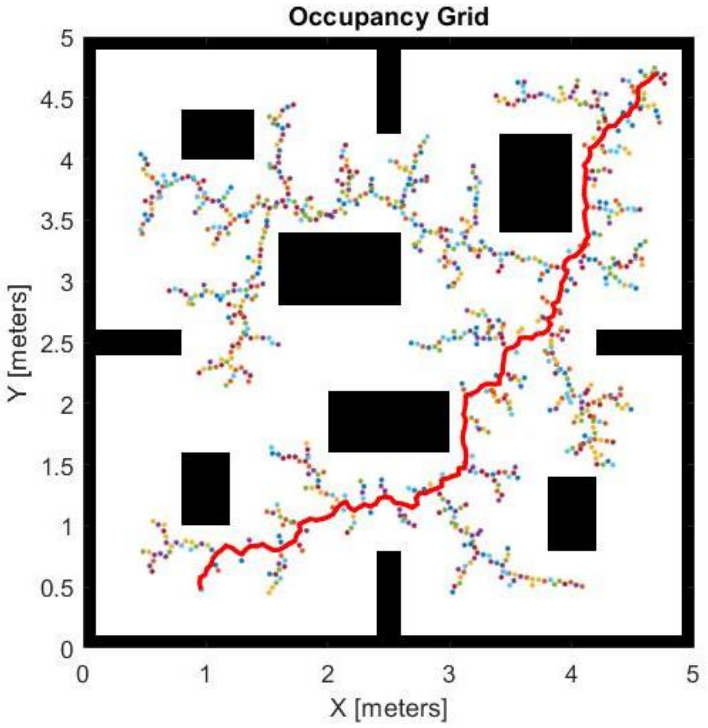

## ANEXO 4

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°2 GUIADO POR ALGORITMO RRT.



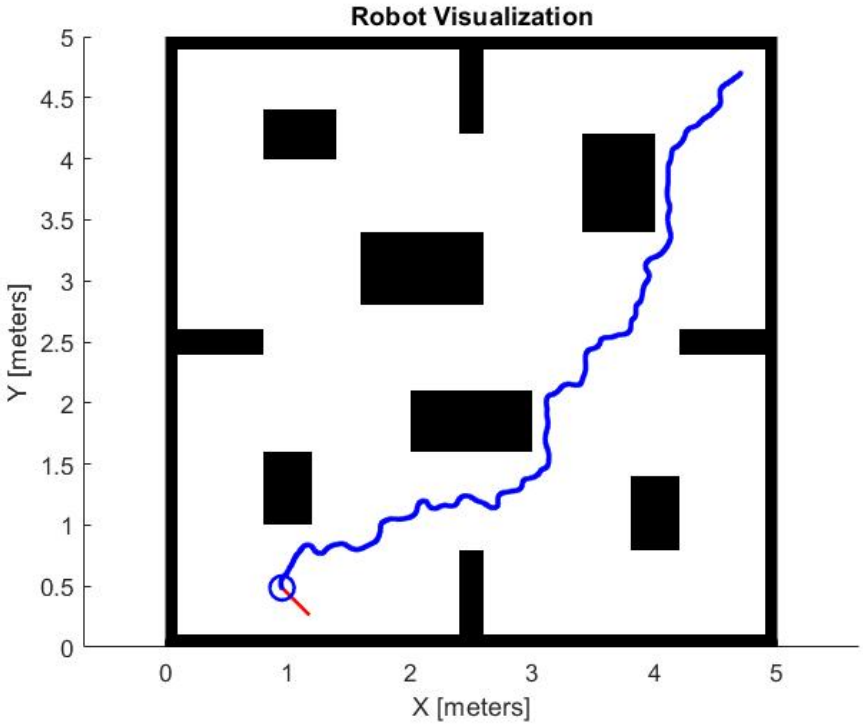
## ANEXO 5

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°3.



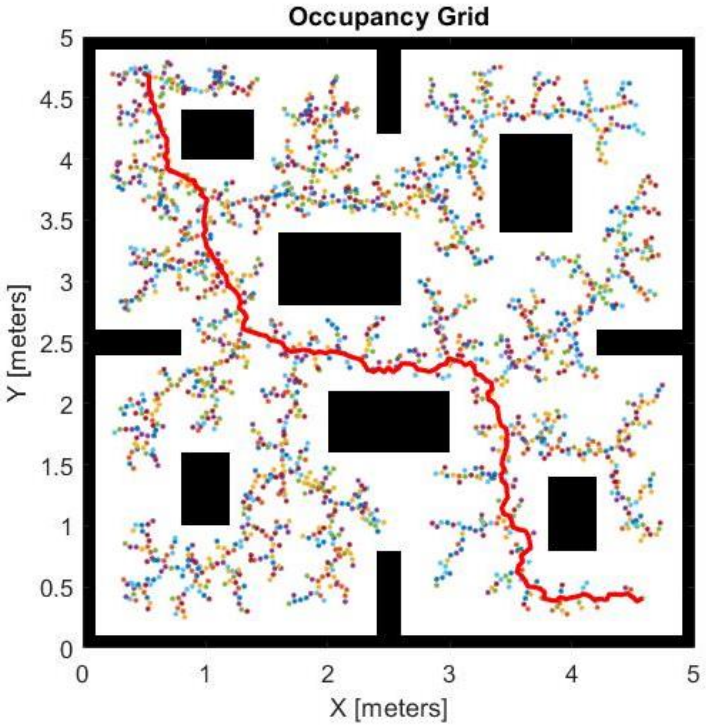

## ANEXO 6

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°3 GUIADO POR ALGORITMO RRT.



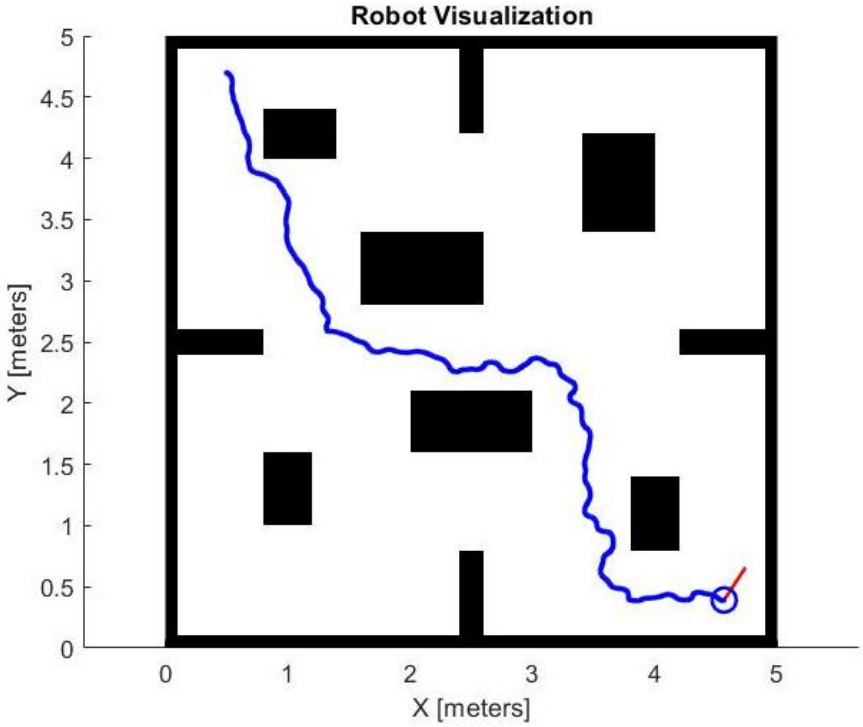
## ANEXO 7

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°4.



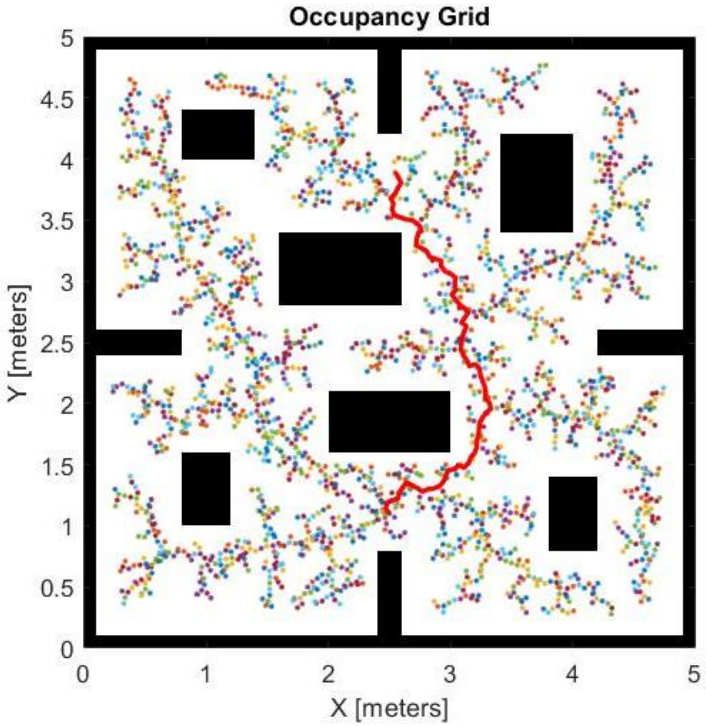

## ANEXO 8

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°4 GUIADO POR ALGORITMO RRT.



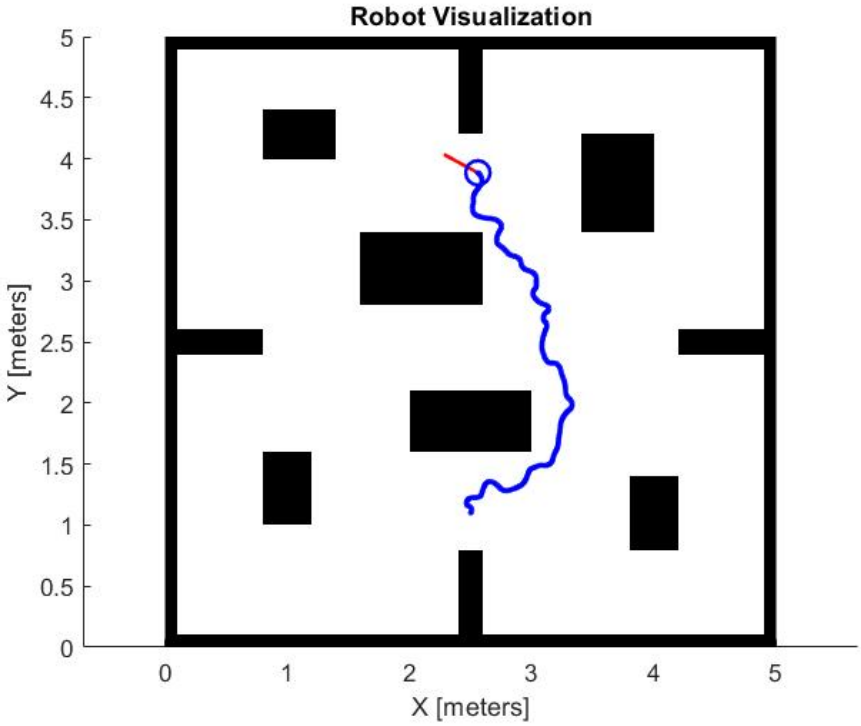
## ANEXO 9

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°5.



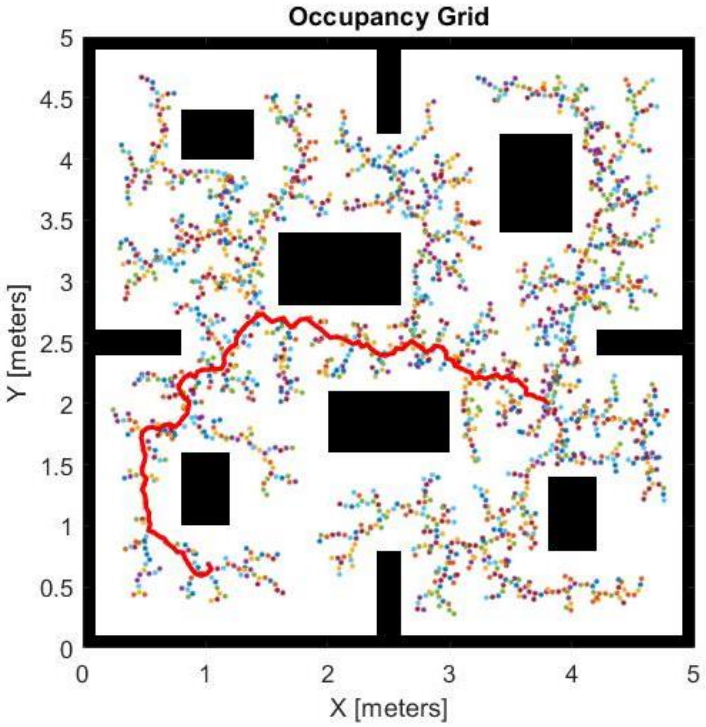

## ANEXO 10

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°5 GUIADO POR ALGORITMO RRT.



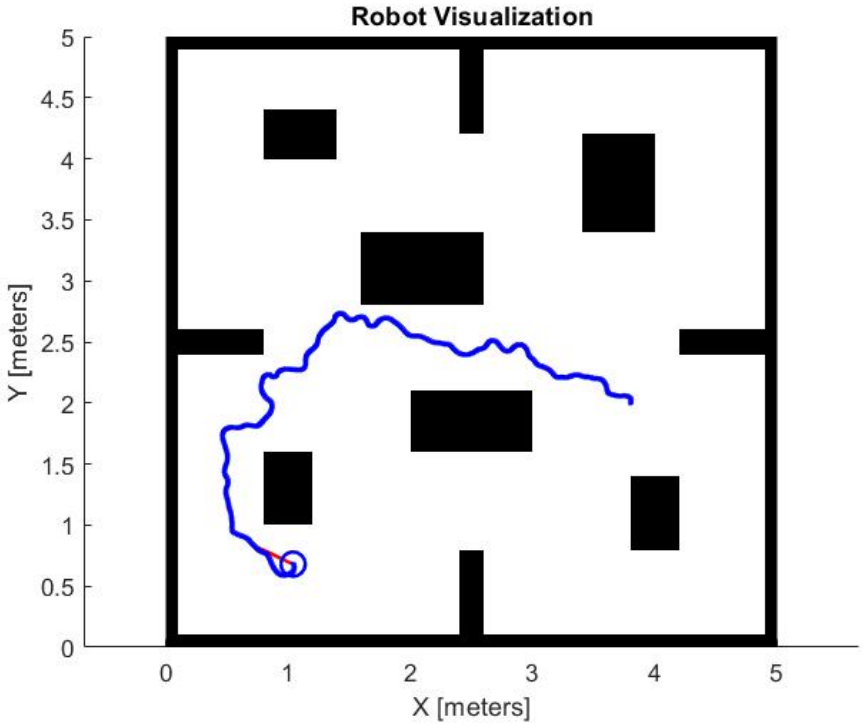
## ANEXO 11

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°6.



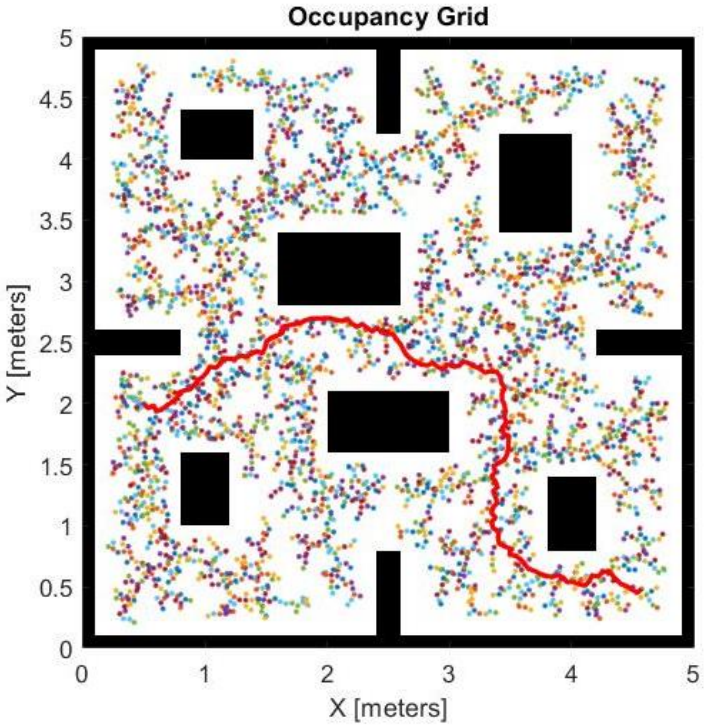

## ANEXO 12

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°6 GUIADO POR ALGORITMO RRT.



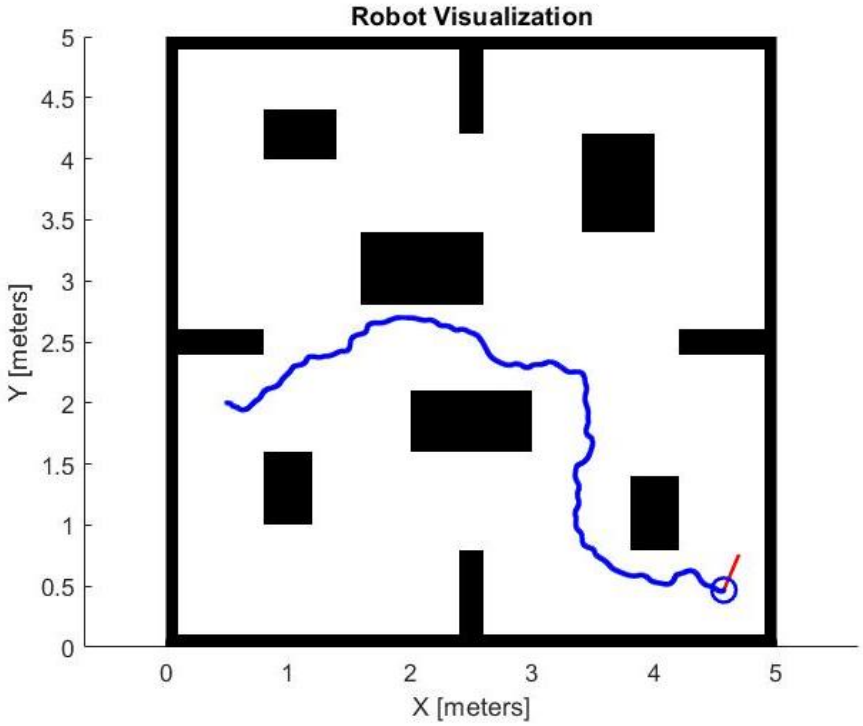
## ANEXO 13

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°7.



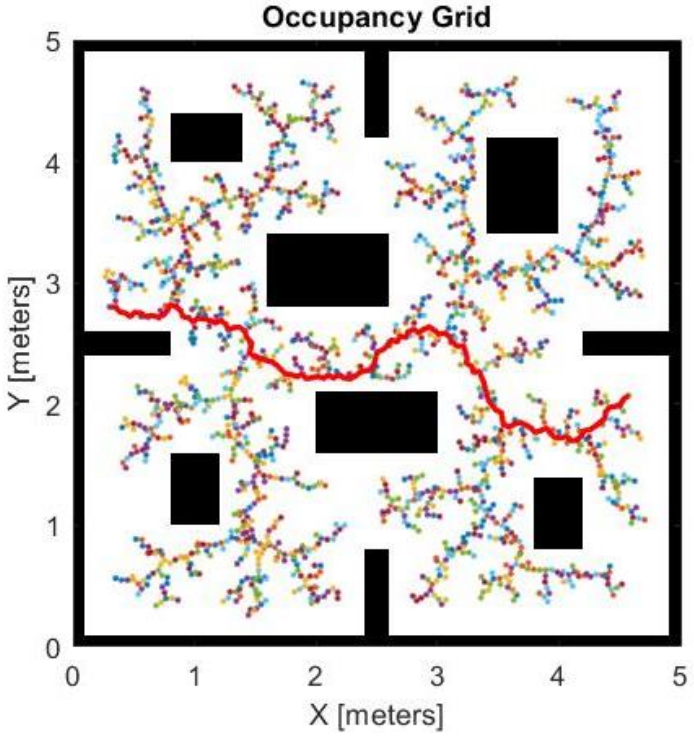

## ANEXO 14

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°7 GUIADO POR ALGORITMO RRT.



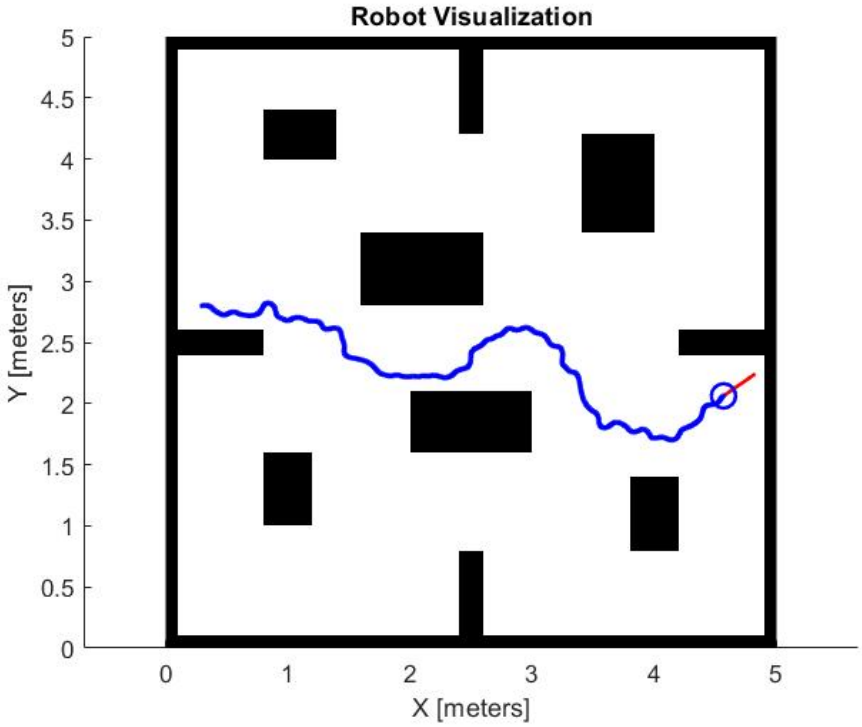
## ANEXO 15

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°8.



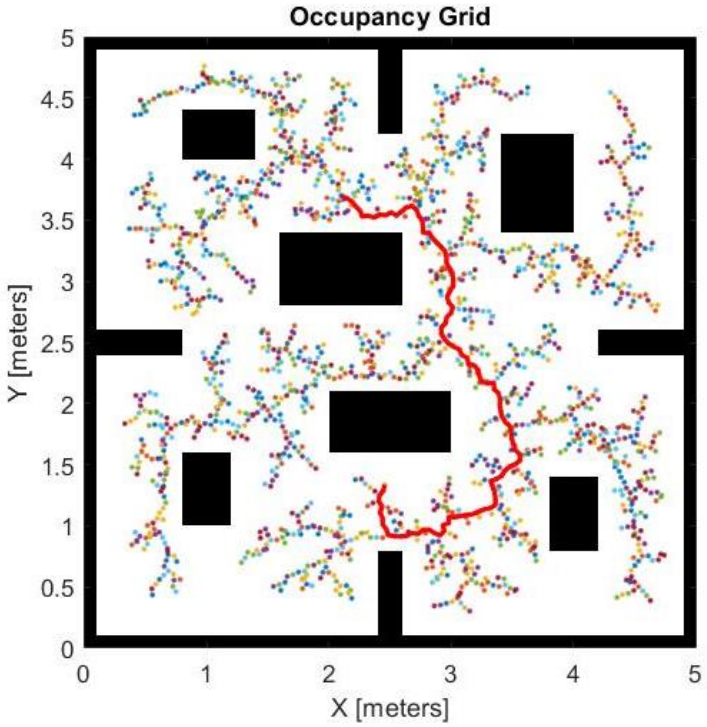

## ANEXO 16

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°8 GUIADO POR ALGORITMO RRT.



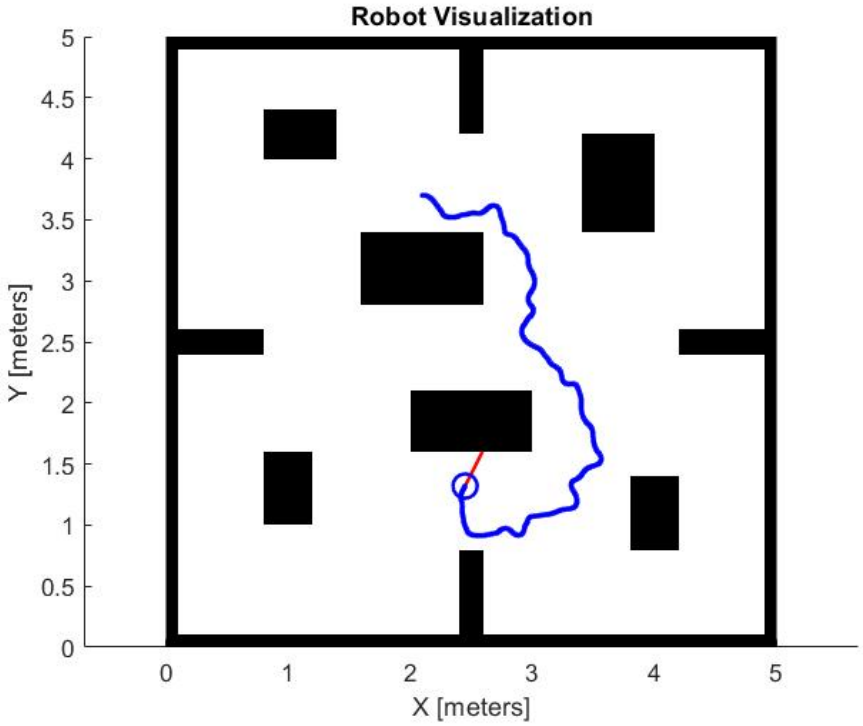
## ANEXO 17

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°9.



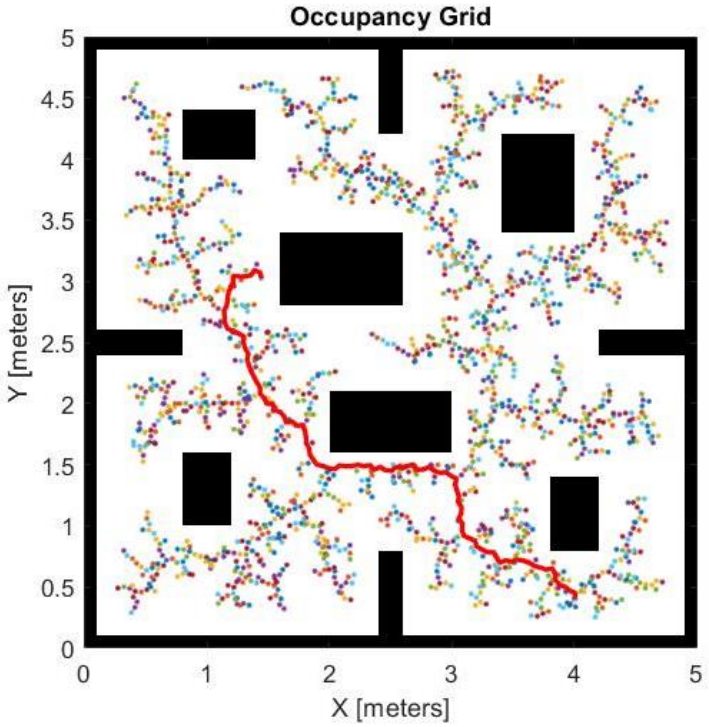

## ANEXO 18

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°9 GUIADO POR ALGORITMO RRT.



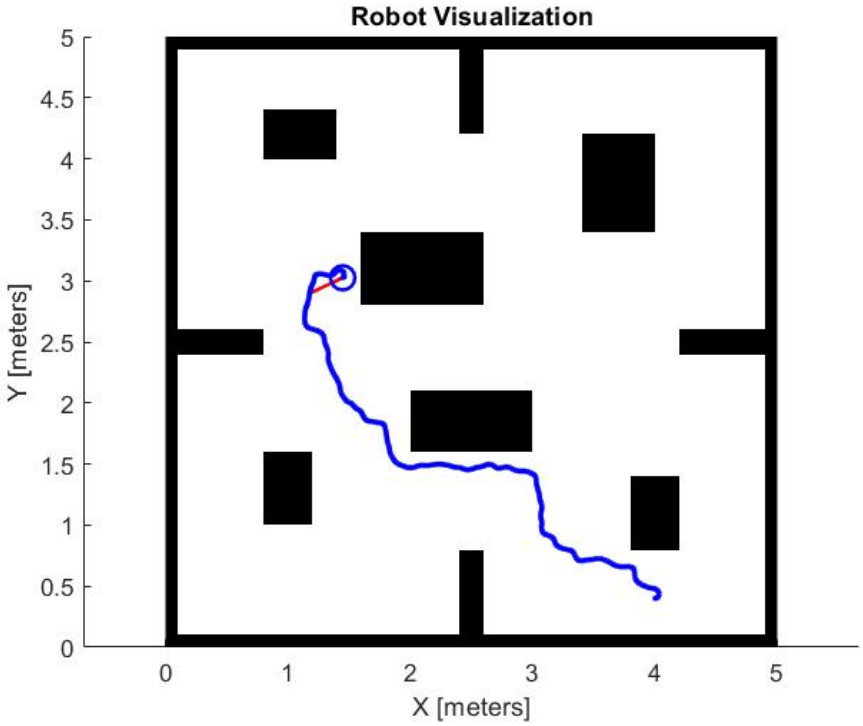
## ANEXO 19

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT PARA RUTA DE PRUEBA N°10.



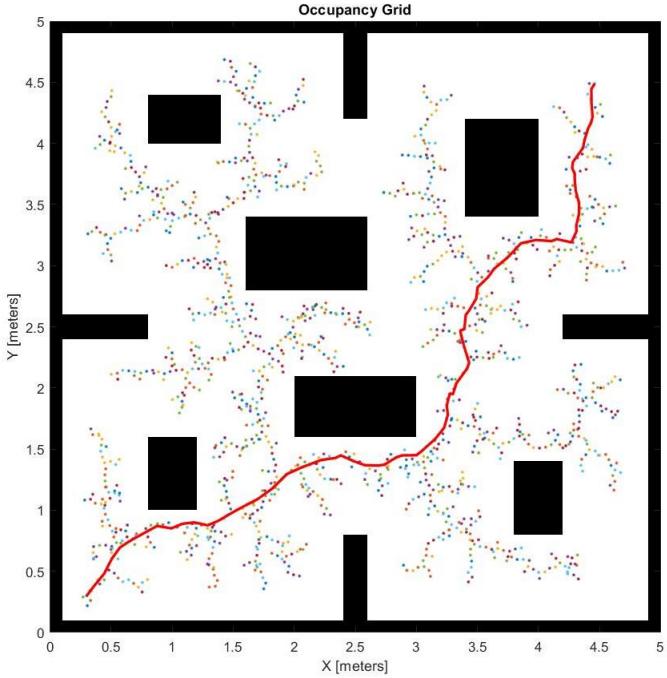

## ANEXO 20

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°10 GUIADO POR ALGORITMO RRT.



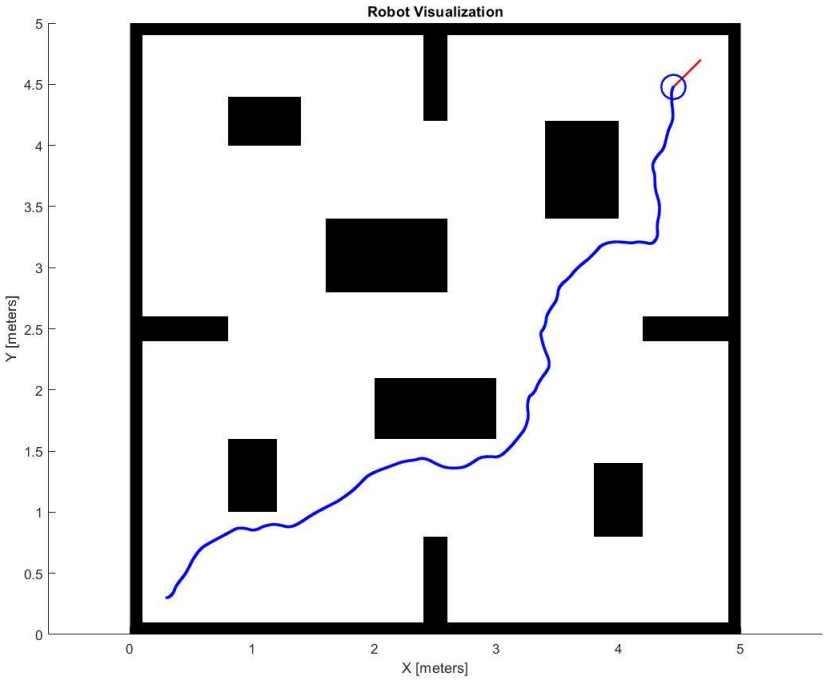
## ANEXO 21

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°1.



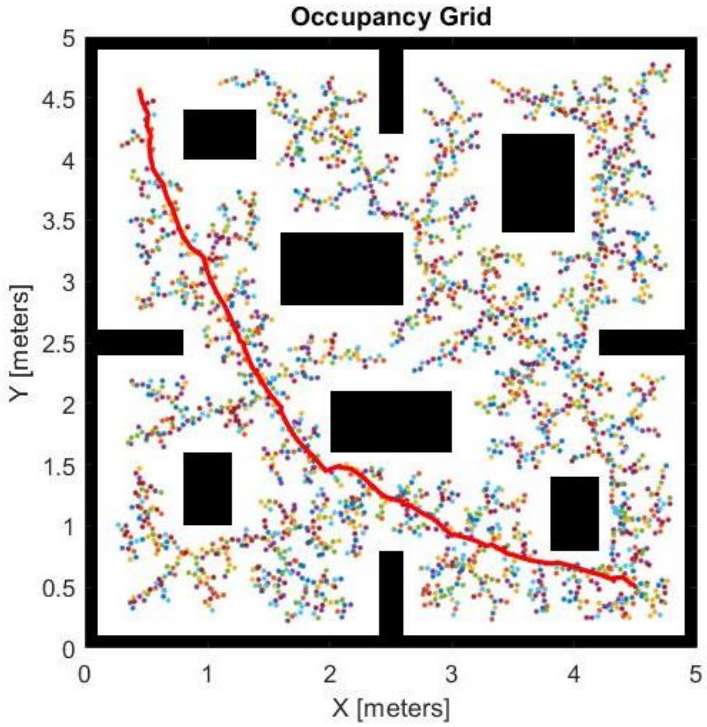

## ANEXO 22

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°1 GUIADO POR ALGORITMO RRT*.



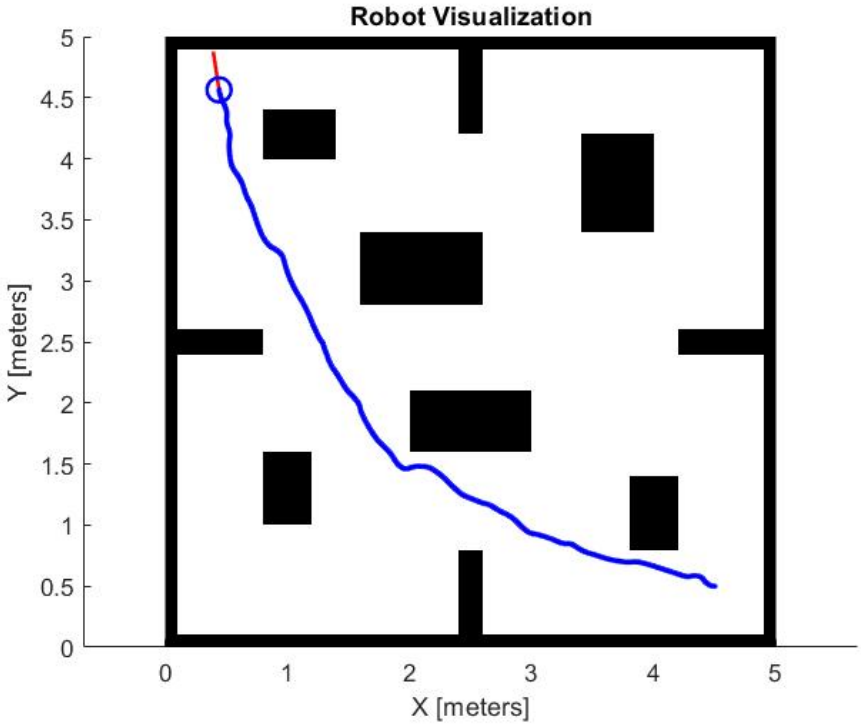
## ANEXO 23

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°2.



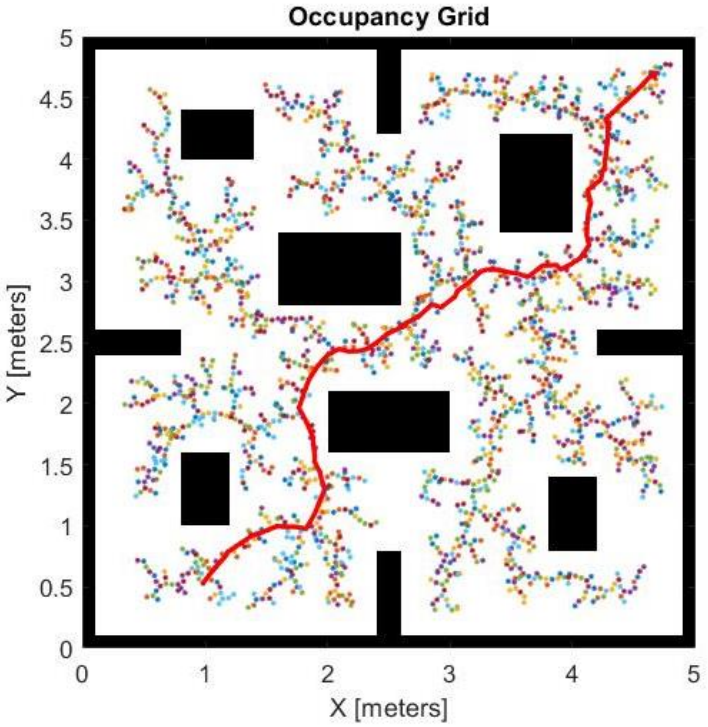

## ANEXO 24

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°2 GUIADO POR ALGORITMO RRT*.



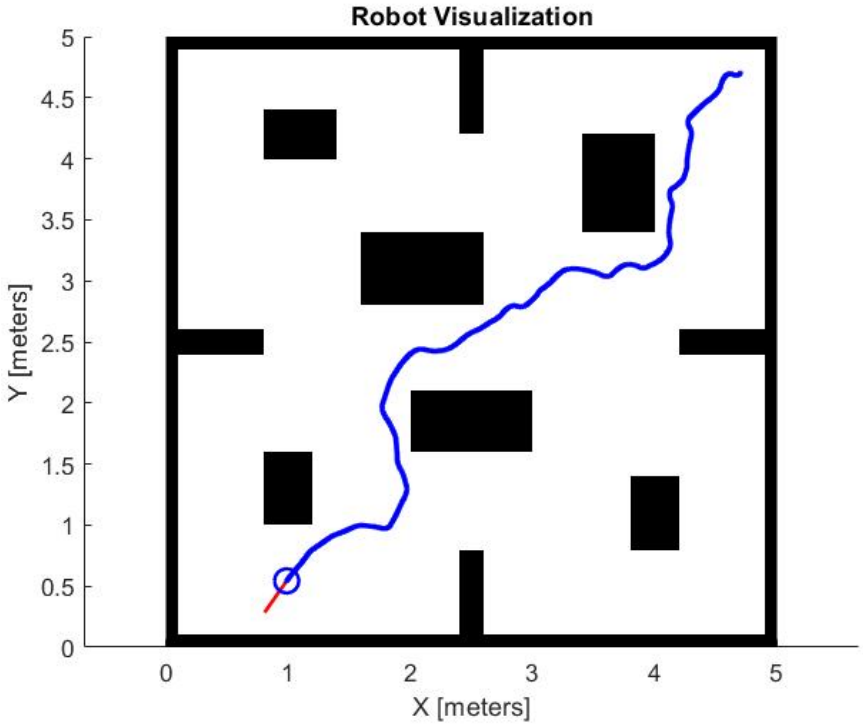
## ANEXO 25

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°3.



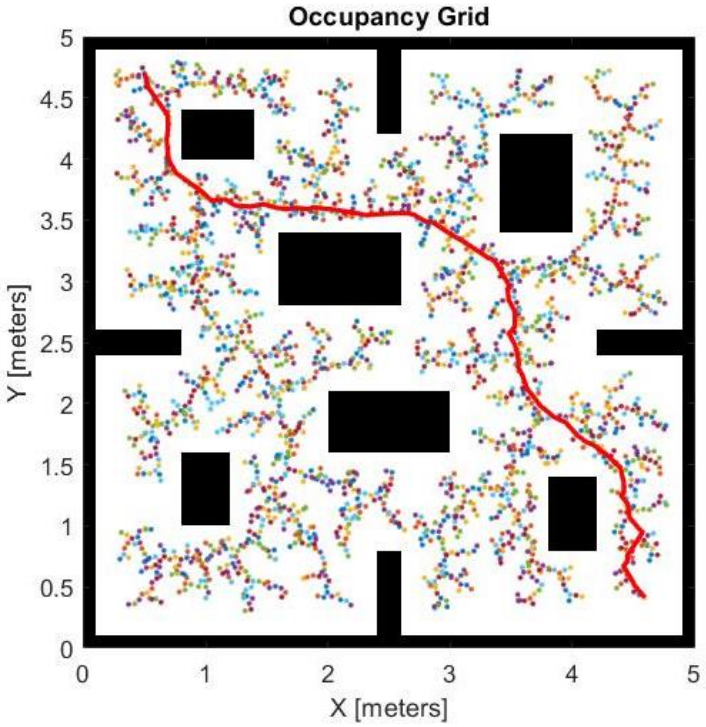

## ANEXO 26

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°3 GUIADO POR ALGORITMO RRT*.



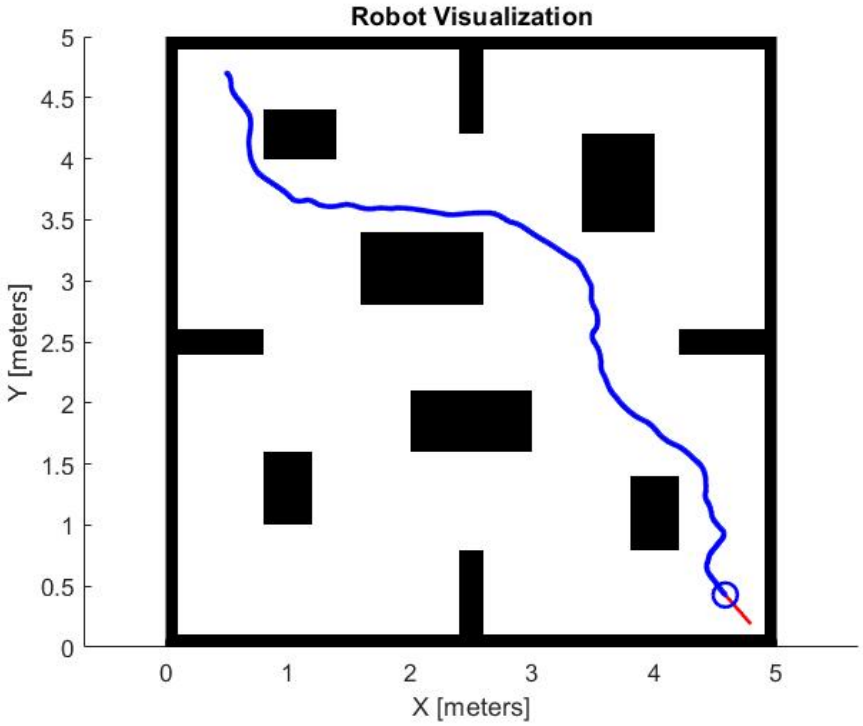
## ANEXO 27

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°4.



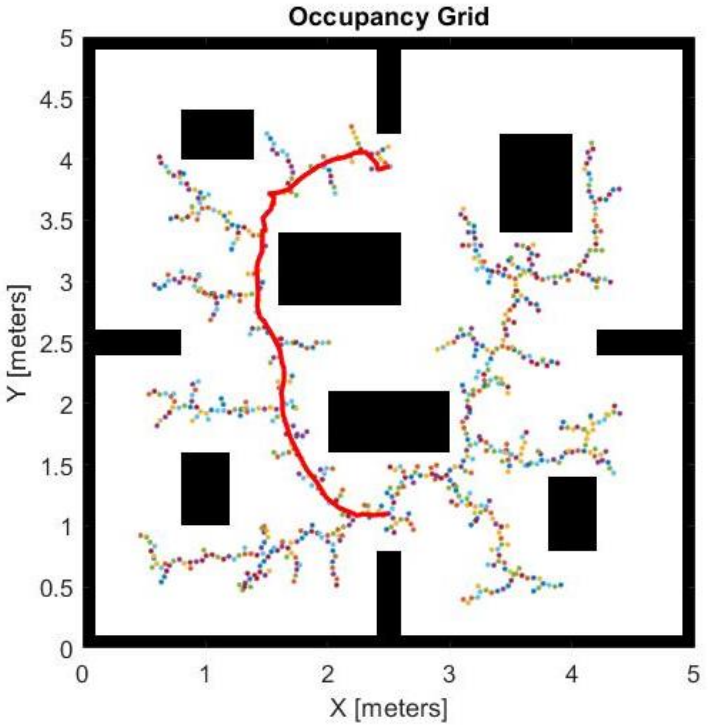

## ANEXO 28

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°4 GUIADO POR ALGORITMO RRT*.



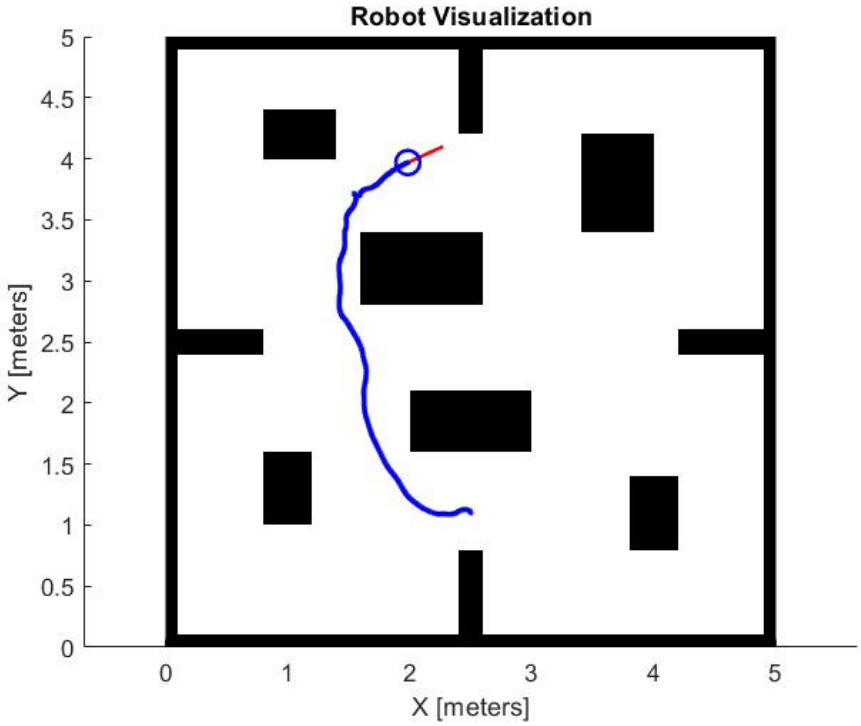
## ANEXO 29

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°5.



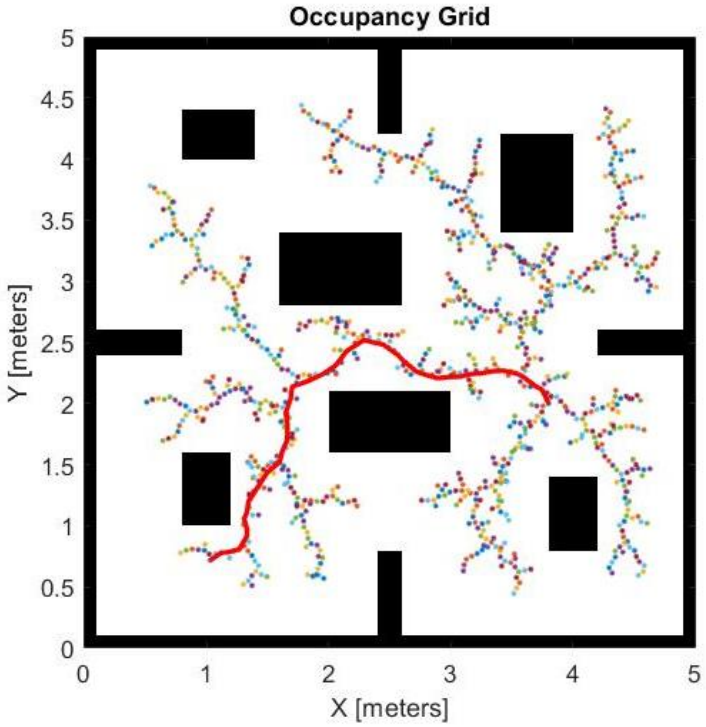

## ANEXO 30

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°5 GUIADO POR ALGORITMO RRT*.



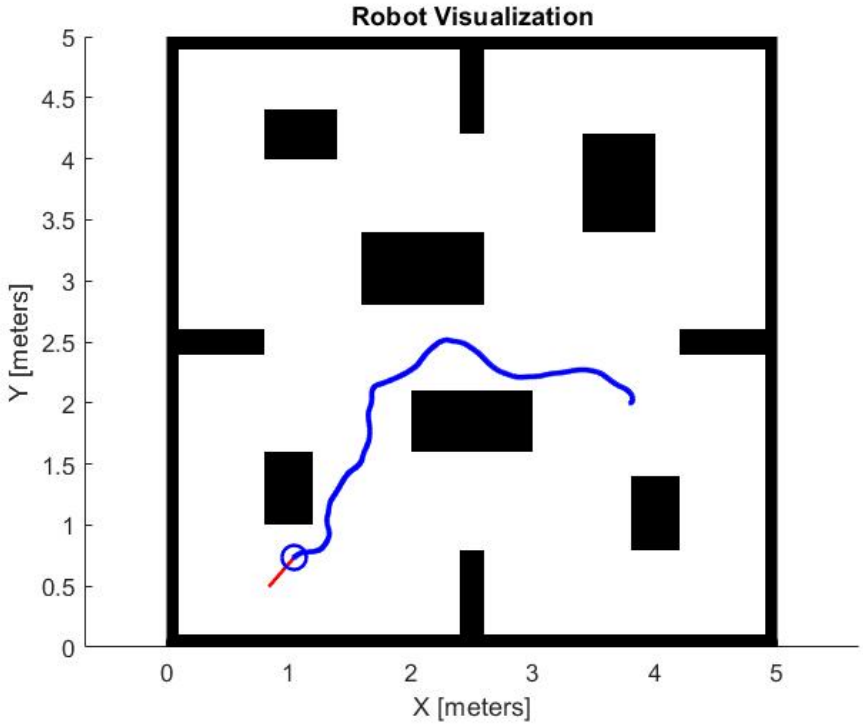
## ANEXO 31

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°6.



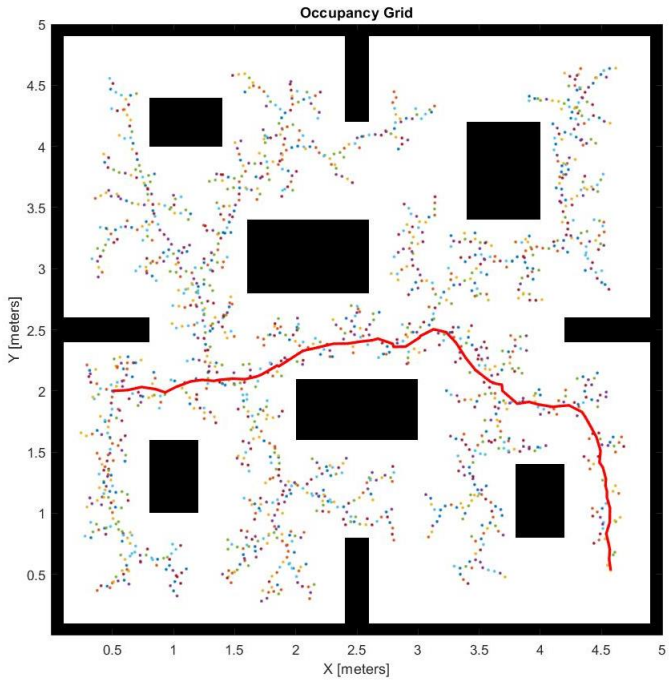

## ANEXO 32

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°6 GUIADO POR ALGORITMO RRT*.



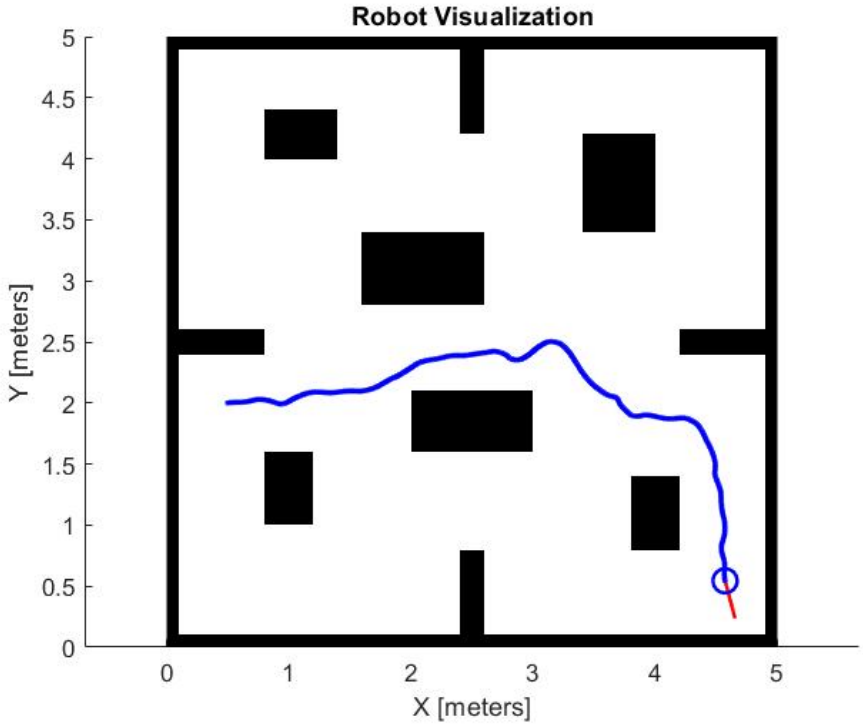
## ANEXO 33

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°7.



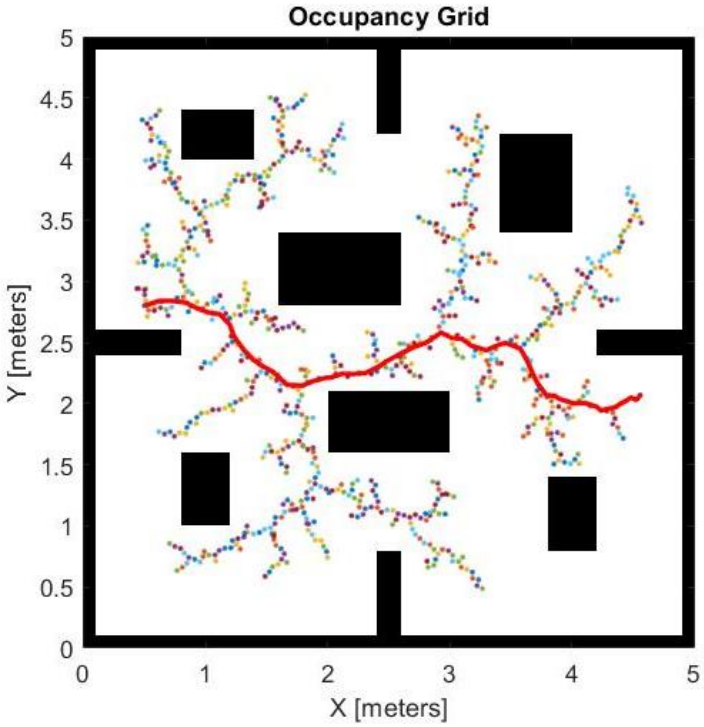

## ANEXO 34

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°7 GUIADO POR ALGORITMO RRT*.



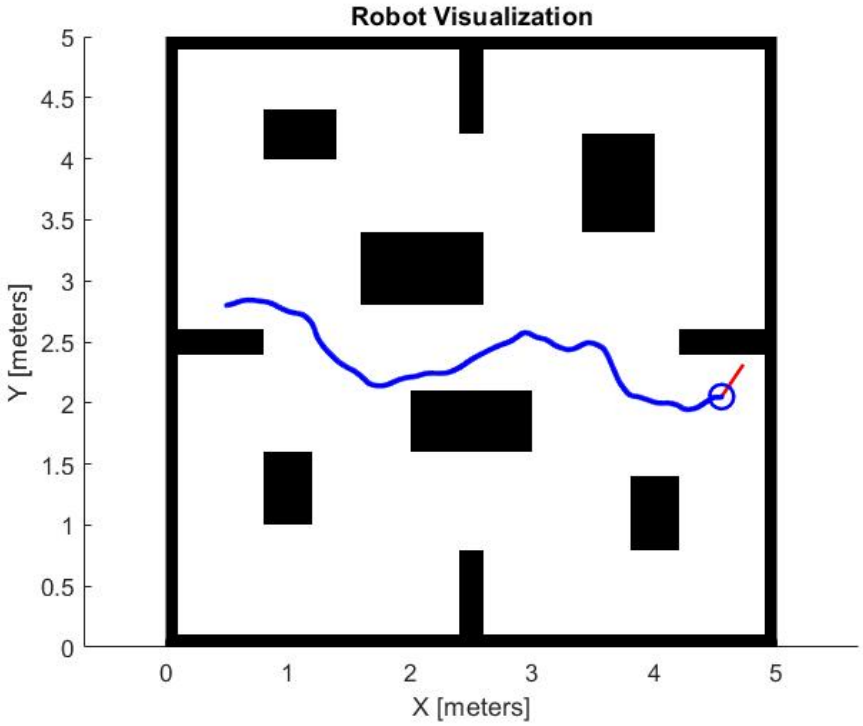
## ANEXO 35

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°8.



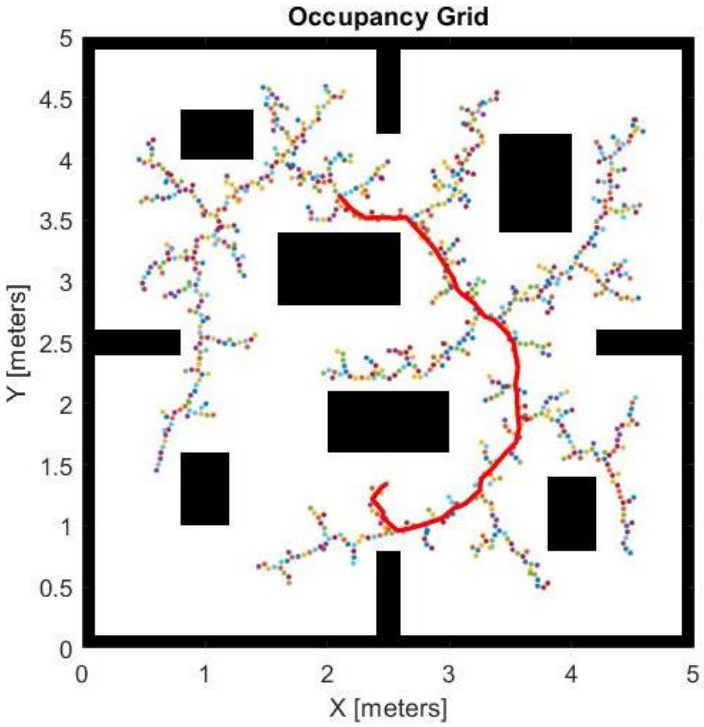

## ANEXO 36

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°8 GUIADO POR ALGORITMO RRT*.



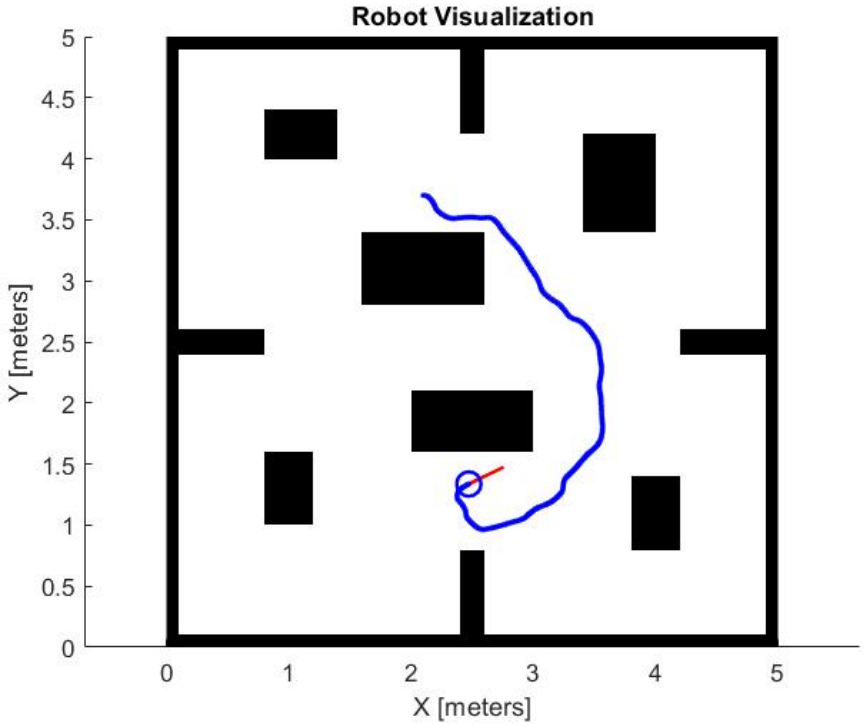
## ANEXO 37

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°9.



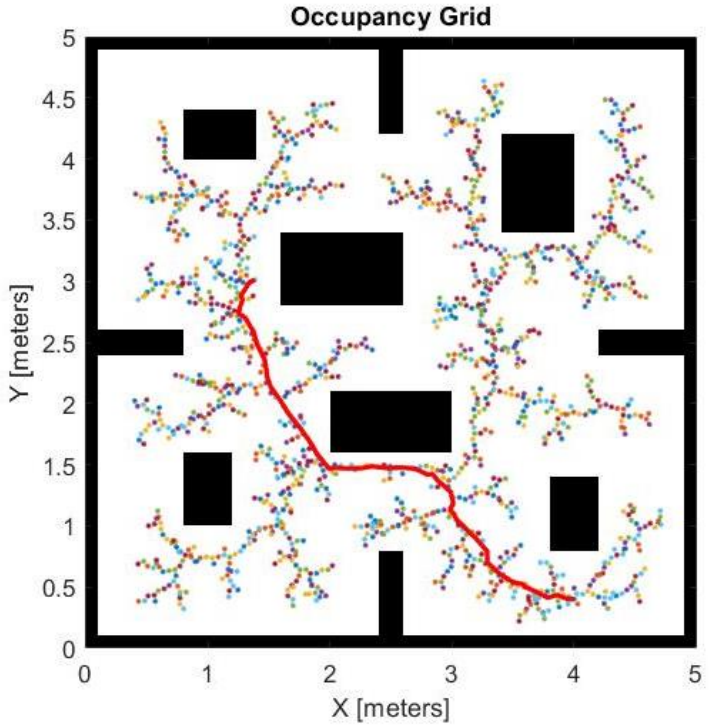

## ANEXO 38

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°9 GUIADO POR ALGORITMO RRT*.



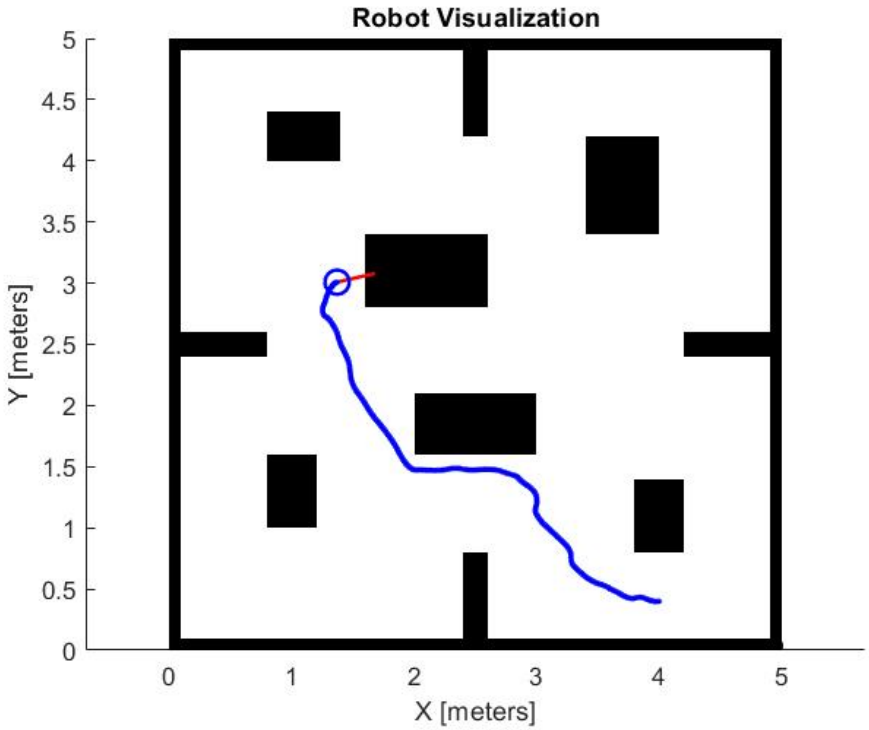
## ANEXO 39

## PLANIFICACIÓN DE RUTA GENERADA POR ALGORITMO RRT* PARA RUTA DE PRUEBA N°10.



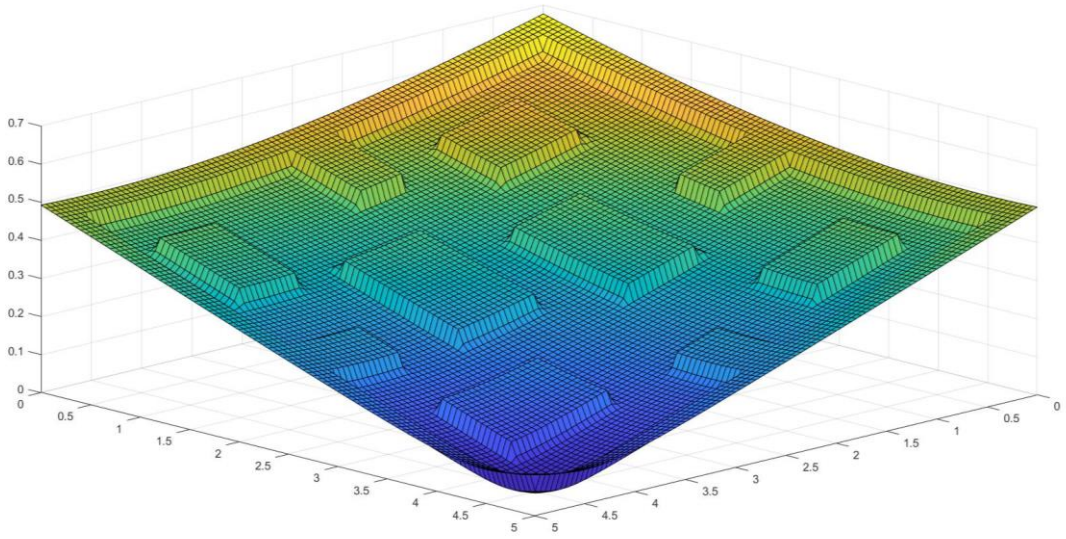

## ANEXO 40

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°10 GUIADO POR ALGORITMO RRT*.



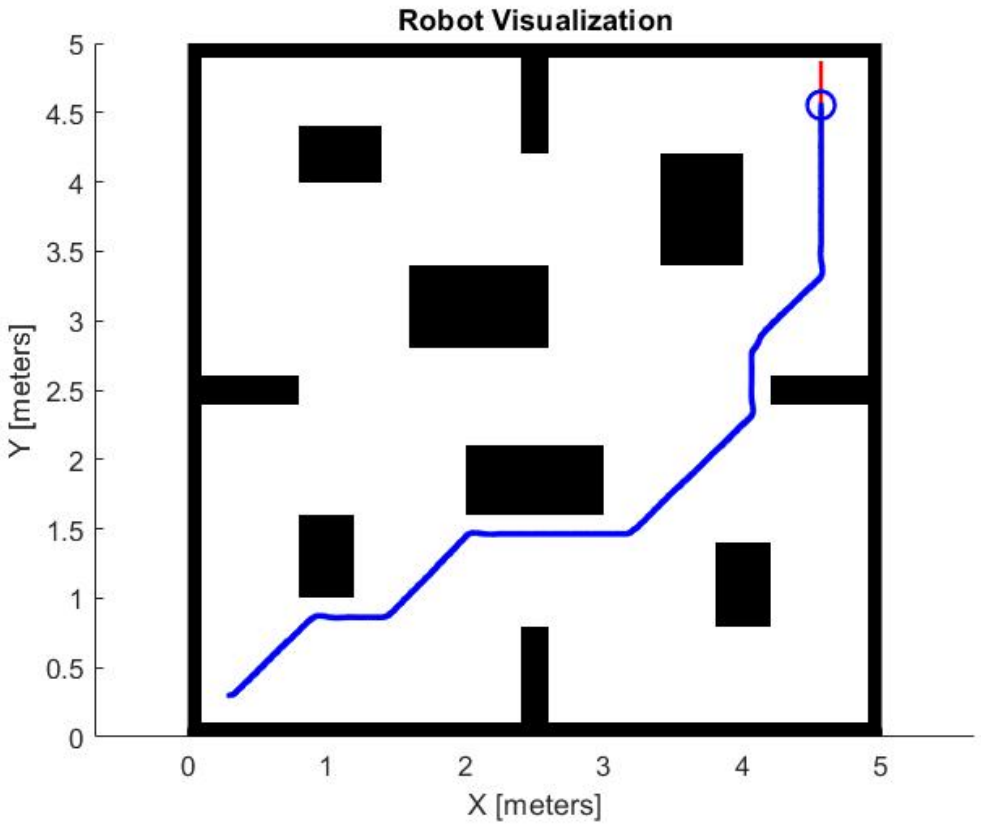
## ANEXO 41

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°1.



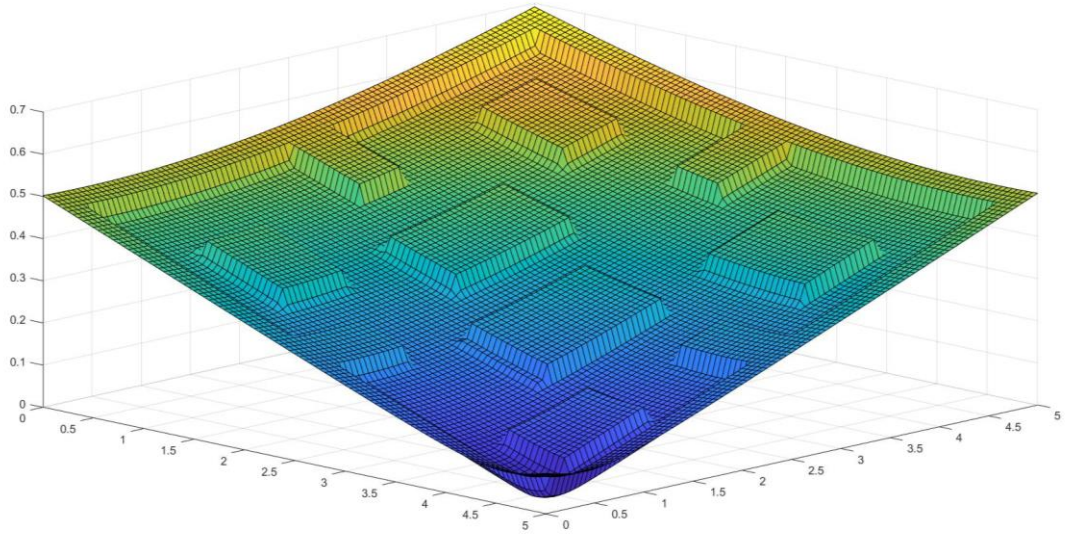
## ANEXO 42

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°1 GUIADO POR ALGORITMO PFPP.



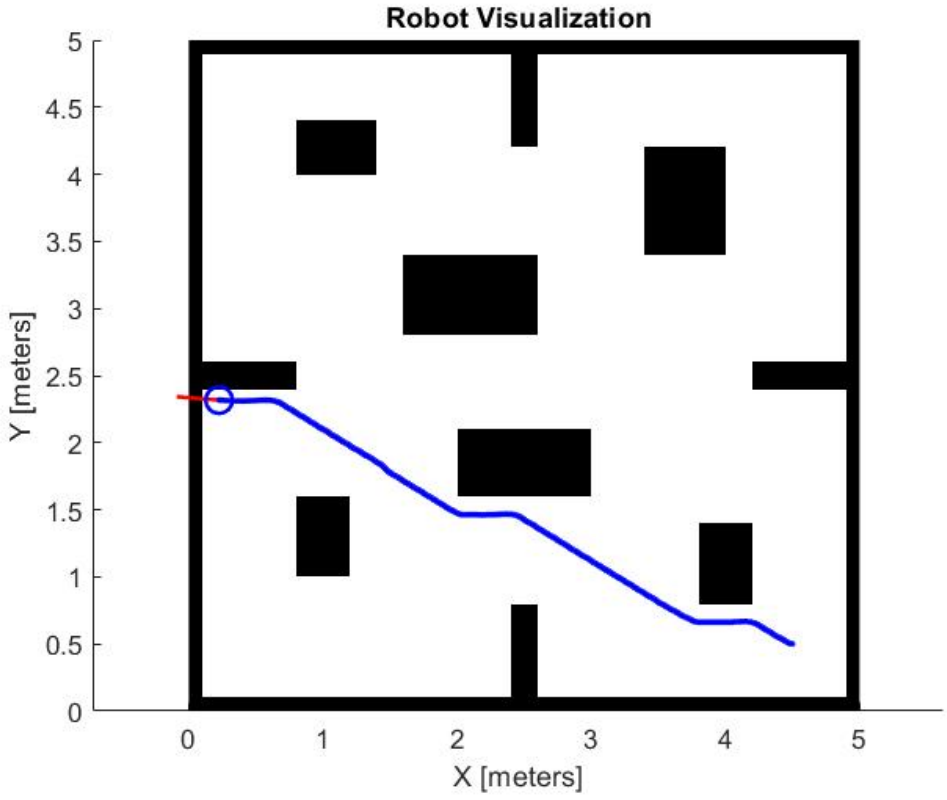
## ANEXO 43

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°2.



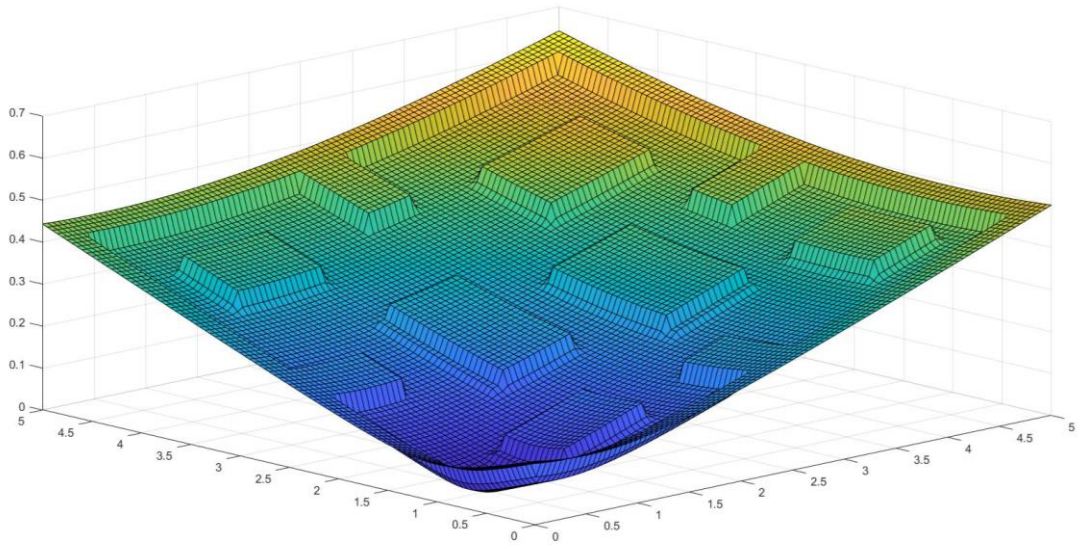
## ANEXO 44

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°2 GUIADO POR ALGORITMO PFPP.



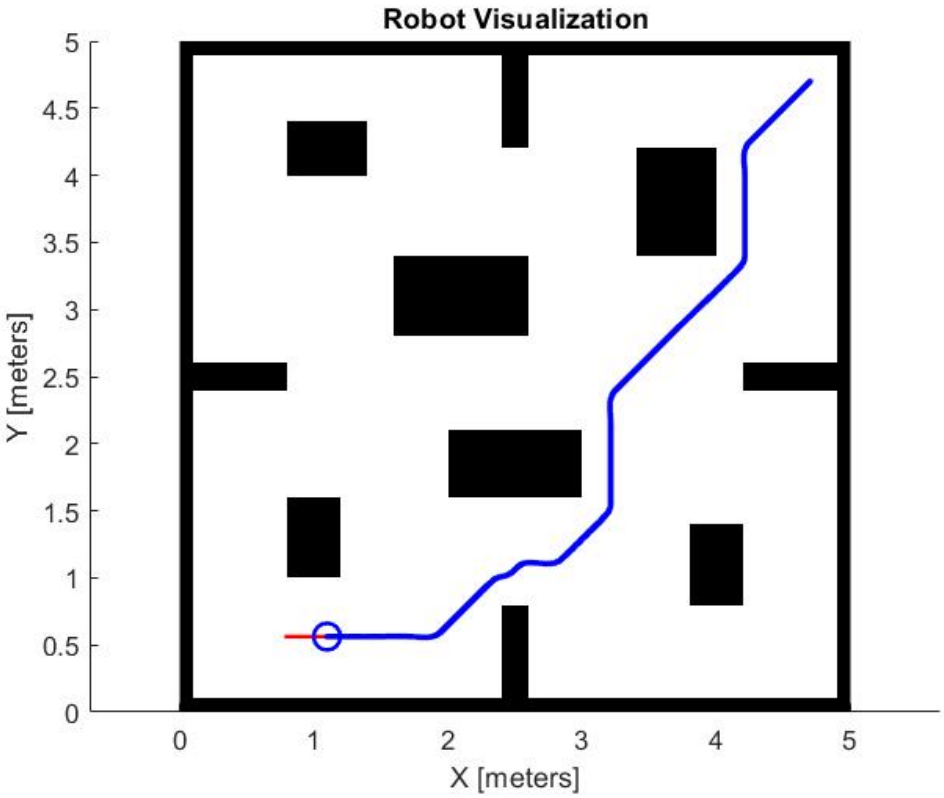
## ANEXO 45

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°3.



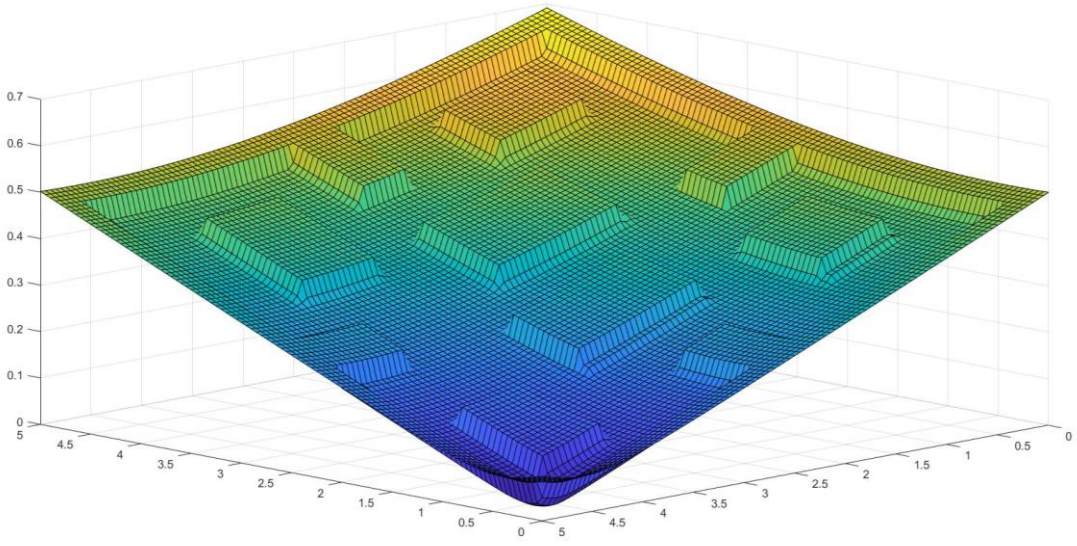
## ANEXO 46

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°3 GUIADO POR ALGORITMO PFPP.



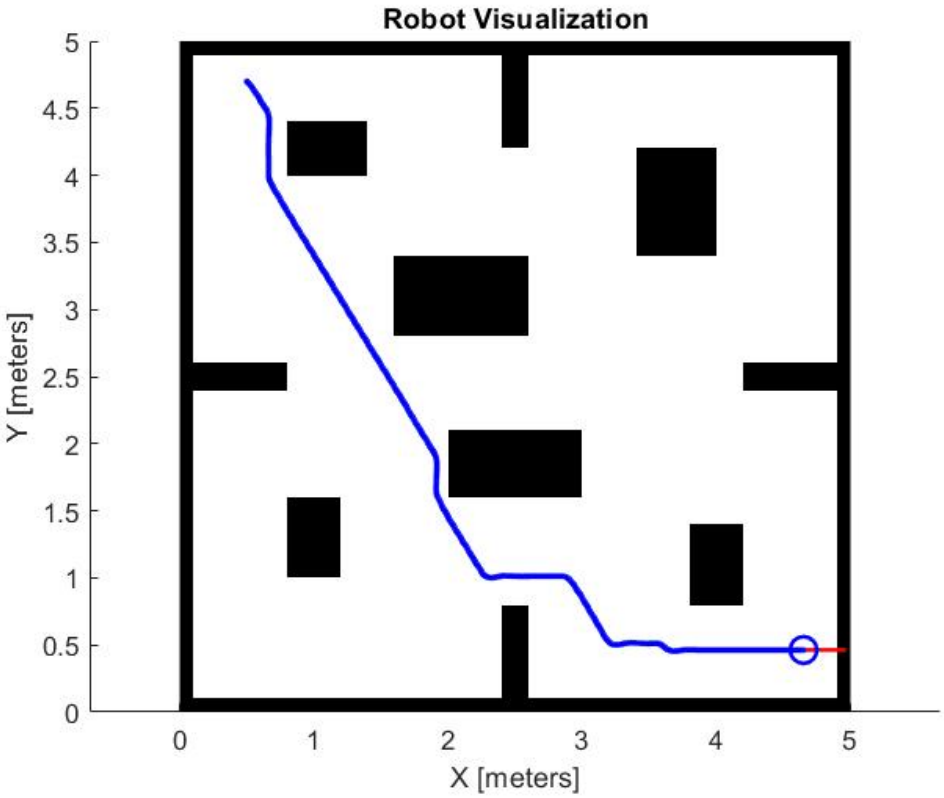
## ANEXO 47

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°4.



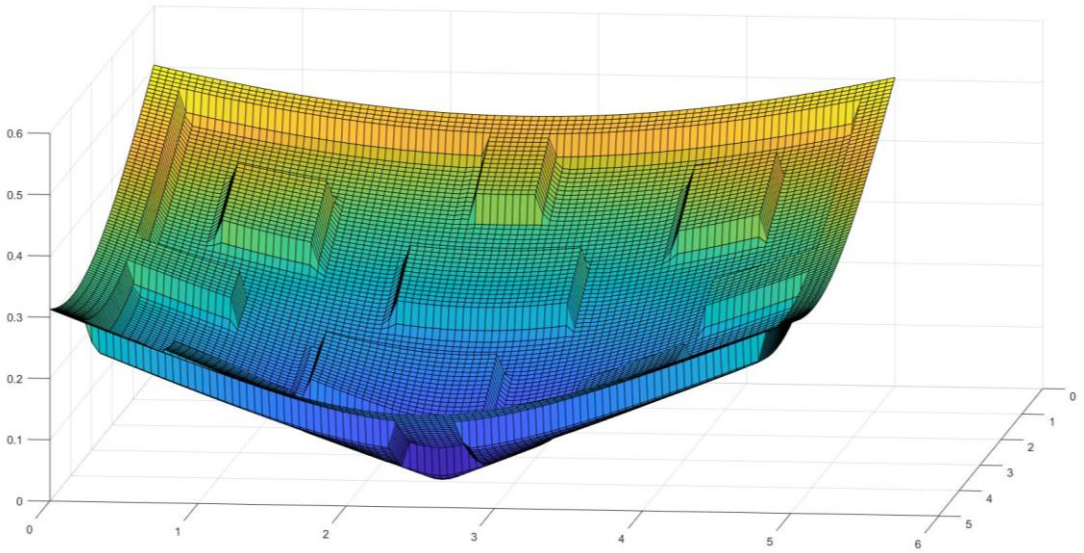

## ANEXO 48

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°4 GUIADO POR ALGORITMO PFPP.



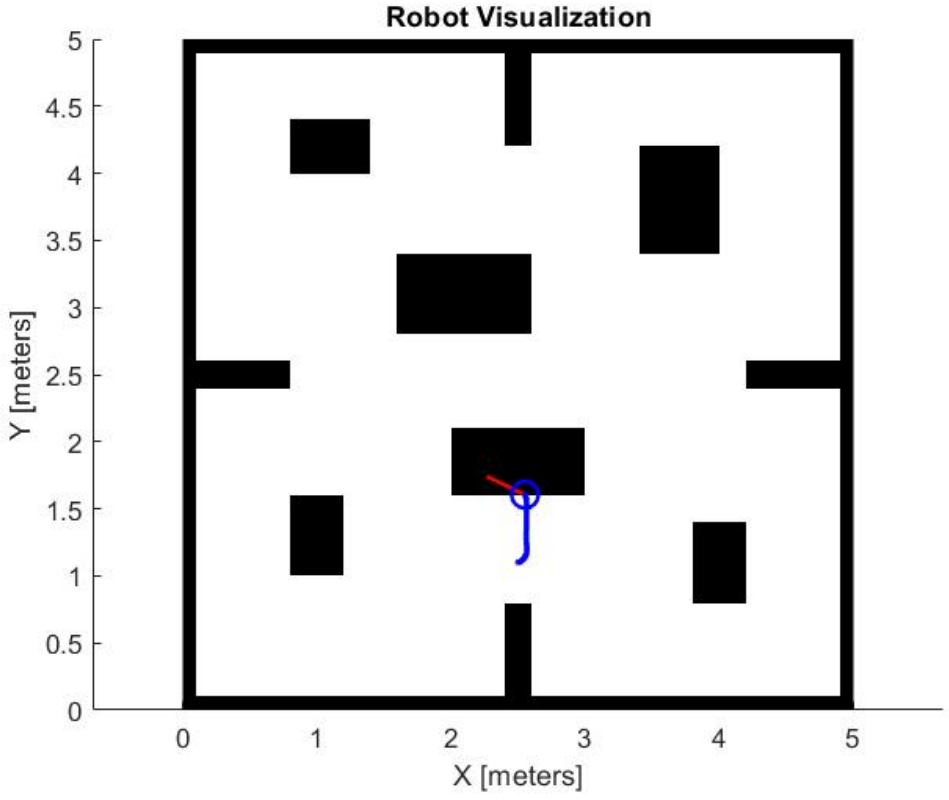
## ANEXO 49

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°5.



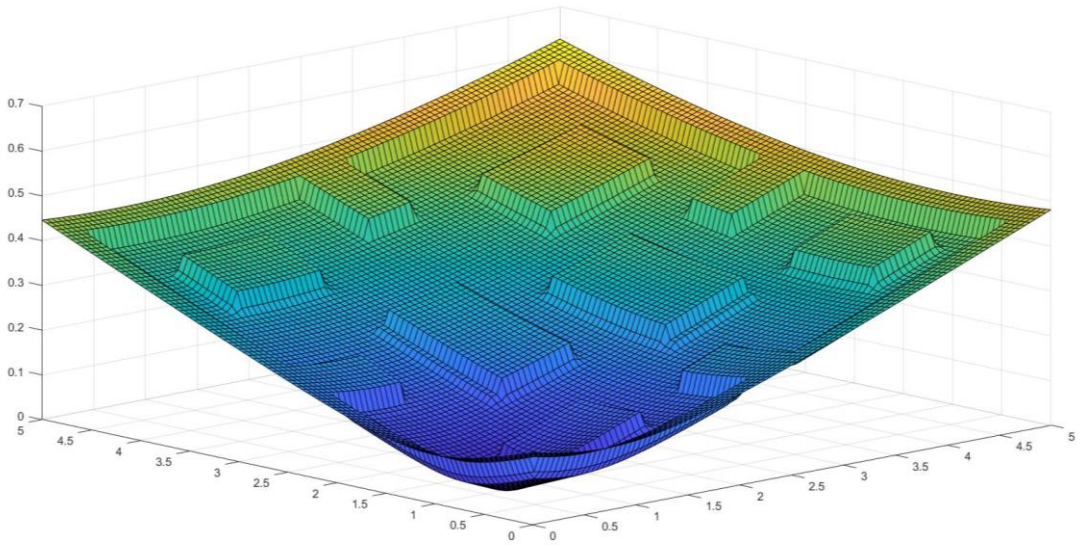

## ANEXO 50

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°5 GUIADO POR ALGORITMO PFPP.



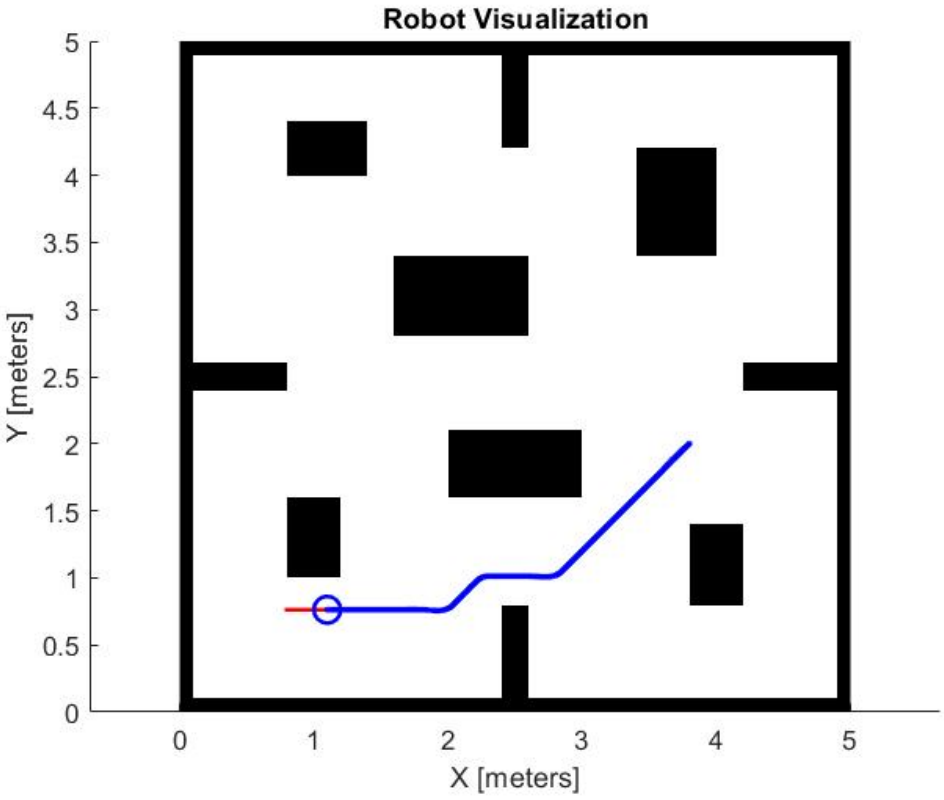
## ANEXO 51

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°6.



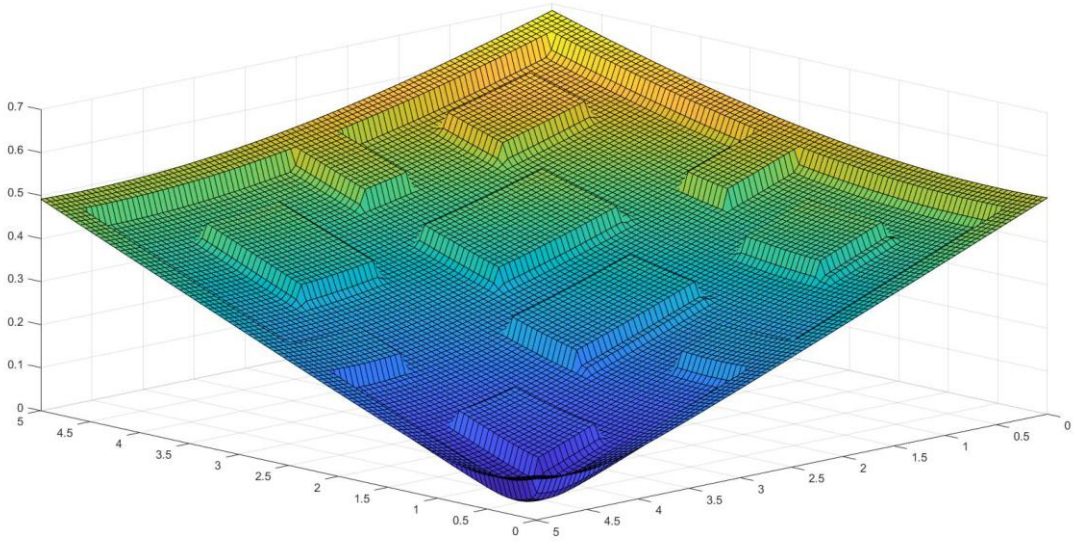
## ANEXO 52

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°6 GUIADO POR ALGORITMO PFPP.



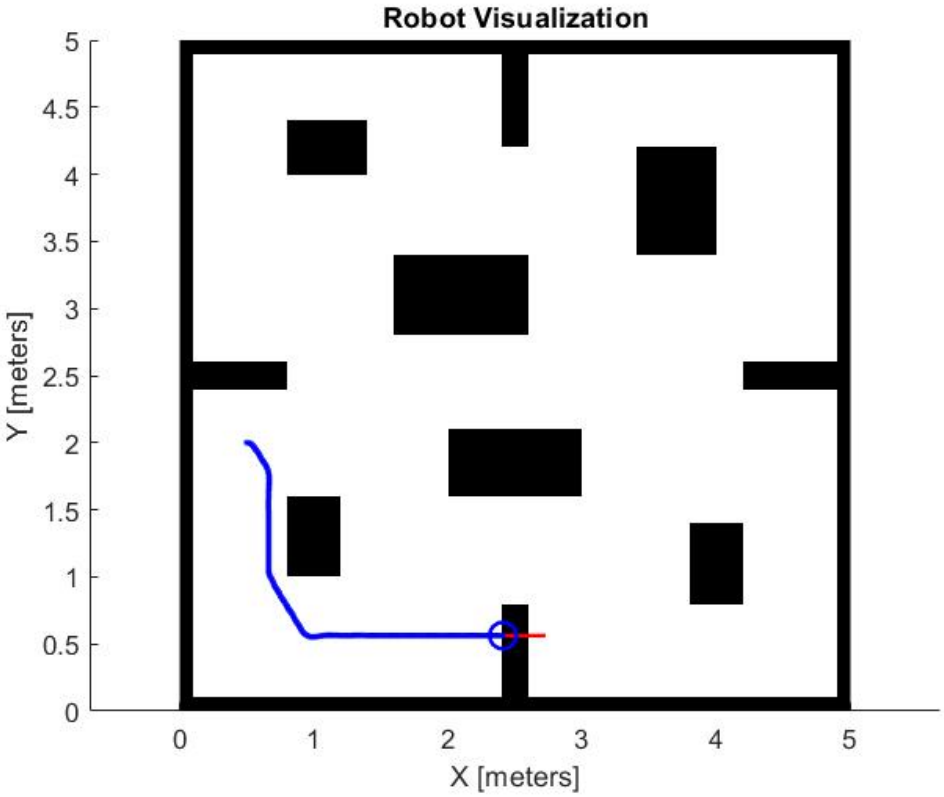
## ANEXO 53

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°7.



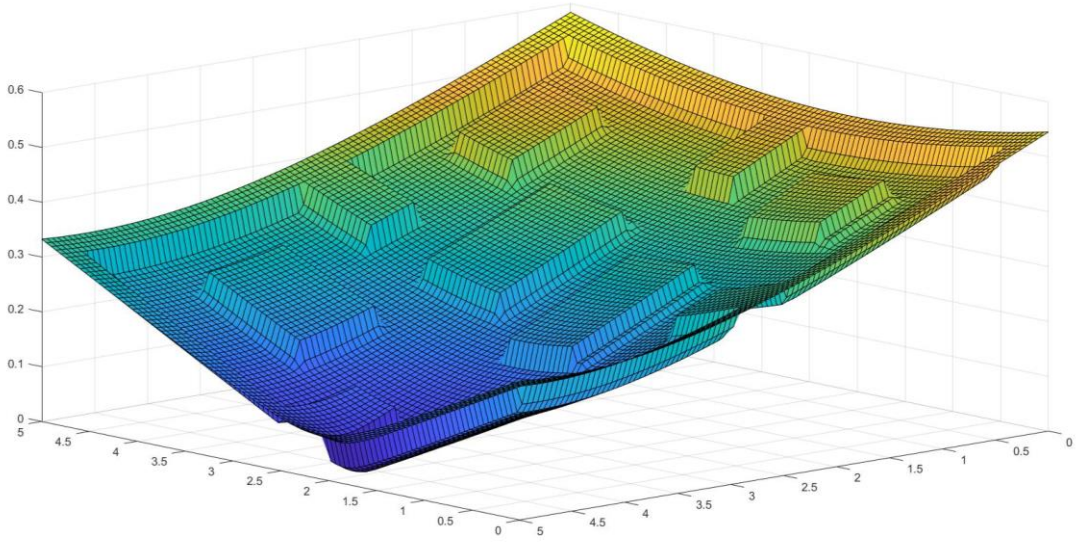
## ANEXO 54

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°7 GUIADO POR ALGORITMO PFPP.



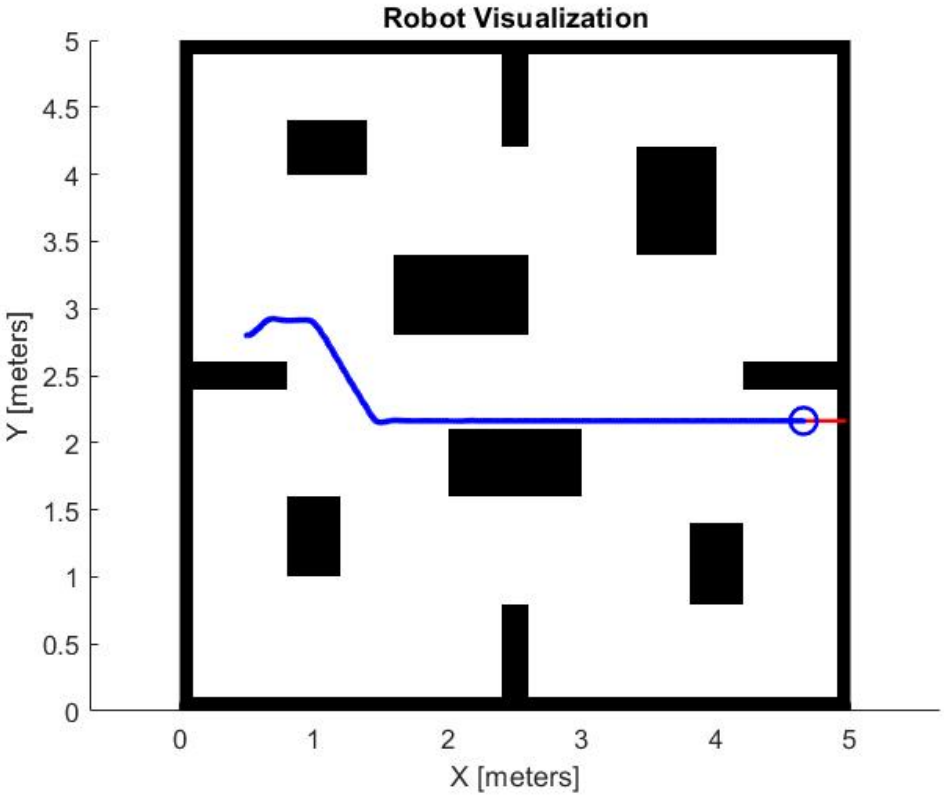
## ANEXO 55

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°8.



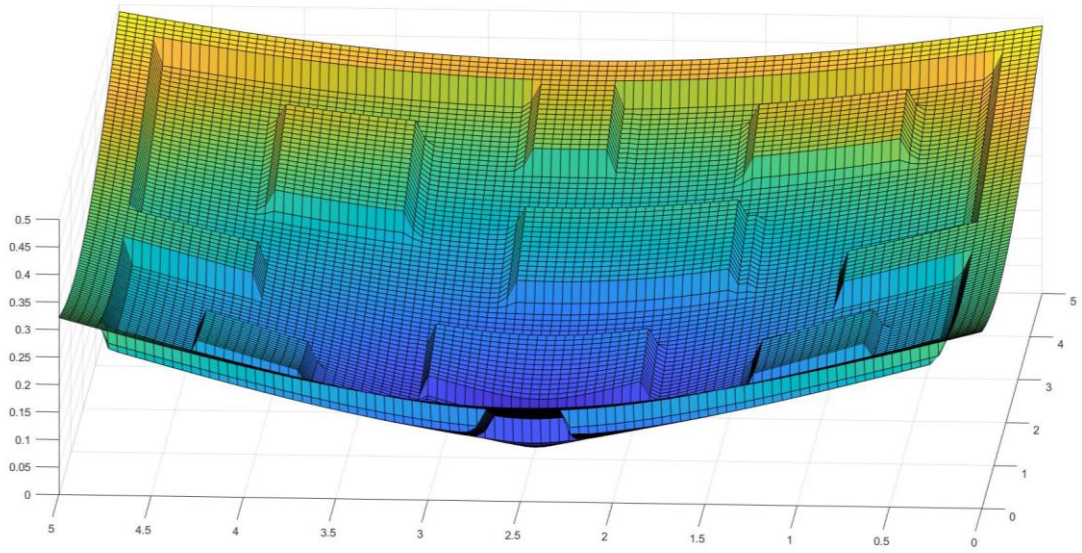

## ANEXO 56

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°8 GUIADO POR ALGORITMO PFPP.



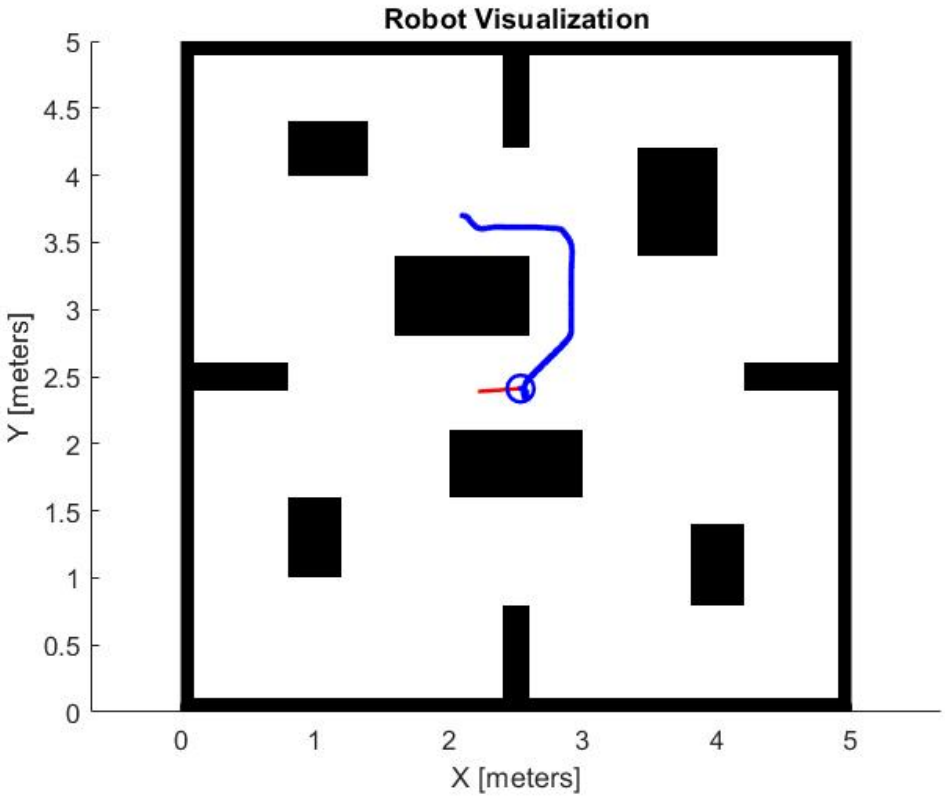
## ANEXO 57

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°9.



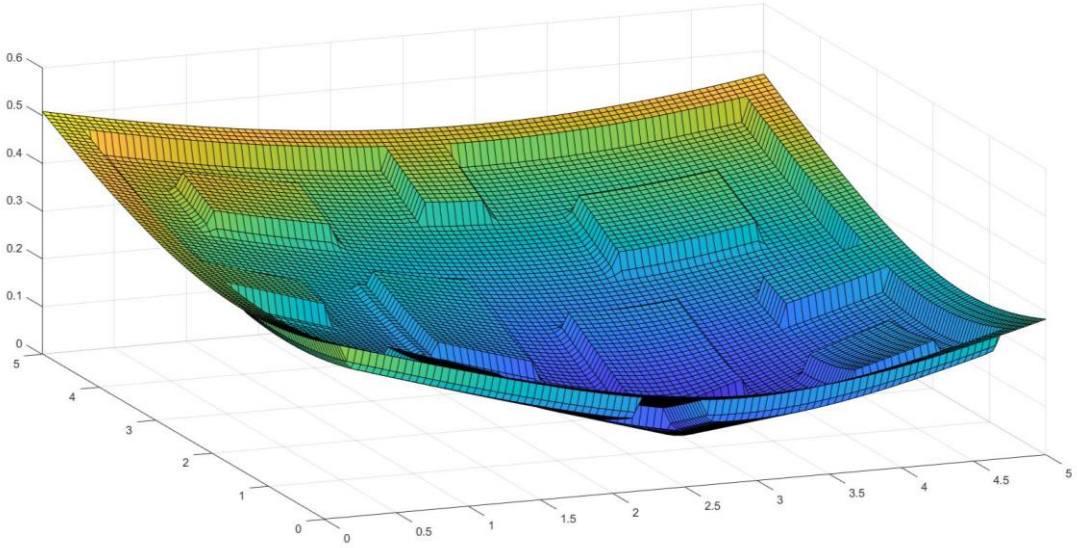

## ANEXO 58

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°9 GUIADO POR ALGORITMO PFPP.



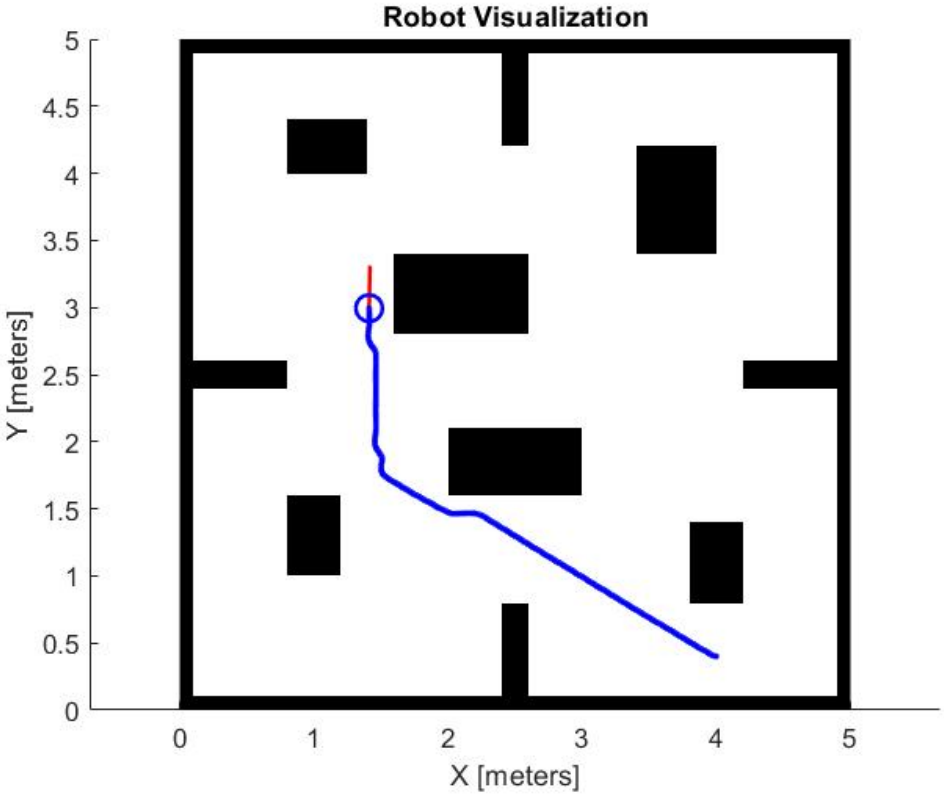
## ANEXO 59

## CAMPO POTENCIAL GENERADO POR ALGORITMO PFPP PARA RUTA DE PRUEBA N°10.



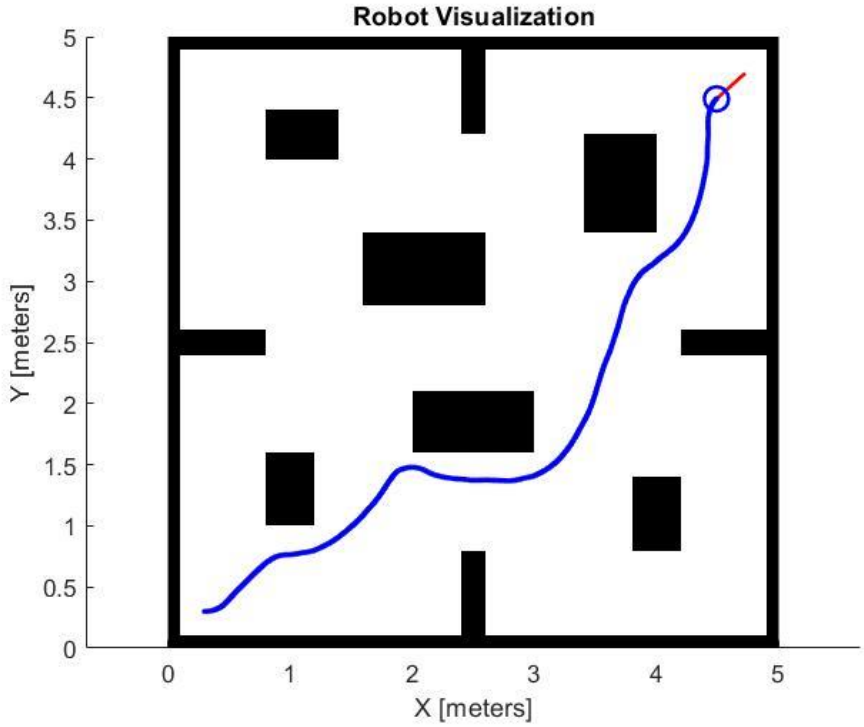

## ANEXO 60

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°10 GUIADO POR ALGORITMO PFPP.



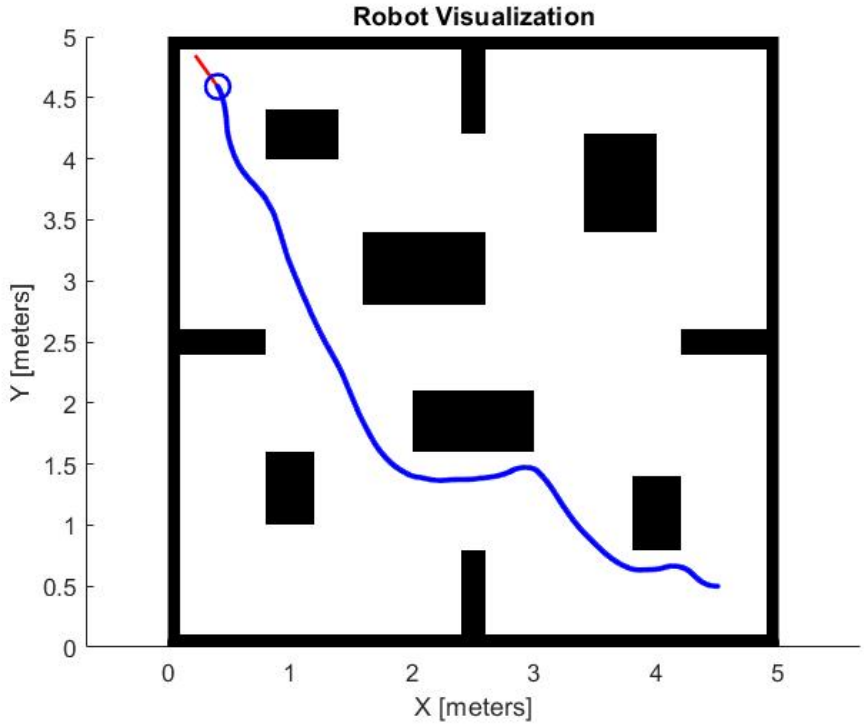
## ANEXO 61

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°1 GUIADO POR ALGORITMO VFF.



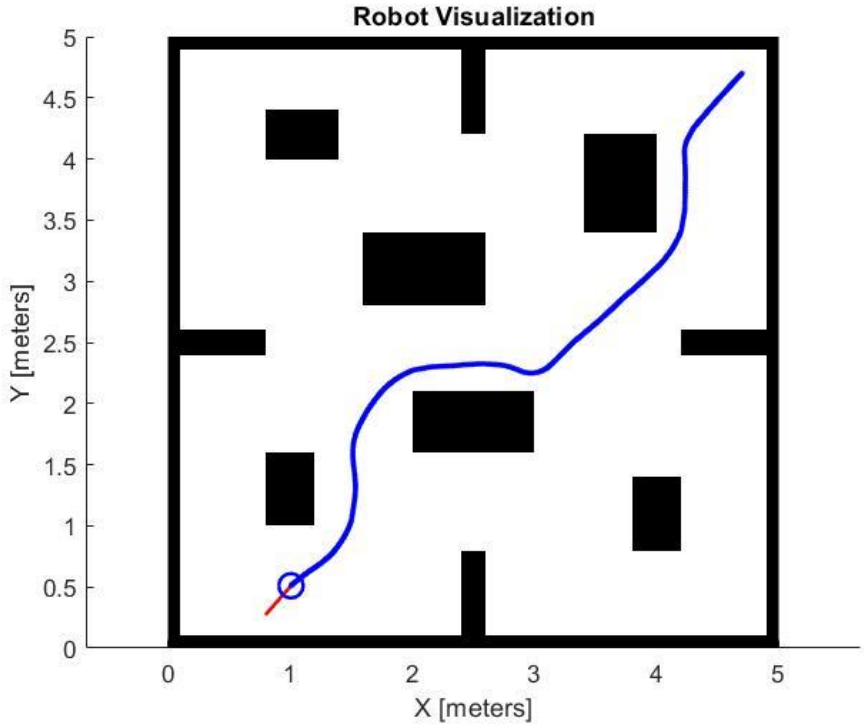

## ANEXO 62

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°2 GUIADO POR ALGORITMO VFF.



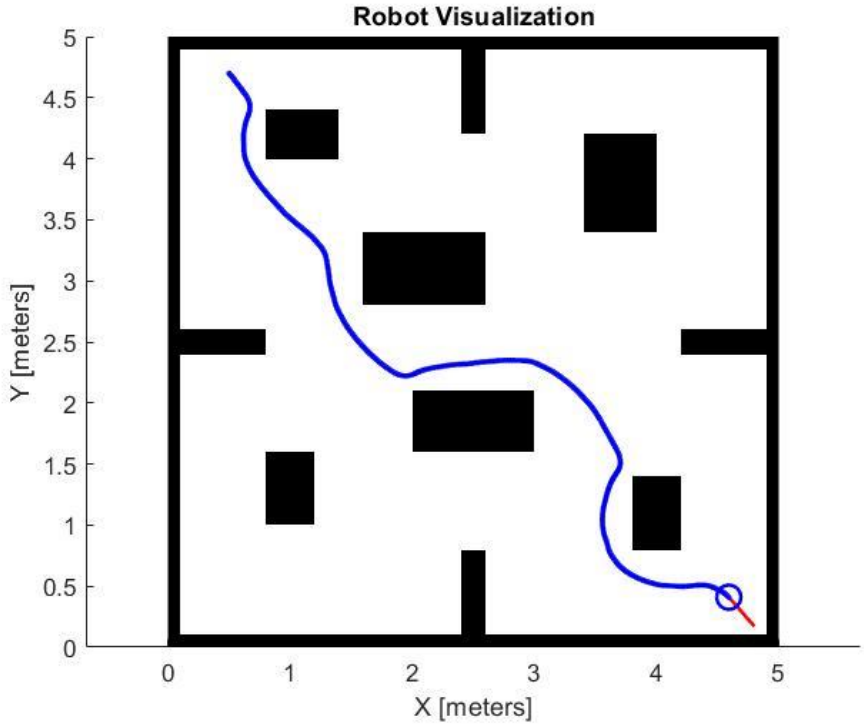
## ANEXO 63

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°3 GUIADO POR ALGORITMO VFF.



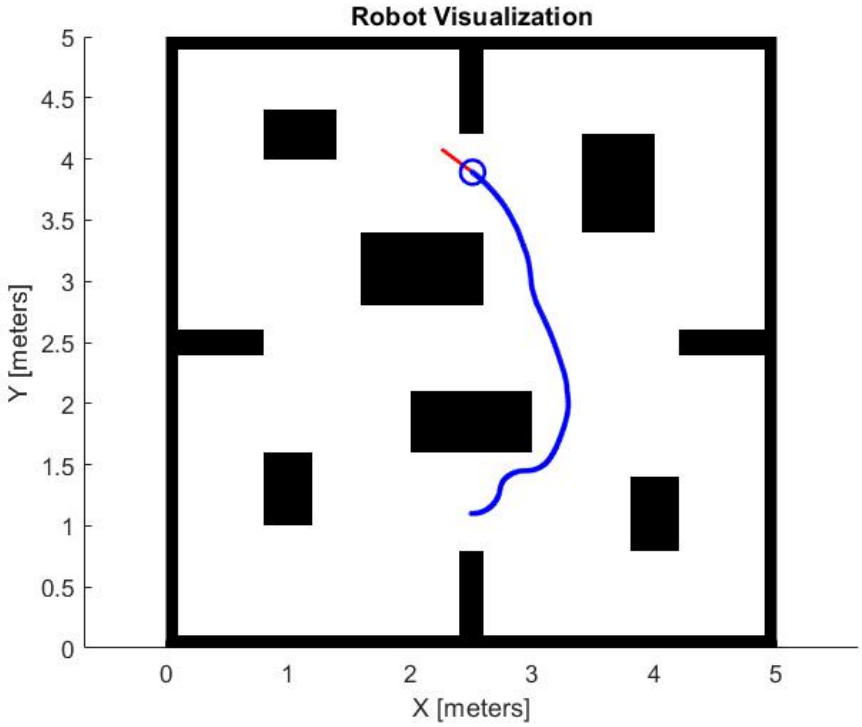

## ANEXO 64

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°4 GUIADO POR ALGORITMO VFF.



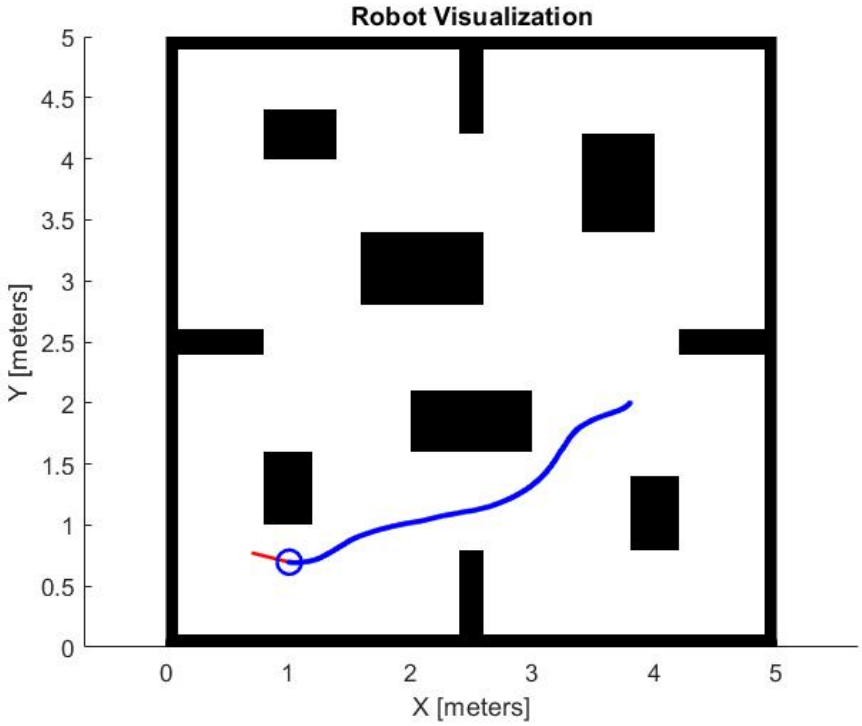
## ANEXO 65

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°5 GUIADO POR ALGORITMO VFF.



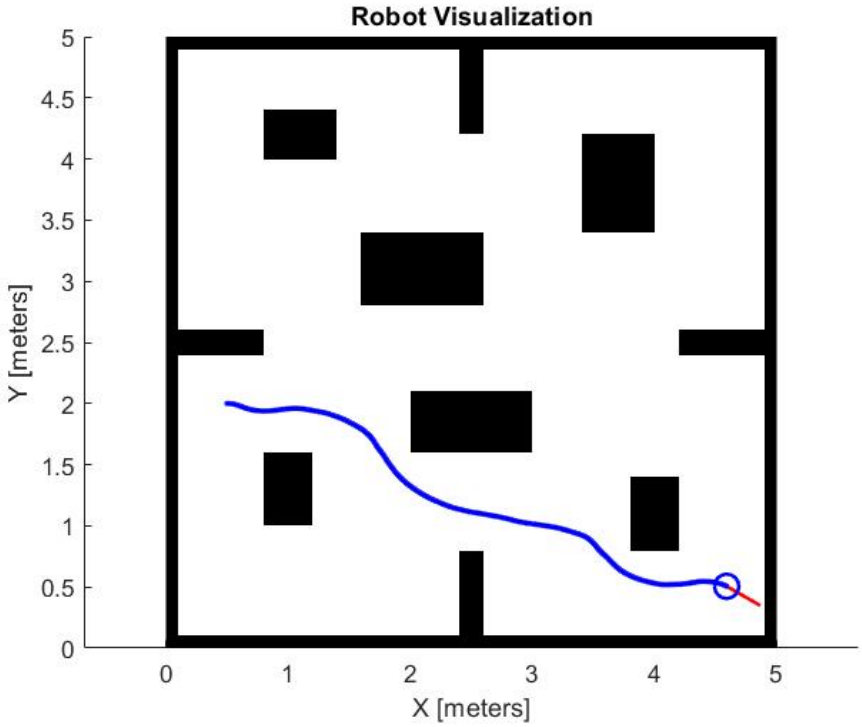

## ANEXO 66

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°6 GUIADO POR ALGORITMO VFF.



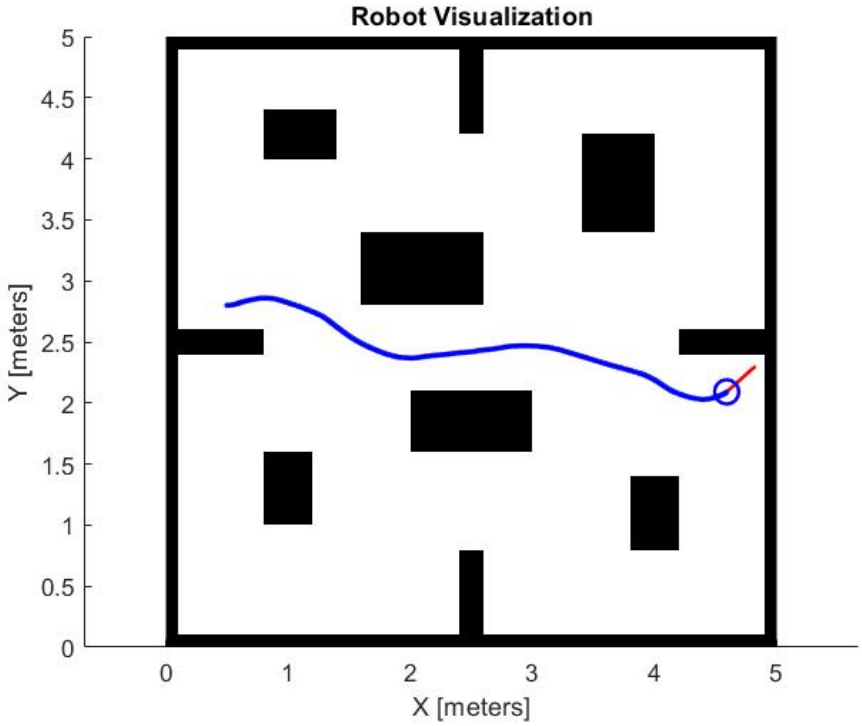
## ANEXO 67

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°7 GUIADO POR ALGORITMO VFF.



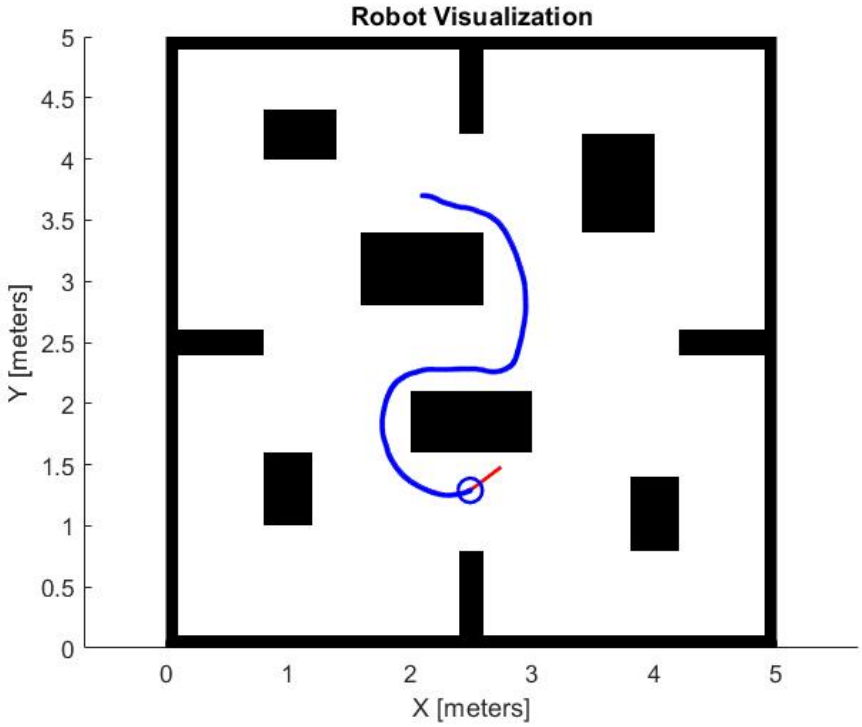

## ANEXO 68

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°8 GUIADO POR ALGORITMO VFF.



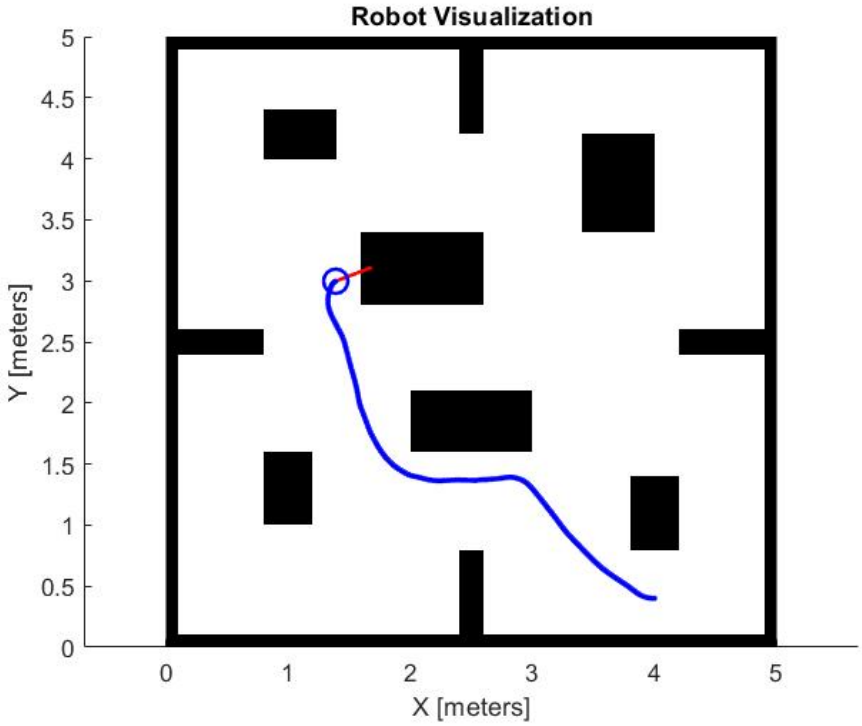
## ANEXO 69

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°9 GUIADO POR ALGORITMO VFF.



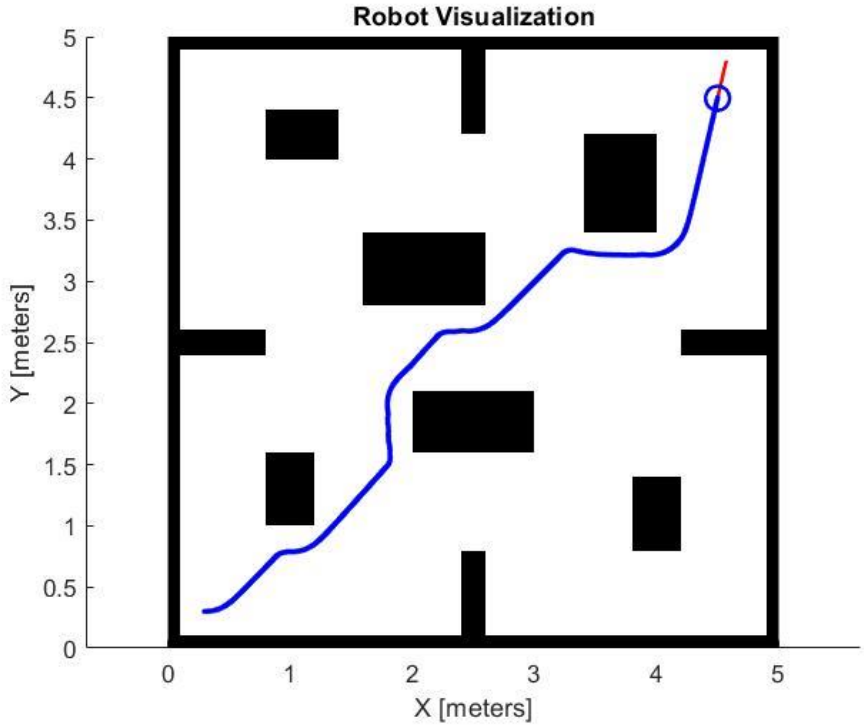

## ANEXO 70

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°10 GUIADO POR ALGORITMO VFF.



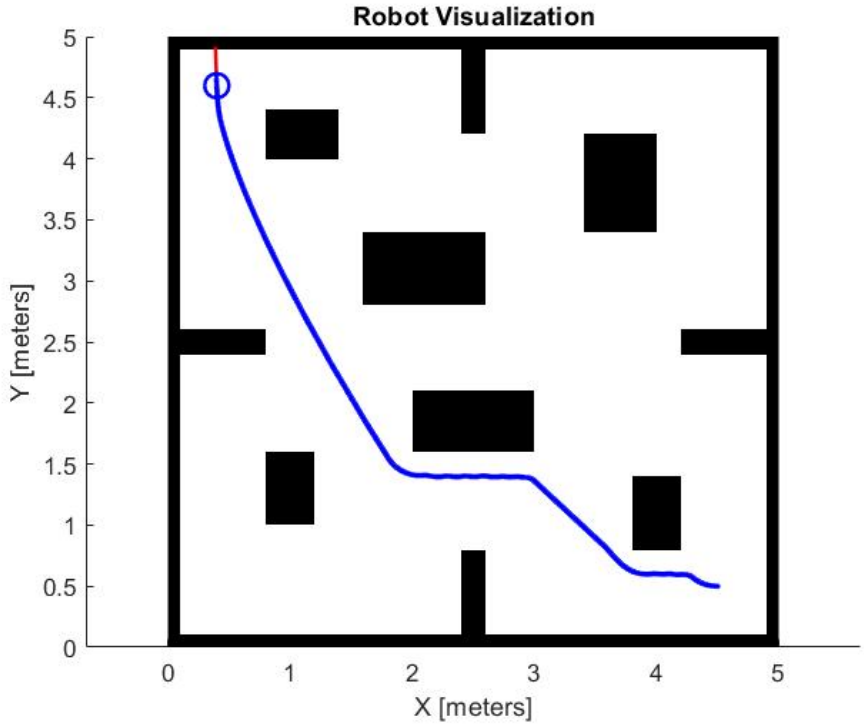
## ANEXO 71

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°1 GUIADO POR ALGORITMO VFH.



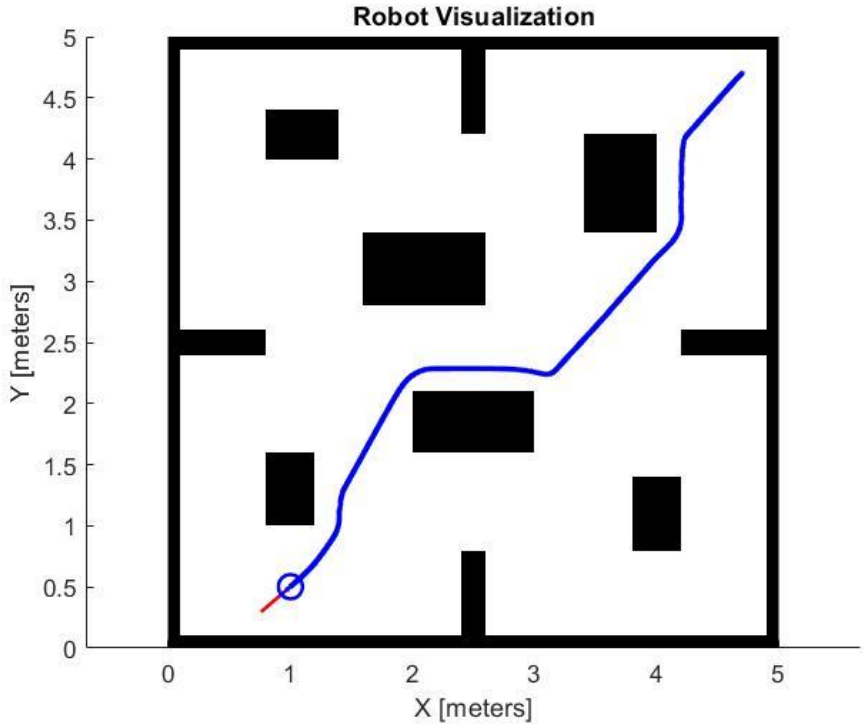

## ANEXO 72

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°2 GUIADO POR ALGORITMO VFH.



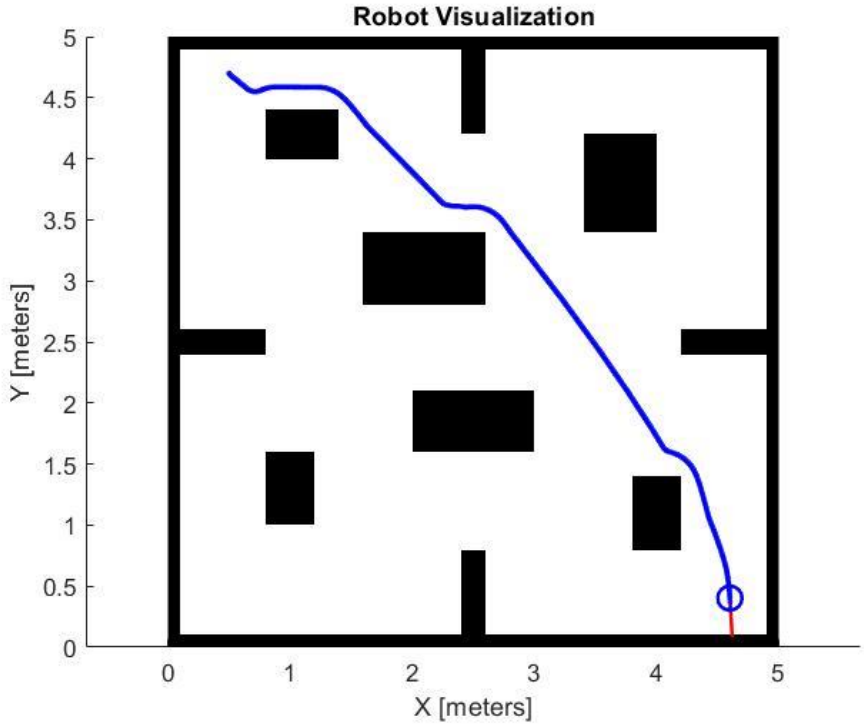
## ANEXO 73

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°3 GUIADO POR ALGORITMO VFH.



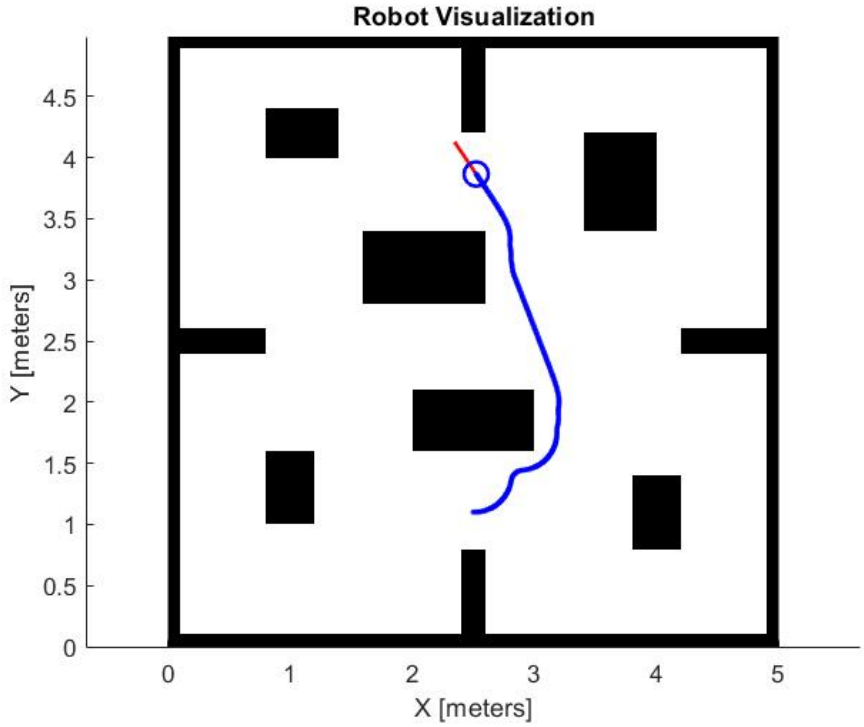

## ANEXO 74

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°4 GUIADO POR ALGORITMO VFH.



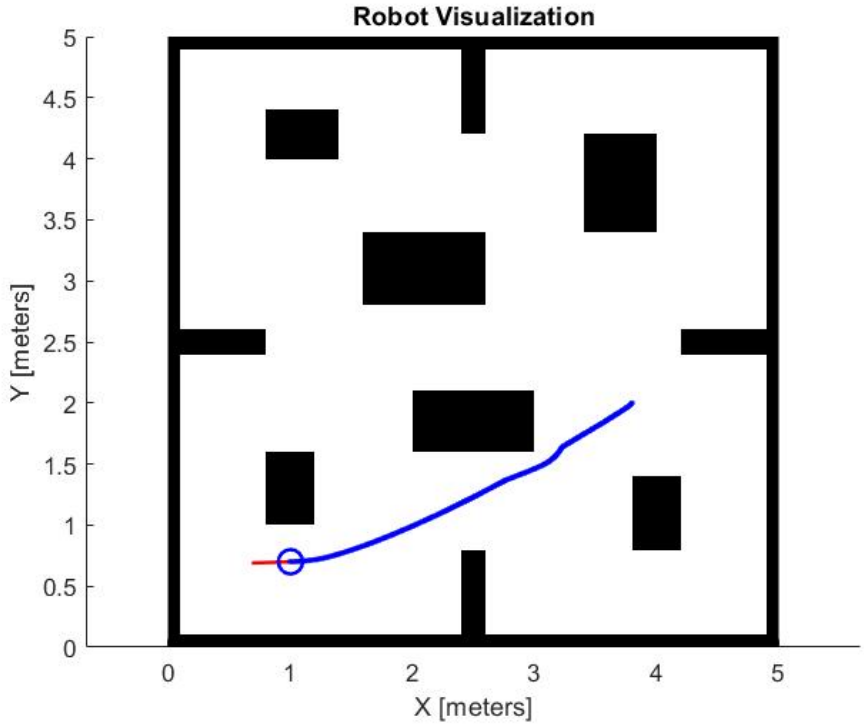
## ANEXO 75

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°5 GUIADO POR ALGORITMO VFH.



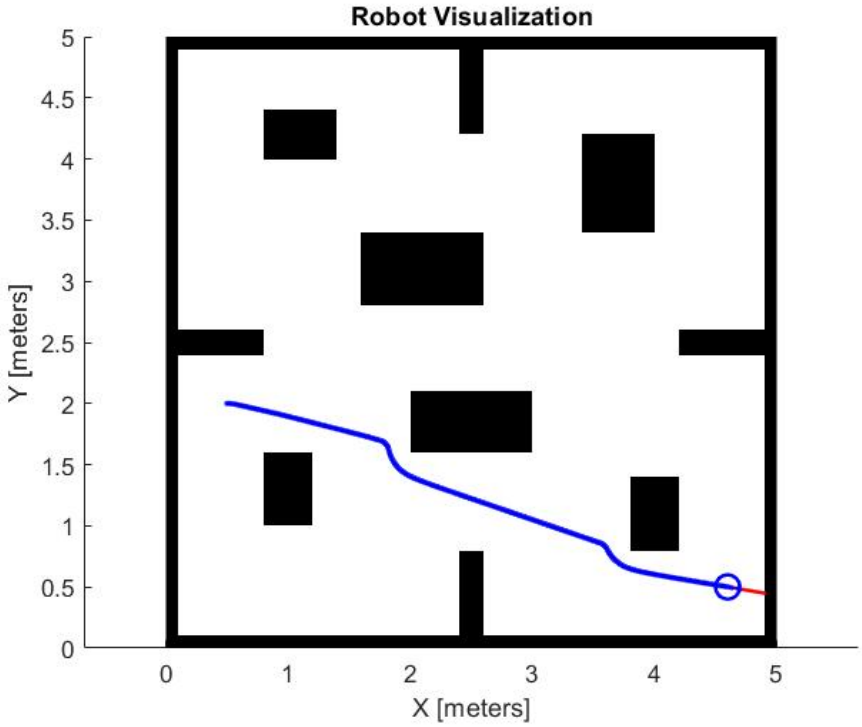

## ANEXO 76

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°6 GUIADO POR ALGORITMO VFH.



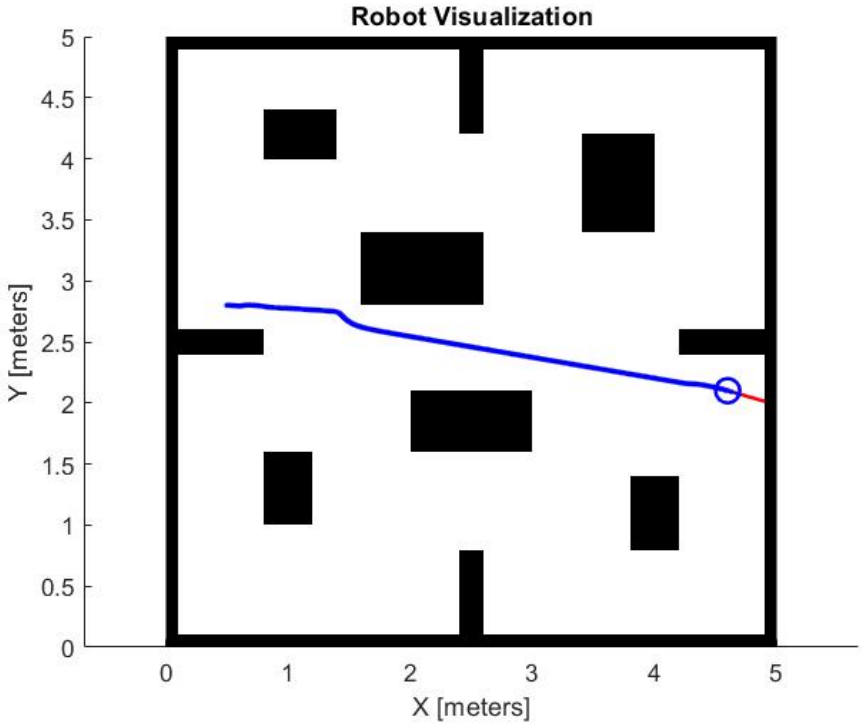
## ANEXO 77

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°7 GUIADO POR ALGORITMO VFH.



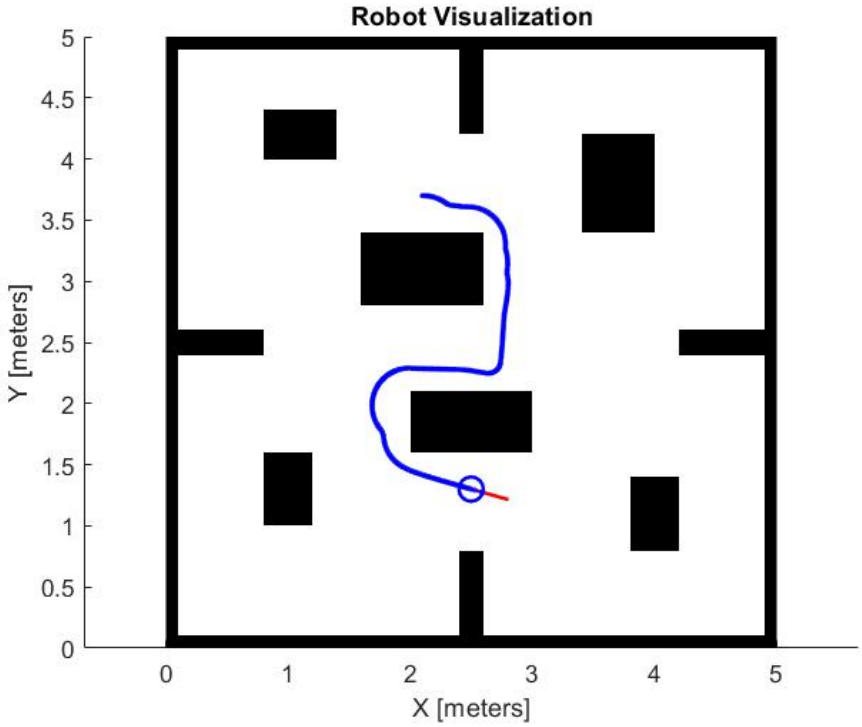

## ANEXO 78

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°8 GUIADO POR ALGORITMO VFH.



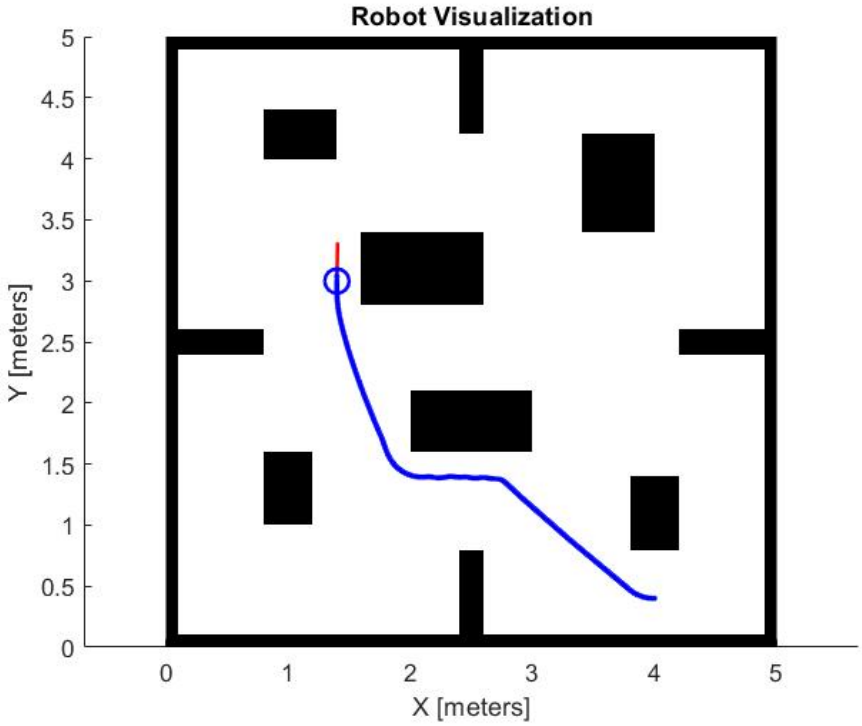
## ANEXO 79

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°9 GUIADO POR ALGORITMO VFH.




## ANEXO 80

## SIMULACIÓN DE ROBOT MÓVIL EN RUTA DE PRUEBA N°10 GUIADO POR ALGORITMO VFH.



# ANEXO 81

## CÓDIGO UTILIZADO PARA IMPLEMENTAR ALGORITMO RRT.

```
%ALGORITMO RRT
%DECLARACIÓN DE POSICION INICIAL, OBJETIVO Y VARIABLES
x_inicial=4
y_inicial=0.4
objetivo_x=1.4
objetivo_y=3
vector_padre=[1]
resolucion=0.1
paso=0.05
goal=0
goal_inverso=0
rango_llegada=0.05
%DECLARACIÓN DEL MAPA EN EL QUE SE APLICARÁ EL ALGORITMO
mapa_actual=mapa_1(resolucion)
mapa_actual_trabajo=mapa_1_trabajo(resolucion)
 longitud_mapa_metros=5
%DECLARACION DEL RANGO ALEATORIO DENTRO DEL CUAL SE GENERARÁ LOS PUNTOS
%ALEATORIOS
rango_aleatorio=longitud_mapa_metros*100
%INICIO DE LA LÓGICA DEL ALGORITMO
vector_puntos_x=[x_inicial]
vector_puntos_y=[y_inicial]
contador_vector=1
while goal==0
    px_rand=(randi([0 rango_aleatorio],1,1))/100
    py_rand=(randi([0 rango_aleatorio],1,1))/100
    valor_ocupacion=getOccupancy(mapa_actual_trabajo, [px_rand py_rand])
    if valor_ocupacion<0.2
        dist_min=9999
        for i=contador_vector:-1:1
            dist_nueva=sqrt(((px_rand-vector_puntos_x(i))^(2))+((py_rand-vector_puntos_y(i))^(2)))
            if dist_nueva<dist_min
                dist_min=dist_nueva
                p_cercano=i
            end
        end
        if dist_min>paso
            x3=nuevo_punto_x(vector_puntos_x(p_cercano), px_rand, vector_puntos_y(p_cercano),
py_rand,paso)
            y3=nuevo_punto_y(vector_puntos_x(p_cercano), px_rand, vector_puntos_y(p_cercano), py_rand, x3)
        else
            x3=px_rand
            y3=py_rand
        end
        px_nuevo=x3
        py_nuevo=y3
        valor_ocupacion=getOccupancy(mapa_actual_trabajo, [px_nuevo py_nuevo])
        if valor_ocupacion<0.2
            vector_puntos_x=[vector_puntos_x px_nuevo]
            vector_puntos_y=[vector_puntos_y py_nuevo]
            vector_padre=[vector_padre p_cercano]
            contador_vector=contador_vector+1
            goal=llegada_objetivo(objetivo_x, objetivo_y, px_nuevo, py_nuevo, rango_llegada)
        end
        if goal==1
            vector_solucion_x=[px_nuevo]
            vector_solucion_y=[py_nuevo]
            posicion_actual=contador_vector
            while goal_inverso==0
                padre_posicion_actual=vector_padre(posicion_actual)
                posicion_actual=padre_posicion_actual
                vector_solucion_x=[vector_puntos_x(posicion_actual) vector_solucion_x]
                vector_solucion_y=[vector_puntos_y(posicion_actual) vector_solucion_y]
                punto_x_actual=vector_puntos_x(posicion_actual)
                punto_y_actual=vector_puntos_y(posicion_actual)
                goal_inverso=llegada_punto_inicial(x_inicial, y_inicial, punto_x_actual, punto_y_actual)
            end
        end
    end
end
```

```
show(mapa_actual)
hold on
for i=1:+1:contador_vector
    plot(vector_puntos_x(i),vector_puntos_y(i),'.')
    hold on
end
plot (vector_solucion_x, vector_solucion_y,'r','LineWidth',2)
sim('Modelo_RRT_normal')
```

## ANEXO 82

## CÓDIGO UTILIZADO PARA IMPLEMENTAR ALGORITMO RRT*.

```
%ALGORITMO RRT*
%DECLARACIÓN DE POSICION INICIAL, OBJETIVO Y VARIABLES
x_inicial=4
y_inicial=0.4
objetivo_x=1.4
objetivo_y=3
angulo_inicial=180
vector_padre=[1]
resolucion_busqueda=0.125
resolucion=0.1
rango_llegada=0.05
paso=0.05
goal=0
goal_inverso=0
vector_costos=[0]
%DECLARACIÓN DEL MAPA EN EL QUE SE APLICARÁ EL ALGORITMO
mapa_actual=mapa_1(resolucion)
mapa_actual_trabajo=mapa_1_trabajo(resolucion)
longitud_mapa_metros=5
%DECLARACION DEL RANGO ALEATORIO DENTRO DEL CUAL SE GENERARÁ LOS PUNTOS
%ALEATORIOS
rango_aleatorio=longitud_mapa_metros*100
%INICIO DE LA LÓGICA DEL ALGORITMO
vector_puntos_x=[x_inicial]
vector_puntos_y=[y_inicial]
contador_vector=1
while goal==0
    px_rand=(randi([0 rango_aleatorio],1,1))/100
    py_rand=(randi([0 rango_aleatorio],1,1))/100
    valor_ocupacion=getOccupancy(mapa_actual_trabajo, [px_rand py_rand])
    if valor_ocupacion<0.2
        dist_min=9999
        for i=contador_vector:-1:1
            dist_nueva=sqrt(((px_rand-vector_puntos_x(i))^(2))+((py_rand-vector_puntos_y(i))^(2)))
            if dist_nueva<dist_min
                dist_min=dist_nueva
                p_cercano=i
            end
        end
        if dist_min>paso
            x3=nuevo_punto_x(vector_puntos_x(p_cercano), px_rand, vector_puntos_y(p_cercano),
py_rand,paso)
            y3=nuevo_punto_y(vector_puntos_x(p_cercano), px_rand, vector_puntos_y(p_cercano), py_rand, x3)
        else
            x3=px_rand
            y3=py_rand
        end
        px_nuevo=x3
        py_nuevo=y3
        valor_ocupacion=getOccupancy(mapa_actual_trabajo, [px_nuevo py_nuevo])
        if valor_ocupacion<0.2
            vector_puntos_x=[vector_puntos_x px_nuevo]
            vector_puntos_y=[vector_puntos_y py_nuevo]
            vector_padre=[vector_padre p_cercano]
            contador_vector=contador_vector+1
            % INICIO MODIFICACION A RRT ESTRELLA
            p_actual_auxiliar_x=px_nuevo
            p_actual_auxiliar_y=py_nuevo
```

```
            p_actual_auxiliar=contador_vector
            distancia_auxiliar=0
            costo_punto_actual=0
            %While colocado para darle el costo a cada nuevo punto que se
            %genere
            while p_actual_auxiliar_x~=x_inicial && p_actual_auxiliar_y~=y_inicial
                padre_auxiliar=vector_padre(p_actual_auxiliar)
                padre_auxiliar_x=vector_puntos_x(padre_auxiliar)
                padre_auxiliar_y=vector_puntos_y(padre_auxiliar)

distancia_corta=distancia_2_puntos(p_actual_auxiliar_x,p_actual_auxiliar_y,padre_auxiliar_x,padre_auxiliar
_y)
                costo_punto_actual=costo_punto_actual+distancia_corta
                p_actual_auxiliar=padre_auxiliar
                p_actual_auxiliar_x=padre_auxiliar_x
                p_actual_auxiliar_y=padre_auxiliar_y
            end
            vector_costos=[vector_costos costo_punto_actual]
            limite_vecino_x_superior=px_nuevo+resolucion_busqueda
            limite_vecino_x_inferior=px_nuevo-resolucion_busqueda
            limite_vecino_y_superior=py_nuevo+resolucion_busqueda
            limite_vecino_y_inferior=py_nuevo-resolucion_busqueda
            vector_vecinos=[]
            %FOR COLOCADO PARA IDENTIFICAR LOS VECINOS DEL PUNTO GENERADO
            for i=1:+1:contador_vector
                posible_vecino_x=vector_puntos_x(i)
                posible_vecino_y=vector_puntos_y(i)
                if posible_vecino_x>=limite_vecino_x_inferior &&
posible_vecino_x<=limite_vecino_x_superior
                    if posible_vecino_y>=limite_vecino_y_inferior &&
posible_vecino_y<=limite_vecino_y_superior
                        if posible_vecino_x~=px_nuevo && posible_vecino_y~=py_nuevo
                            vector_vecinos=[vector_vecinos i]
                        end
                    end
                end
            end
            tamano_vector_vecinos=size(vector_vecinos)
            longitud_vector_vecinos=tamano_vector_vecinos(2)
            costo_actual=costo_punto_actual
            padre_actual=vector_padre(contador_vector)
            padre_actual_x=vector_puntos_x(padre_actual)
            padre_actual_y=vector_puntos_y(padre_actual)
            %FOR COLOCADO PARA IDENTIFICAR SI UNO DE LAS VECINOS TIENE UN
            %MENOR COSTO QUE EL PADRE ACTUAL DEL PUNTO GENERADO
            for i=1:+1:longitud_vector_vecinos
                vecino_actual=vector_vecinos(i)
                vecino_actual_x=vector_puntos_x(vecino_actual)
                vecino_actual_y=vector_puntos_y(vecino_actual)
                if vecino_actual_x~=padre_actual_x && vecino_actual_y~=padre_actual_y
                    distancia_corta=distancia_2_puntos(px_nuevo,py_nuevo,vecino_actual_x,vecino_actual_y)
                    costo_vecino=vector_costos(vecino_actual)+distancia_corta
                    if costo_vecino<costo_actual
                        costo_actual=costo_vecino
                        vector_padre(contador_vector)=vecino_actual
                        vector_costos(contador_vector)=costo_actual
                    end
                end
            end
            padre_actual=vector_padre(contador_vector)
            padre_actual_x=vector_puntos_x(padre_actual)
            padre_actual_y=vector_puntos_y(padre_actual)
            %FOR COLOCADO PARA VERIFICAR SI UNO DE LOS VECINOS DEL PUNTO
            %GENERADO REDUCIRÍA SU COSTO AL CONECTARSE AL MISMO
            for i=1:+1:longitud_vector_vecinos
                vecino_actual=vector_vecinos(i)
                vecino_actual_x=vector_puntos_x(vecino_actual)
                vecino_actual_y=vector_puntos_y(vecino_actual)
                if vecino_actual_x~=padre_actual_x && vecino_actual_y~=padre_actual_y
                    costo_vecino=vector_costos(vecino_actual)
                    distancia_corta=distancia_2_puntos(px_nuevo,py_nuevo,vecino_actual_x,vecino_actual_y)
                    costo_hipotetico=costo_actual+distancia_corta
                    if costo_hipotetico<costo_vecino
                        costo_vecino=costo_hipotetico
                        %CONDICIONALES COLOCADAS PARA EVITAR BUCLES
                        %INFINITOS GENERADOS POR UNIR UN PUNTO X A OTRO
                        %PUNTO CUYO ANTECESOR ES EL MISMO PUNTO X
```

```
                        padre_1_punto_actual=vector_padre(contador_vector)
                        padre_2_punto_actual=vector_padre(padre_1_punto_actual)
                        padre_3_punto_actual=vector_padre(padre_2_punto_actual)
                        padre_4_punto_actual=vector_padre(padre_3_punto_actual)
                        padre_5_punto_actual=vector_padre(padre_4_punto_actual)
                        padre_6_punto_actual=vector_padre(padre_5_punto_actual)
                        padre_7_punto_actual=vector_padre(padre_6_punto_actual)
                        padre_8_punto_actual=vector_padre(padre_7_punto_actual)
                        if padre_2_punto_actual~=vecino_actual && padre_1_punto_actual~=vecino_actual
                            if padre_3_punto_actual~=vecino_actual && padre_4_punto_actual~=vecino_actual
                                if padre_5_punto_actual~=vecino_actual &&
padre_6_punto_actual~=vecino_actual
                                    if padre_7_punto_actual~=vecino_actual &&
padre_8_punto_actual~=vecino_actual
                                        vector_padre(vecino_actual)=contador_vector
                                        vector_costos(vecino_actual)=costo_vecino
                                    end
                                end
                            end
                        end
                    end
                end
            end
            % FIN DE MODIFICACION A RRT ESTRELLA
            goal=llegada_objetivo(objetivo_x, objetivo_y, px_nuevo, py_nuevo, rango_llegada)
        end
        if goal==1
            % INICIO ULTIMA REVISION DE TODO EL CAMINO PARA IDENTIFICAR
            % POSIBLES REDUCCIONES DE COSTE FINALES
            punto_actual=contador_vector
            punto_actual_x=vector_puntos_x(punto_actual)
            punto_actual_y=vector_puntos_y(punto_actual)
            while punto_actual_x~=x_inicial && punto_actual_y~=y_inicial
                padre_actual=vector_padre(punto_actual)
                padre_actual_x=vector_puntos_x(padre_actual)
                padre_actual_y=vector_puntos_y(padre_actual)
                limite_vecino_x_superior=punto_actual_x+resolucion_busqueda
                limite_vecino_x_inferior=punto_actual_x-resolucion_busqueda
                limite_vecino_y_superior=punto_actual_y+resolucion_busqueda
                limite_vecino_y_inferior=punto_actual_y-resolucion_busqueda
                vector_vecinos=[]
                for i=1:+1:contador_vector
                    posible_vecino_x=vector_puntos_x(i)
                    posible_vecino_y=vector_puntos_y(i)
                    if posible_vecino_x<=limite_vecino_x_superior &&
posible_vecino_x>=limite_vecino_x_inferior
                        if posible_vecino_y<=limite_vecino_y_superior &&
posible_vecino_y>=limite_vecino_y_inferior
                            if posible_vecino_x~=padre_actual_x && posible_vecino_y~=padre_actual_y
                                if posible_vecino_x~=punto_actual_x && posible_vecino_y~=punto_actual_y
                                    vector_vecinos=[vector_vecinos i]
                                end
                            end
                        end
                    end
                end
                costo_actual=vector_costos(punto_actual)
                tamano_vector_vecinos=size(vector_vecinos)
                longitud_vector_vecinos=tamano_vector_vecinos(2)
                for i=1:+1:longitud_vector_vecinos
                    vecino_actual=vector_vecinos(i)
                    vecino_actual_x=vector_puntos_x(vecino_actual)
                    vecino_actual_y=vector_puntos_y(vecino_actual)
                    costo_vecino=vector_costos(vecino_actual)

distancia_corta=distancia_2_puntos(punto_actual_x,punto_actual_y,vecino_actual_x,vecino_actual_y)
                    costo_hipotetico=distancia_corta+costo_vecino
                    if costo_hipotetico<costo_actual
                        costo_actual=costo_hipotetico
                        %CONDICIONALES COLOCADAS PARA EVITAR BUCLES
                        %INFINITOS GENERADOS POR UNIR UN PUNTO X A OTRO
                        %PUNTO CUYO ANTECESOR ES EL MISMO PUNTO X
                        padre_1_punto_actual=vector_padre(vecino_actual)
                        padre_2_punto_actual=vector_padre(padre_1_punto_actual)
                        padre_3_punto_actual=vector_padre(padre_2_punto_actual)
                        padre_4_punto_actual=vector_padre(padre_3_punto_actual)
                        padre_5_punto_actual=vector_padre(padre_4_punto_actual)
```

```
                        padre_6_punto_actual=vector_padre(padre_5_punto_actual)
                        padre_7_punto_actual=vector_padre(padre_6_punto_actual)
                        padre_8_punto_actual=vector_padre(padre_7_punto_actual)
                        padre_9_punto_actual=vector_padre(padre_8_punto_actual)
                        padre_10_punto_actual=vector_padre(padre_9_punto_actual)
                        padre_11_punto_actual=vector_padre(padre_10_punto_actual)
                        padre_12_punto_actual=vector_padre(padre_11_punto_actual)
                        padre_13_punto_actual=vector_padre(padre_12_punto_actual)
                        padre_14_punto_actual=vector_padre(padre_13_punto_actual)
                        if padre_2_punto_actual~=punto_actual && padre_1_punto_actual~=punto_actual
                            if padre_3_punto_actual~=punto_actual && padre_4_punto_actual~=punto_actual
                                if padre_5_punto_actual~=punto_actual && padre_6_punto_actual~=punto_actual
                                    if padre_8_punto_actual~=punto_actual &&
padre_7_punto_actual~=punto_actual
                                        if padre_9_punto_actual~=punto_actual &&
padre_10_punto_actual~=punto_actual
                                            if padre_11_punto_actual~=punto_actual &&
padre_12_punto_actual~=punto_actual
                                                if padre_14_punto_actual~=punto_actual &&
padre_13_punto_actual~=punto_actual
                                                    vector_padre(punto_actual)=vecino_actual
                                                    vector_costos(punto_actual)=costo_actual
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
                punto_actual=vector_padre(punto_actual)
                punto_actual_x=vector_puntos_x(punto_actual)
                punto_actual_y=vector_puntos_y(punto_actual)
            end
            % FIN ULTIMA REVISION
            vector_solucion_x=[px_nuevo]
            vector_solucion_y=[py_nuevo]
            posicion_actual=contador_vector
            while goal_inverso==0
                padre_posicion_actual=vector_padre(posicion_actual)
                posicion_actual=padre_posicion_actual
                vector_solucion_x=[vector_puntos_x(posicion_actual) vector_solucion_x]
                vector_solucion_y=[vector_puntos_y(posicion_actual) vector_solucion_y]
                punto_x_actual=vector_puntos_x(posicion_actual)
                punto_y_actual=vector_puntos_y(posicion_actual)
                goal_inverso=llegada_punto_inicial(x_inicial, y_inicial, punto_x_actual, punto_y_actual)
            end
        end
    end
end
show(mapa_actual)
hold on
for i=1:+1:contador_vector
    plot(vector_puntos_x(i),vector_puntos_y(i),'.')
    hold on
end
plot (vector_solucion_x, vector_solucion_y,'r','LineWidth',2)

sim('Modelo_RRT_normal')
```

## ANEXO 83

## CÓDIGO UTILIZADO PARA IMPLEMENTAR ALGORITMO PFPP.

```
%% ALGORITMO PFPP
%DECLARACIÓN DEL MAPA
resolucion=0.1
mapa_actual=mapa_1(resolucion)
mapa_actual_trabajo=mapa_1_trabajo(resolucion)
%DECLARACIÓN DE LA VARIABLE SAE, ETA Y DISTANCIA UMBRAL DEL OBJETIVO
sae=1
eta=0.0001
d_objetivo=0.1
%DECLARACIÓN DE LA POSICIÓN INICIAL
x_inicial=4
```

```
y_inicial=0.4
angulo_inicial=180
%DECLARACIÓN DEL OBJETIVO
objetivo_x=1.4
objetivo_y=3
%DECLARACIÓN DE LOS VECTORES X & Y QUE SERVIRÁN PARA CALCULAR EL POTENCIAL
%DE CADA CELDA DEL MAPA
vector_x=[]
vector_y=[]
%DECLARACIÓN DE LA MATRIZ Z (CONTENEDORA DEL POTENCIAL) Y LA MATRIZ Z FATT
%(CONTENEDORA DE LA GRADIENTE)
matriz_z=[]
matriz_z_Fatt=[]
%DECLARACIÓN DE VARIABLES AUXILIARES PARA RECORRER EL MAPA
contador_x=0
contador_y=0
%DECLARACIÓN DE LA RESOLUCION DE LA MATRIZ
resolucion_matriz=0.05
%DECLARACION DE LA LONGITUD DEL MAPA
longitud_mapa_metros=5
%DECLARACIÓN DEL UMBRAL PARA OBSTÁCULOS "D", SE LO TOMA COMO 6 VECES LA
%RESOLUCION DE LA MATRIZ
D_obstaculo=0.3
%DECLARACIÓN DE UNA VARIABLE AUXILIAR PARA LA BUSQUEDA DE ESPACIOS OCUPADOS
%E EL PUNTO ACTUAL
D_busqueda=resolucion_matriz*5
%INICIO DE LA LÓGICA DEL ALGORITMO
for i=0:+resolucion_matriz:longitud_mapa_metros
    contador_x=contador_x+1
    x=i
    vector_x=[vector_x i]
    vector_y=[vector_y i]
    for j=0:+resolucion_matriz:longitud_mapa_metros
        contador_y=contador_y+1
        y=j
        %CÁLCULO DE LA FUERZA ATRACTORA DE LA CELDA ACTUAL
        distancia_actual_objetivo=distancia_2_puntos(x,y,objetivo_x,objetivo_y)
        %PARABÓLICA
        if distancia_actual_objetivo<=d_objetivo
            Uatt=(1/2)*sae*((distancia_2_puntos(x,y,objetivo_x,objetivo_y))^2)
            Fatt_x=sae*(x-objetivo_x)
            Fatt_y=sae*(y-objetivo_y)
        end
        %CÓNICA
        if distancia_actual_objetivo>d_objetivo
            Uatt=d_objetivo*sae*distancia_2_puntos(x,y,objetivo_x,objetivo_y)-(1/2)*sae*(d_objetivo^2)
            Fatt_x=(d_objetivo*sae*(x-objetivo_x))/distancia_2_puntos(x,y,objetivo_x,objetivo_y)
            Fatt_y=(d_objetivo*sae*(y-objetivo_y))/distancia_2_puntos(x,y,objetivo_x,objetivo_y)
        end
        %CÁLCULO DE LA FUERZA REPULSIVA DE LA CELDA ACTUAL
        %SE ANALIZA SI LA CELDA ACTUAL ANALIZADA ESTA OCUPADA, DE SER ASÍ,
        %SE LE ASIGNA UN VALOR DE REPULSIÓN PREDETERMINADO.
        dist_min=D_obstaculo
        valor_ocupancia_actual=getOccupancy(mapa_actual_trabajo,[i j])
        if valor_ocupancia_actual>0.9
            dist_min=D_obstaculo-(D_obstaculo*0.9)
        else
            %EN CASO DE QUE LA CELDA ACTUAL NO ESTE OCUPADA, SE HACE UN
            %ANÁLISIS DE LOS ALREDEDORES PARA SABER SI UNA CELDA CERCANA SE
            %ENCUENTRA OCUPADA. SE DECLARA UN VARIABLE "d_min" QUE AYUDA A
            %VERIFICAR QUE OBSTÁCULO ES EL MÁS CERCANO
            for i_aux=(i-D_busqueda):+resolucion_matriz:(i+D_busqueda)
                for j_aux=(j-D_busqueda):+resolucion_matriz:(j+D_busqueda)
                    valor_ocupancia_actual=getOccupancy(mapa_actual_trabajo,[i_aux,j_aux])
                    if valor_ocupancia_actual>0.9
                        distancia_ocupado=distancia_2_puntos(i_aux,j_aux,i,j)
                        if distancia_ocupado<dist_min
                            dist_min=distancia_ocupado
                        end
                    end
                end
            end
        end
        % SE EVALÚA SI LA DISTANCIA DEL OBSTÁCULO MÁS CERCANO "dist_min" ES
        % MENOR A LA DISTANCIA UMBRAL "D_obstáculo", SI LO ES, SE HACE EL
        % CÁLCULO DEL POTENCIAL REPULSIBO "U_rep", SINO, LA "U_rep" ES 0
        if dist_min<D_obstaculo
            Urep=(1/2*eta)*(((1/dist_min)-(1/D_obstaculo))^2)
        else
            Urep=0
        end
        matriz_z(contador_x,contador_y)=Uatt+Urep
        Fatt_mod=sqrt((Fatt_x^2)+(Fatt_y^2))
        matriz_z_Fatt(contador_x,contador_y)=Fatt_mod
    end
    contador_y=0
end
[X,Y]=meshgrid(vector_x,vector_y)
```

```
surf(X,Y,matriz_z)
```

## **ANEXO 84.** FUNCIÓN DE MATLAB UTILIZADA PARA IDENTIFICAR Y SEGUIR EL MENOR POTENCIAL PARA SIMULACIÓN DE ALGORITMO PFPP.

```
function [pos_deseada_x,pos_deseada_y,velocidad] =
Potential_rute(pos_x,pos_y,matriz_z,resolucion_matriz,matriz_z_Fatt,objetivo_x,objetivo_y)
Xa=pos_x
Ya=pos_y
Xn=Xa/resolucion_matriz
Yn=Ya/resolucion_matriz
distancia_objetivo=distancia_2_puntos(Xa,Ya,objetivo_x,objetivo_y)
pos_actual_x_matriz=fix(Xn)
pos_actual_y_matriz=fix(Yn)
vecino_menor_x=pos_actual_x_matriz
vecino_menor_y=pos_actual_y_matriz
if distancia_objetivo<0.015
    velocidad=0
else
    velocidad=matriz_z_Fatt(pos_actual_x_matriz,pos_actual_y_matriz)
end
valor_actual_z=matriz_z(pos_actual_x_matriz,pos_actual_y_matriz)
for i=pos_actual_x_matriz-1:+1:pos_actual_x_matriz+1
    for j=pos_actual_y_matriz-1:+1:pos_actual_y_matriz+1
        valor_vecino=matriz_z(i,j)
        if valor_vecino<valor_actual_z
            valor_actual_z=valor_vecino
            vecino_menor_x=i
            vecino_menor_y=j
        end
    end
end
pos_deseada_x=(vecino_menor_x*resolucion_matriz)+(resolucion_matriz/4)
pos_deseada_y=(vecino_menor_y*resolucion_matriz)+(resolucion_matriz/4)

end
```

## ANEXO 85

## FUNCIÓN DE MATLAB UTILIZADA PARA IMPLEMENTACIÓN Y SIMULACIÓN DE ALGORITMO VFF.

```
function [angulo_corregido, Velocidad,W_cero, giro_evasivo1, giro_evasivo0]   =
DIRECCION_GUIA(d1,d2,d3,d4,d5,d6,d7,d8,d9,x_robot,y_robot,angulo_actual,objetivo_x,objetivo_y)
%LOS ÁNGULOS DE LOS SENSORES RESPECTO AL CENTRO DEL ROBOT MÓVIL YA ESTÁN
%PREDEFINIDOS
theta_robot=angulo_actual
%CALCULO DE LOS ÁNGULOS EN LOS QUE SE ENCUENTRAN LOS SENSORES LIDAR
theta_d1=theta_robot-(pi/2)
theta_d2=theta_robot-(3*pi/8)
theta_d3=theta_robot-(pi/4)
theta_d4=theta_robot-(pi/8)
theta_d5=theta_robot
theta_d6=theta_robot+(pi/8)
theta_d7=theta_robot+(pi/4)
theta_d8=theta_robot+(3*pi/8)
theta_d9=theta_robot+(pi/2)
%CALCULO DE LA POSICIÓN X DE LOS OBSTÁCULOS CERCANOS, SE CALCULAN A PARTIR
%DE LA POSICIÓN X EN LA QUE SE ENCUENTRA EL ROBOT
Xobs1=x_robot+(d1*cos(theta_d1))
Xobs2=x_robot+(d2*cos(theta_d2))
Xobs3=x_robot+(d3*cos(theta_d3))
Xobs4=x_robot+(d4*cos(theta_d4))
Xobs5=x_robot+(d5*cos(theta_d5))
Xobs6=x_robot+(d6*cos(theta_d6))
Xobs7=x_robot+(d7*cos(theta_d7))
Xobs8=x_robot+(d8*cos(theta_d8))
Xobs9=x_robot+(d9*cos(theta_d9))
%CALCULO DE LA POSICIÓN Y DE LOS OBSTÁCULOS CERCANOS, SE CALCULAN A PARTIR
```

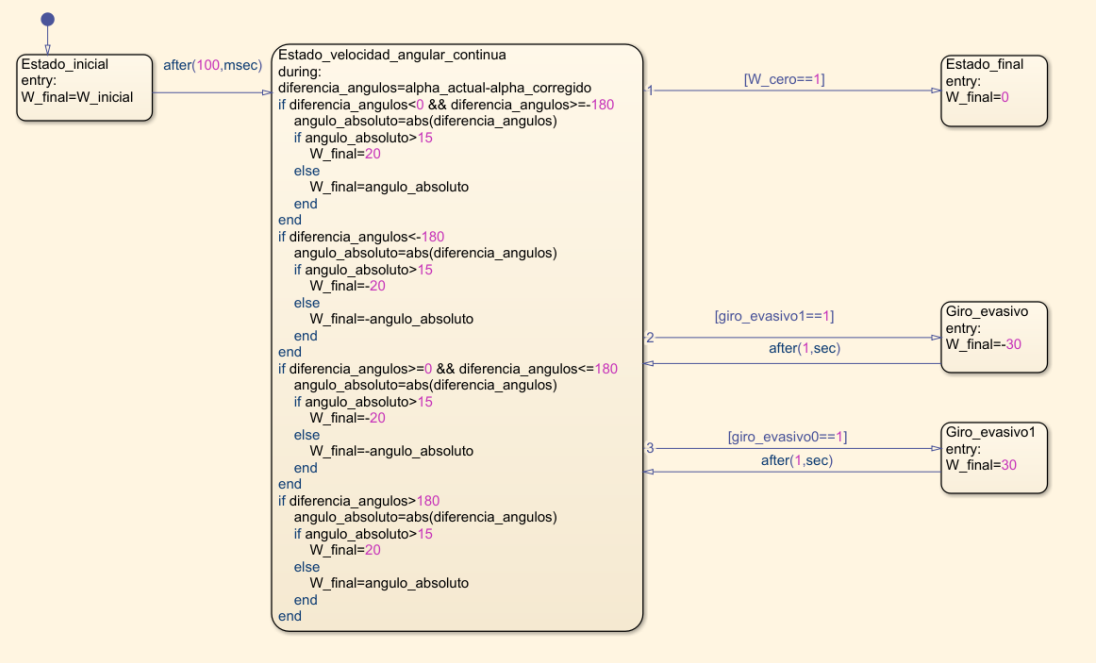
```
%DE LA POSICIÓN Y EN LA QUE SE ENCUENTRA EL ROBOT
Yobs1=y_robot+(d1*sin(theta_d1))
Yobs2=y_robot+(d2*sin(theta_d2))
Yobs3=y_robot+(d3*sin(theta_d3))
Yobs4=y_robot+(d4*sin(theta_d4))
Yobs5=y_robot+(d5*sin(theta_d5))
Yobs6=y_robot+(d6*sin(theta_d6))
Yobs7=y_robot+(d7*sin(theta_d7))
Yobs8=y_robot+(d8*sin(theta_d8))
Yobs9=y_robot+(d9*sin(theta_d9))
%DECLARACIÓN DEL OBJETIVO
xt=objetivo_x
yt=objetivo_y
dt=distancia_2_puntos(x_robot,y_robot,xt,yt)
%DEFINICIÓN DE LAS CONSTANTES Fcr Y Fca, DE REPULSIÓN Y ATRACCIÓN
%RESPECTIVAMENTE. SE AGREGA UNA CONDICIÓN PARA UN CRECIMIENTO MAYOR DEL
%Fca, DE MANERA QUE NO SEA REPELIDO CUANDO EL OBJETIVO SE ENCUENTRE MUY
%CERCA DE OBSTÁCULOS
Fcr=1
if dt<0.25
    Fca=60
else
    Fca=9
end
%DEFINICIÓN DE UNA VARIABLE PARA UN GIRO EVASIVO DE EMERGENCIA EN CASOS DE
%SIMETRÍA PERFECTA
giro_evasivo1=0
giro_evasivo0=0
%definicion de la velocidad del robot movil, cuando mas se acerque al
%objetivo, se movera a menor velocidad, para una mejor aproximación final,
%adicional se agrega una condicional para detener el carrito cuando se
%llegue al objetivo con una tolerancia de +-0.01 equivalente a 1cm2
%adicional, se adjunta una condicional para que, en caso de llegar al
%objetivo, se permanezca en un angulo constante, el angulo en el que el
%robot haya llegado al objetivo, utilizamos la misma condicional que para
%la velocidad
limite_superior_llegada_x=objetivo_x+0.01
limite_inferior_llegada_x=objetivo_x-0.01
limite_superior_llegada_y=objetivo_y+0.01
limite_inferior_llegada_y=objetivo_y-0.01
if x_robot>=limite_inferior_llegada_x && x_robot<=limite_superior_llegada_x
    if y_robot>=limite_inferior_llegada_y && y_robot<=limite_superior_llegada_y
        V_cero=1
        W_cero=1
    else
        V_cero=0
        W_cero=0
    end
else
    V_cero=0
    W_cero=0
end
if V_cero==1
    Velocidad=0
else
    if dt<=0.15 && dt>=0.03
        Velocidad=0.75*dt
    else
        if dt<0.03 && dt>=0.01
            Velocidad=0.3*dt
        else
            %agregamos una condicion de lenta velocidad en caso de perfecta
            %simetria e inminente choque

            if d5<0.225 || d4<0.15 || d6<0.15 || d3<0.15 || d7<0.15
                Velocidad=0
                if d4>d6
                    giro_evasivo1=1
                    giro_evasivo0=0
                else
                    if d4<d6
                        giro_evasivo1=0
                        giro_evasivo0=1
                    else
                        giro_evasivo1=1
                        giro_evasivo0=0
                    end
```

```
                end
            else
                if d5<0.35
                   Velocidad=0.05
                else
                    Velocidad=0.08
                end
            end
        end
    end
end
%CALCULO DE LAS FUERZAS REPULSIVAS, SE DIVIDEN EN FUERZAS EN X Y FUERZAS EN
%Y
Fr1x=(Fcr/d1)*((Xobs1-x_robot)/d1)
Fr1y=(Fcr/d1)*((Yobs1-y_robot)/d1)
Fr2x=(Fcr/d2)*((Xobs2-x_robot)/d2)
Fr2y=(Fcr/d2)*((Yobs2-y_robot)/d2)
Fr3x=(Fcr/d3)*((Xobs3-x_robot)/d3)
Fr3y=(Fcr/d3)*((Yobs3-y_robot)/d3)
Fr4x=(Fcr/d4)*((Xobs4-x_robot)/d4)
Fr4y=(Fcr/d4)*((Yobs4-y_robot)/d4)
Fr5x=(Fcr/d5)*((Xobs5-x_robot)/d5)
Fr5y=(Fcr/d5)*((Yobs5-y_robot)/d5)
Fr6x=(Fcr/d6)*((Xobs6-x_robot)/d6)
Fr6y=(Fcr/d6)*((Yobs6-y_robot)/d6)
Fr7x=(Fcr/d7)*((Xobs7-x_robot)/d7)
Fr7y=(Fcr/d7)*((Yobs7-y_robot)/d7)
Fr8x=(Fcr/d8)*((Xobs8-x_robot)/d8)
Fr8y=(Fcr/d8)*((Yobs8-y_robot)/d8)
Fr9x=(Fcr/d9)*((Xobs9-x_robot)/d9)
Fr9y=(Fcr/d9)*((Yobs9-y_robot)/d9)
%CALCULO DE LA FUERZA DE ATRACCIÓN, IGUAL SE DIVIDE EN X & Y
Fax=Fca*((xt-x_robot)/dt)
Fay=Fca*((yt-y_robot)/dt)
%CALCULO DE LA FUERZA RESPULSIVA TOTAL EN X & Y
Frx=-(Fr1x+Fr2x+Fr3x+Fr4x+Fr5x+Fr6x+Fr7x+Fr8x+Fr9x)
Fry=-(Fr1y+Fr2y+Fr3y+Fr4y+Fr5y+Fr6y+Fr7y+Fr8y+Fr9y)
%CALCULO DE LE FUERZA TOTAL EN X & Y
Ftx=Frx+Fax
Fty=Fry+Fay
%CALCULO DEL ÁNGULO ALPHA DEL VECTOR CALCULADO, EL CUAL MODIFICARÁ
%LA DIRECCIÓN DEL ROBOT MÓVIL
%SE AGREGA UNA CONDICIONAL PARA SABER EN QUE CUADRANTE SE ENCUENTRA EL
%VECTOR. DEPENDIENDO DEL CUADRANTE, SE CALCULA EL ÁNGULO DESEADO
alpha_resultante=abs((atan(Fty/Ftx))*180/pi)
if Ftx>0 && Fty>0
    angulo_corregido=alpha_resultante
else
    if Ftx<0 && Fty<0
        angulo_corregido=180+alpha_resultante
    else
        if Ftx>0 && Fty<0
            angulo_corregido=360-alpha_resultante
        else
            if Ftx<0 && Fty>0
                angulo_corregido=180-alpha_resultante
            else
                angulo_corregido=alpha_resultante
            end
        end
    end
end
end
```

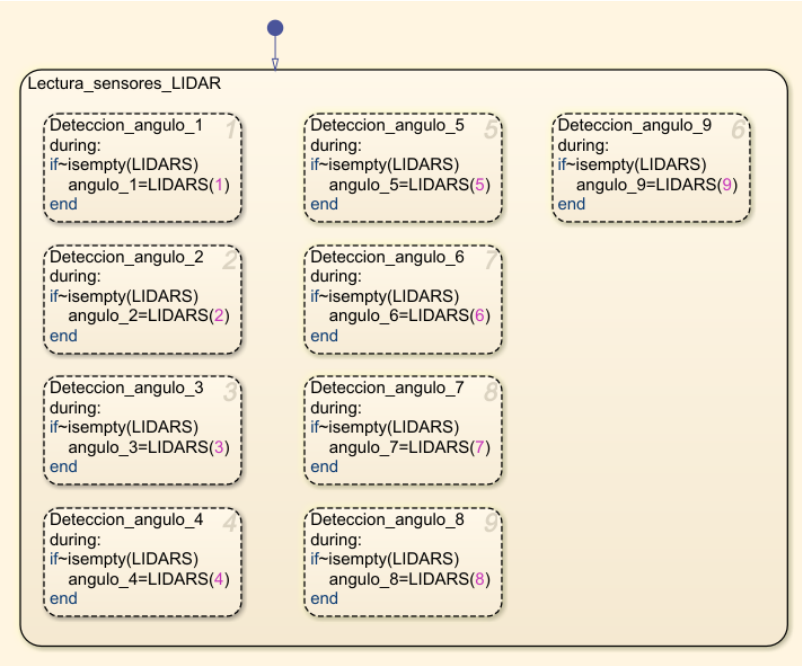
## ANEXO 86

## MÁQUINA DE ESTADOS PARA DETERMINACIÓN DE LA VELOCIDAD ANGULAR DESEADA PARA EL ROBOT MÓVIL BASÁNDOSE EN EL ÁNGUO DESEADO CALCULADO PARA EL ROBOT MÓVIL EN ALGORITMO VFF.




## ANEXO 87

## MÁQUINA DE ESTADOS PARA LECTURA DE LAS DISTANCIAS DETECTADAS POR CADA SENSOR LIDAR EN ALGORITMO VFF.



## ANEXO 88

## FUNCIÓN DE MATLAB PARA CALCULAR EL VALOR DE OCUPACIÓN DE CADA SECTOR BASÁNDOSE EN LA LECTURA DE SU PROPIO SENSOR JUNTO CON LA LECTURA DE LOS SECTORES VECINOS EN ALGORITMO VFH.

```
function hk_prima  =
CONVERSION_HISTOGRAMA_POLAR(d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15,d16,d1
7,d18,d19,d20,d21,d22,d23,d24)
d=[d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15,d16,d17,d18,d19,d20,d21,d22,d23
,d24]
hk=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
hk_prima=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
a=2
b=1
c=1
l=2
for i=1:+1:24
    hk(i)=(c^2)*(a-(b*d(i)))
end
for i=1:+1:24
    elementos=[0,0,0,0,0,0,0,0,0,0]
    for j=1:+1:((2*l)-1)
        elemento_posible=i-l+j
        if elemento_posible<=0
            elemento_posible=24+elemento_posible
        else
            if elemento_posible>=25
                elemento_posible=elemento_posible-24
            else
                elemento_posible=elemento_posible
            end
        end
        elementos(j)=hk(elemento_posible)
    end
    suma_total=0
    for j=1:+1:((2*l)-1)
        if j<=l
            resultado_actual=j*elementos(j)
        else
            resultado_actual=((2*l)-j)*elementos(j)
        end
        suma_total=suma_total+resultado_actual
    end
    hk_prima(i)=suma_total/((2*l)+1)
end
end
```

## ANEXO 89

## FUNCIÓN DE MATLAB PARA DISCRETIZACIÓN DE LOS SECTORES BASÁNDOSE EN UN UMBRAL PREDETERMINADO EN ALGORITMO VFH.

```
function h_discreto  =
DISCRETIZACION_SENAL(hk_prima,x_robot,y_robot,x_objetivo,y_objetivo)
h_discreto=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
sectores=hk_prima
distancia_objetivo=distancia_2_puntos(x_robot,y_robot,x_objetivo,y_objetivo)
if distancia_objetivo>0.5
    coeficiente=1.5
else
    coeficiente=5
end
for i=1:+1:24
    if sectores(i)<=coeficiente
        h_discreto(i)=1
    else
        h_discreto(i)=0
```

```
    end
end
end
```

## ANEXO 90

## FUNCIÓN DE MATLAB PARA IDENTIFICACIÓN DE SECTORES CONSIDERADOS COMO LIBRES EN ALGORITMO VFH.

```
function valles   = ASILAMIENTO_SECTORES_ACEPTADOS(h_discreto)
sector=h_discreto
valles=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
espacio_actual=1
for i=1:+1:24
    sector_actual=sector(i)
    if sector_actual==1
        valles(espacio_actual)=(i)
        espacio_actual=espacio_actual+1
    end
end
end
```

## ANEXO 91

## FUNCIÓN DE MATLAB PARA DIFERENCIACIÓN Y SEPARACIÓN DE SECTORES LIBRES EN VALLES EN ALGORITMO VFH.

```
function [valle1,valle2,valle3,valle4,valle5,valle6,valle7,valle8,valle9] =
DIFERENCIACION_VALLES(vector_valles,contador)
valle1=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle2=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle3=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle4=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle5=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle6=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle7=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle8=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
valle9=[0,0]
longitud=0
pos_valle1=1
pos_valle2=1
pos_valle3=1
pos_valle4=1
pos_valle5=1
pos_valle6=1
pos_valle7=1
pos_valle8=1
memoria_valle_activa=0
valle_actual=1
%
memoria_24_valle1=0
memoria_24_valle2=0
memoria_24_valle3=0
memoria_24_valle4=0
memoria_24_valle5=0
memoria_24_valle6=0
memoria_24_valle7=0
memoria_24_valle8=0
%
ultimo_sector=0
primer_sector=0
%
for i=1:+1:24
    sector_comprobado_actual=vector_valles(i)
    if sector_comprobado_actual~=0
        longitud=longitud+1
    end
end
for i=1:+1:longitud
    elemento_actual=i
    elemento_superior=i+1
    sector_actual=vector_valles(elemento_actual)
    if elemento_actual==longitud
        if memoria_valle_activa==1
```

```
            if valle_actual==1
                valle1(pos_valle1)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==2
                valle2(pos_valle2)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==3
                valle3(pos_valle3)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==4
                valle4(pos_valle4)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==5
                valle5(pos_valle5)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==6
                valle6(pos_valle6)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==7
                valle7(pos_valle7)=sector_actual
                memoria_valle_activa=0
            end
            if valle_actual==8
                valle8(pos_valle8)=sector_actual
                memoria_valle_activa=0
            end
            valle_actual=valle_actual+1
        end
    else
        sector_superior=vector_valles(elemento_superior)
        if sector_superior==(sector_actual+1)
            if valle_actual==1
                valle1(pos_valle1)=sector_actual
                pos_valle1=pos_valle1+1
                memoria_valle_activa=1
            end
            if valle_actual==2
                valle2(pos_valle2)=sector_actual
                pos_valle2=pos_valle2+1
                memoria_valle_activa=1
            end
            if valle_actual==3
                valle3(pos_valle3)=sector_actual
                pos_valle3=pos_valle3+1
                memoria_valle_activa=1
            end
            if valle_actual==4
                valle4(pos_valle4)=sector_actual
                pos_valle4=pos_valle4+1
                memoria_valle_activa=1
            end
            if valle_actual==5
                valle5(pos_valle5)=sector_actual
                pos_valle5=pos_valle5+1
                memoria_valle_activa=1
            end
            if valle_actual==6
                valle6(pos_valle6)=sector_actual
                pos_valle6=pos_valle6+1
                memoria_valle_activa=1
            end
            if valle_actual==7
                valle7(pos_valle7)=sector_actual
                pos_valle7=pos_valle7+1
                memoria_valle_activa=1
            end
            if valle_actual==8
                valle8(pos_valle8)=sector_actual
                pos_valle8=pos_valle8+1
                memoria_valle_activa=1
            end
```

```
        else
            if memoria_valle_activa==1
                if valle_actual==1
                    valle1(pos_valle1)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==2
                    valle2(pos_valle2)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==3
                    valle3(pos_valle3)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==4
                    valle4(pos_valle4)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==5
                    valle5(pos_valle5)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==6
                    valle6(pos_valle6)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==7
                    valle7(pos_valle7)=sector_actual
                    memoria_valle_activa=0
                end
                if valle_actual==8
                    valle8(pos_valle8)=sector_actual
                    memoria_valle_activa=0
                end
                valle_actual=valle_actual+1
            end
        end
    end
end
% aqui verificamos si es un área totalmente libre de obstaculos, en cuyo caso el
valle 1 está lleno
if valle1(24)==24
    memoria_area_libre=1
else
    memoria_area_libre=0
end
%
if contador==1
    if memoria_area_libre==0
        memoria_1_loc_valle=0
        memoria_24_loc_valle=0
        primer_sector=vector_valles(1)
        ultimo_sector=vector_valles(longitud)
        if ultimo_sector==24 && primer_sector==1
            if valle1(1)==1
                memoria_1_loc_valle=1
            end
            for i=1:+1:8
                valle_actual_analizado=i
                %
                if valle_actual_analizado==1
                    for j=1:+1:24
                        pos_actual=valle1(j)
                        if pos_actual==24
                            memoria_24_valle1=1
                            memoria_24_loc_valle=1
                        end
                    end
                end
                if valle_actual_analizado==2
                    for j=1:+1:24
                        pos_actual=valle2(j)
                        if pos_actual==24
                            memoria_24_valle2=1
                            memoria_24_loc_valle=1
                        end
                    end
```

```
    end
    if valle_actual_analizado==3
        for j=1:+1:24
            pos_actual=valle3(j)
            if pos_actual==24
                memoria_24_valle3=1
                memoria_24_loc_valle=1
            end
        end
    end
    if valle_actual_analizado==4
        for j=1:+1:24
            pos_actual=valle4(j)
            if pos_actual==24
                memoria_24_valle4=1
                memoria_24_loc_valle=1
            end
        end
    end
    if valle_actual_analizado==5
        for j=1:+1:24
            pos_actual=valle5(j)
            if pos_actual==24
                memoria_24_valle5=1
                memoria_24_loc_valle=1
            end
        end
    end
    if valle_actual_analizado==6
        for j=1:+1:24
            pos_actual=valle6(j)
            if pos_actual==24
                memoria_24_valle6=1
                memoria_24_loc_valle=1
            end
        end
    end
    if valle_actual_analizado==7
        for j=1:+1:24
            pos_actual=valle7(j)
            if pos_actual==24
                memoria_24_valle7=1
                memoria_24_loc_valle=1
            end
        end
    end
    if valle_actual_analizado==8
        for j=1:+1:24
            pos_actual=valle8(j)
            if pos_actual==24
                memoria_24_valle8=1
                memoria_24_loc_valle=1
            end
        end
    end
%
end
% caso 1, ambos no están en un valle
if memoria_1_loc_valle==0 && memoria_24_loc_valle==0
    valle9=[24 1]
end
% caso 2, solo el 1 está en un valle, se mueven de posicion todos
% los del valle 1 y se coloca al principio el 24
if memoria_1_loc_valle==1 && memoria_24_loc_valle==0
    for i=24:-1:2
        valle1(i)=valle1(i-1)
    end
    valle1(1)=24
end
% caso 3 solo el 24 está en un valle, en este caso se busca en que
% valle se encuentra el 24 y se lo agrega al 1 al final de este
% valle
if memoria_1_loc_valle==0 && memoria_24_loc_valle==1
    %
    memoria_bloqueo_repeticion_1=1
    if memoria_24_valle1==1
        for i=1+1:24
```

```
            pos_analizada=valle1(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle1(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    if memoria_24_valle2==1
        for i=1+1:24
            pos_analizada=valle2(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle2(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    if memoria_24_valle3==1
        for i=1+1:24
            pos_analizada=valle3(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle3(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    if memoria_24_valle4==1
        for i=1+1:24
            pos_analizada=valle4(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle4(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    if memoria_24_valle5==1
        for i=1+1:24
            pos_analizada=valle5(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle5(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    if memoria_24_valle6==1
        for i=1+1:24
            pos_analizada=valle6(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle6(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    if memoria_24_valle7==1
        for i=1+1:24
            pos_analizada=valle7(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle7(i)=1
                i=24
            end
        end
    end
    if memoria_24_valle8==1
        for i=1+1:24
            pos_analizada=valle8(i)
            if pos_analizada==0 && memoria_bloqueo_repeticion_1==1
                valle8(i)=1
                memoria_bloqueo_repeticion_1=0
            end
        end
    end
    %
end
% caso 4 los do se encuentran en valles distintos se tendra que
% buscar en que valle se encuentra el 24, luego verificar la
% longitud del valle donde se encuentra el 24 para recorrer esos
% espacios en el valle1
if memoria_1_loc_valle==1 && memoria_24_loc_valle==1
```

```
%
longitud_valle24=0
if memoria_24_valle2==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle2(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
    end
end
if memoria_24_valle3==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle3(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
    end
end
if memoria_24_valle4==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle4(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
    end
end
if memoria_24_valle5==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle5(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
    end
end
if memoria_24_valle6==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle6(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
    end
end
if memoria_24_valle7==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle7(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
    end
end
if memoria_24_valle8==1
    longitud_valle24=0
    for i=1:+1:24
        pos_analizada=valle8(i)
        if pos_analizada~=0
            longitud_valle24=longitud_valle24+1
        else
            i=24
        end
```

```
                    end
                end
                %
                for i=1:+1:longitud_valle24
                    if (i+longitud_valle24)<=24
                        valle1(i+longitud_valle24)=valle1(i)
                    end
                end
                %
                if memoria_24_valle2==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle2(i)
                    end
                    valle2=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                if memoria_24_valle3==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle3(i)
                    end
                    valle3=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                if memoria_24_valle4==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle4(i)
                    end
                    valle4=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                if memoria_24_valle5==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle5(i)
                    end
                    valle5=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                if memoria_24_valle6==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle6(i)
                    end
                    valle6=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                if memoria_24_valle7==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle7(i)
                    end
                    valle7=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                if memoria_24_valle8==1
                    for i=1:+1:longitud_valle24
                        valle1(i)=valle8(i)
                    end
                    valle8=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                end
                %
            end
        end
    end
    end
    end
```

## ANEXO 92

## FUNCIÓN DE MATLAB PARA IDENTIFICAR EL INICIO, FINAL Y EXTENSIÓN DE CADA VALLE IDENTIFICADO.

```
function  [memorias_valles,inicio_valles,final_valles,extension_valles]=
IDENTIFICACION_INICIO_FIN_VALLES(valle1,valle2,valle3,valle4,valle5,valle6,valle7,valle8,valle9)
memorias_valles=[0,0,0,0,0,0,0,0,0]
inicio_valles=[0,0,0,0,0,0,0,0,0]
final_valles=[0,0,0,0,0,0,0,0,0]
extension_valles=[0,0,0,0,0,0,0,0,0]
longitudes_valles=[0,0,0,0,0,0,0,0,0]
elementos_finales=[0,0,0,0,0,0,0,0,0]
%
for i=1:+1:9
    valle_actual=i
    if valle_actual==1
        if valle1(1)~=0
            memorias_valles(valle_actual)=1
            inicio_valles(valle_actual)=(valle1(1)-1)*15
```

```
        for j=1:+1:24
            if valle1(j)~=0
                longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
            end
        end
        elementos_finales(valle_actual)=valle1(1)+(longitudes_valles(valle_actual)-1)
        if elementos_finales(valle_actual)>=25
            elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
        end
        final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
        extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
    end
end
if valle_actual==2
    if valle2(1)~=0
        memorias_valles(valle_actual)=1
        inicio_valles(valle_actual)=(valle2(1)-1)*15
        for j=1:+1:24
            if valle2(j)~=0
                longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
            end
        end
        elementos_finales(valle_actual)=valle2(1)+(longitudes_valles(valle_actual)-1)
        if elementos_finales(valle_actual)>=25
            elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
        end
        final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
        extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
    end
end
if valle_actual==3
    if valle3(1)~=0
        memorias_valles(valle_actual)=1
        inicio_valles(valle_actual)=(valle3(1)-1)*15
        for j=1:+1:24
            if valle3(j)~=0
                longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
            end
        end
        elementos_finales(valle_actual)=valle3(1)+(longitudes_valles(valle_actual)-1)
        if elementos_finales(valle_actual)>=25
            elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
        end
        final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
        extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
    end
end
if valle_actual==4
    if valle4(1)~=0
        memorias_valles(valle_actual)=1
        inicio_valles(valle_actual)=(valle4(1)-1)*15
        for j=1:+1:24
            if valle4(j)~=0
                longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
            end
        end
        elementos_finales(valle_actual)=valle4(1)+(longitudes_valles(valle_actual)-1)
        if elementos_finales(valle_actual)>=25
            elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
        end
        final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
        extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
    end
end
if valle_actual==5
    if valle5(1)~=0
        memorias_valles(valle_actual)=1
        inicio_valles(valle_actual)=(valle5(1)-1)*15
        for j=1:+1:24
            if valle5(j)~=0
                longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
            end
        end
        elementos_finales(valle_actual)=valle5(1)+(longitudes_valles(valle_actual)-1)
        if elementos_finales(valle_actual)>=25
            elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
        end
        final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
        extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
    end
end
if valle_actual==6
    if valle6(1)~=0
        memorias_valles(valle_actual)=1
        inicio_valles(valle_actual)=(valle6(1)-1)*15
        for j=1:+1:24
            if valle6(j)~=0
                longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
```

```
                end
            end
            elementos_finales(valle_actual)=valle6(1)+(longitudes_valles(valle_actual)-1)
            if elementos_finales(valle_actual)>=25
                elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
            end
            final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
            extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
        end
    end
    if valle_actual==7
        if valle7(1)~=0
            memorias_valles(valle_actual)=1
            inicio_valles(valle_actual)=(valle7(1)-1)*15
            for j=1:+1:24
                if valle7(j)~=0
                    longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
                end
            end
            elementos_finales(valle_actual)=valle7(1)+(longitudes_valles(valle_actual)-1)
            if elementos_finales(valle_actual)>=25
                elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
            end
            final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
            extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
        end
    end
    if valle_actual==8
        if valle8(1)~=0
            memorias_valles(valle_actual)=1
            inicio_valles(valle_actual)=(valle8(1)-1)*15
            for j=1:+1:24
                if valle8(j)~=0
                    longitudes_valles(valle_actual)=longitudes_valles(valle_actual)+1
                end
            end
            elementos_finales(valle_actual)=valle8(1)+(longitudes_valles(valle_actual)-1)
            if elementos_finales(valle_actual)>=25
                elementos_finales(valle_actual)=elementos_finales(valle_actual)-24
            end
            final_valles(valle_actual)=(elementos_finales(valle_actual)-1)*15
            extension_valles(valle_actual)=(longitudes_valles(valle_actual)-1)*15
        end
    end
    %
    if valle_actual==9
        if valle9(1)~=0
            inicio_valles(valle_actual)=345
            final_valles(valle_actual)=0
            longitudes_valles(valle_actual)=2
            extension_valles(valle_actual)=15
        end
    end
end
end
```

## ANEXO 93

## FUNCIÓN DE MATLAB PARA IDENTIFICAR EN QUE DIRECCIÓN SE ENCUENTRA EL OBJETIVO CON RESPECTO AL SISTEMA DE REFERENCIA GLOBAL Y CON RESPECTO A LA PERSPECTIVA DEL ROBOT MÓVIL EN ALGORITMO VFH.

```
function [angulo_objetivo,angulo_perspectivo] =
DIRECCIONAMIENTO_ANGULAR(objetivo_x,objetivo_y,pos_x,pos_y,angulo_robot)
% se debe saber en que angulo se encuentra el objetivo, en este caso hay
% cuatro opciones distintas, correspondientes a los 4 cuadrantes posibles
% de un plano cartesiano, tomando como punto 0 el centro del robot
% caso 1, cuadrante 1
%
alpha=0
if objetivo_x>pos_x && objetivo_y>pos_y
    b=objetivo_y-pos_y
    a=objetivo_x-pos_x
    alpha=(atan(b/a))*(180/pi)
end
%caso 2, cuadrante 2
if objetivo_x<pos_x && objetivo_y>pos_y
    b=objetivo_y-pos_y
    a=abs(objetivo_x-pos_x)
    alpha=180-((atan(b/a))*(180/pi))
```

```
    end
    %caso 3, cuadrante 3
    if objetivo_x<pos_x && objetivo_y<pos_y
        b=abs(objetivo_y-pos_y)
        a=abs(objetivo_x-pos_x)
        alpha=180+((atan(b/a))*(180/pi))
    end
    %caso 4, cuadrante 4
    if objetivo_x>pos_x && objetivo_y<pos_y
        b=abs(objetivo_y-pos_y)
        a=objetivo_x-pos_x
        alpha=360-((atan(b/a))*(180/pi))
    end
    if objetivo_x==pos_x && objetivo_y>pos_y
        alpha=90
    end
    if objetivo_x==pos_x && objetivo_y<pos_y
        alpha=270
    end
    if objetivo_x>pos_x && objetivo_y==pos_y
        alpha=0
    end
    if objetivo_x<pos_x && objetivo_y==pos_y
        alpha=180
    end
    if objetivo_x==pos_x && objetivo_y==pos_y
        alpha=0
    end
    angulo_objetivo=alpha
    angulo_perspectivo=angulo_objetivo-angulo_robot
    if angulo_perspectivo<0
        angulo_perspectivo=360-abs(angulo_perspectivo)
    end
    end
```

## ANEXO 94

## FUNCIÓN DE MATLAB PARA CALCULAR EL ÁNGULO DESEADO PARA EL ROBOT MÓVIL EN ALGORITMO VFH.

```
function [angulo_deseado,memoria_valle] =
CALCULO_ANGULO_DESEADO(angulo_robot,angulo_perspectivo,angulo_objetivo,memoria_valles,extension_v
alles,inicio_valles,final_valles)
angulo_deseado=0
memoria_valle=0
valoracion_valle=0
distancia_min=9999
distancia_inicio=0
distancia_final=0
memoria_inicio=0
memoria_final=0
valle_elegido=1
angulo_deseado_perspectivo=0
extension_maxima=90
for i=1:+1:9
    valle_analizado=memoria_valles(i)
    if valle_analizado==1
        valle_actual=i
        inicio=inicio_valles(valle_actual)
        final=final_valles(valle_actual)
        if final<inicio
            if angulo_perspectivo>=inicio && angulo_perspectivo<360
                memoria_valle=1
                angulo_deseado=angulo_objetivo
            else
                if angulo_perspectivo>=0 && angulo_perspectivo<=final
                    memoria_valle=1
                    angulo_deseado=angulo_objetivo
                end
            end
        else
            if angulo_perspectivo>=inicio && angulo_perspectivo<=final
                memoria_valle=1
                angulo_deseado=angulo_objetivo
```

```
            end
        end
    end
end
%
if memoria_valle==0
    for i=1:+1:9
        valle_actual=i
        valoracion_valle=memoria_valles(valle_actual)
        if valoracion_valle==1
            inicio=inicio_valles(valle_actual)
            final=final_valles(valle_actual)
            distancia_inicio=abs(angulo_perspectivo-inicio)
            distancia_final=abs(angulo_perspectivo-final)
            if distancia_inicio>180
                distancia_inicio=360-distancia_inicio
            end
            if distancia_final>180
                distancia_final=360-distancia_final
            end
            if distancia_inicio<distancia_min
                distancia_min=distancia_inicio
                valle_elegido=valle_actual
                memoria_inicio=1
                memoria_final=0
            end
            if distancia_final<distancia_min
                distancia_min=distancia_final
                valle_elegido=valle_actual
                memoria_inicio=0
                memoria_final=1
            end
        end
    end
    %
    extension_valle_elegido=extension_valles(valle_elegido)
    if extension_valle_elegido>=extension_maxima
        if memoria_inicio==1
            inicio_valle_corregido=inicio_valles(valle_elegido)
            angulo_deseado_perspectivo=inicio_valle_corregido+(extension_maxima/2)
            if angulo_deseado_perspectivo>=360
                angulo_deseado_perspectivo=angulo_deseado_perspectivo-360
            end
        end
        if memoria_final==1
            final_valle_corregido=final_valles(valle_elegido)
            angulo_deseado_perspectivo=final_valle_corregido-(extension_maxima/2)
            if angulo_deseado_perspectivo<0
                angulo_deseado_perspectivo=360+angulo_deseado_perspectivo
            end
        end
    else
        inicio_valle_corregido=inicio_valles(valle_elegido)
        angulo_deseado_perspectivo=inicio_valle_corregido+((extension_valle_elegido/2)-1)
        if angulo_deseado_perspectivo>=360
            angulo_deseado_perspectivo=angulo_deseado_perspectivo-360
        end
    end
    angulo_deseado=angulo_deseado_perspectivo+angulo_robot
    if angulo_deseado>=360
        angulo_deseado=angulo_deseado-360
    end
end
end
```

## ANEXO 95

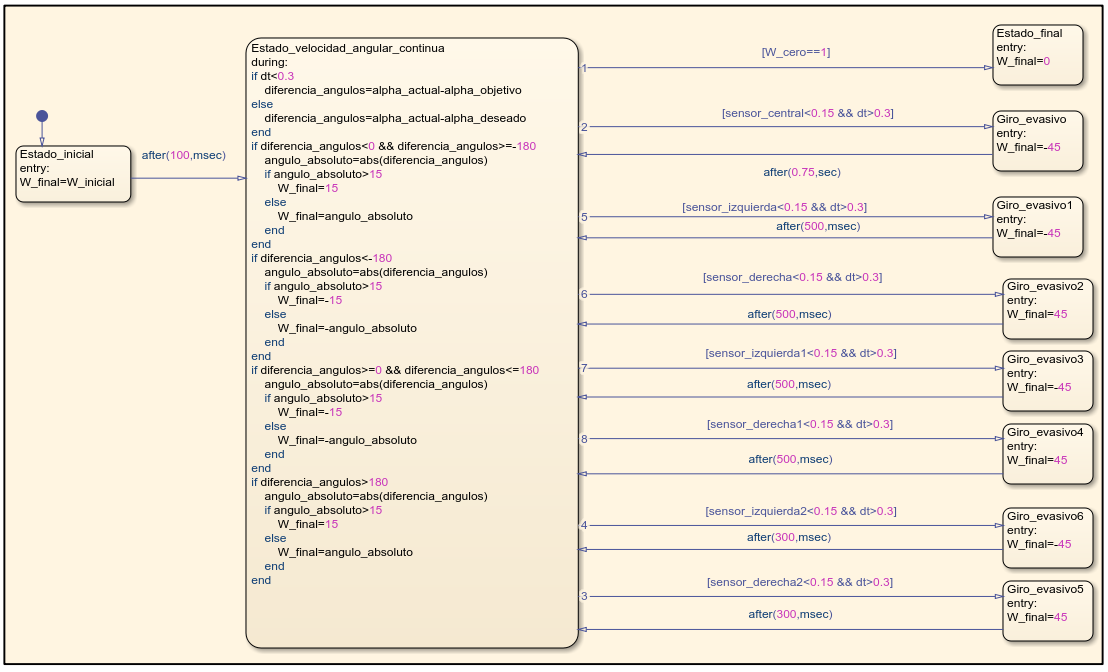
## FUNCIÓN DE MATLAB PARA EL CÁLCULO DE LA VELOCIDAD EN ALGORITMO VFH.

```
function [Velocidad,W_cero,dt] =
CALCULO_VELOCIDAD(x_robot,y_robot,objetivo_x,objetivo_y,sensor_central)
dt=distancia_2_puntos(x_robot,y_robot,objetivo_x,objetivo_y)
limite_superior_llegada_x=objetivo_x+0.005
limite_inferior_llegada_x=objetivo_x-0.005
limite_superior_llegada_y=objetivo_y+0.005
```

```
limite_inferior_llegada_y=objetivo_y-0.005
if x_robot>limite_inferior_llegada_x && x_robot<limite_superior_llegada_x
    if y_robot>limite_inferior_llegada_y && y_robot<limite_superior_llegada_y
        V_cero=1
        W_cero=1
    else
        V_cero=0
        W_cero=0
    end
else
    V_cero=0
    W_cero=0
end
if V_cero==1
    Velocidad=0
else
    if sensor_central<0.12 && dt>0.3
        Velocidad=0
    else
        if dt<=0.15 && dt>=0.025
            Velocidad=0.5*dt
        else
            if dt<0.0 && dt>=0.005
                Velocidad=0.3*dt
            else
                Velocidad=0.08
            end
        end
    end
end
end
```

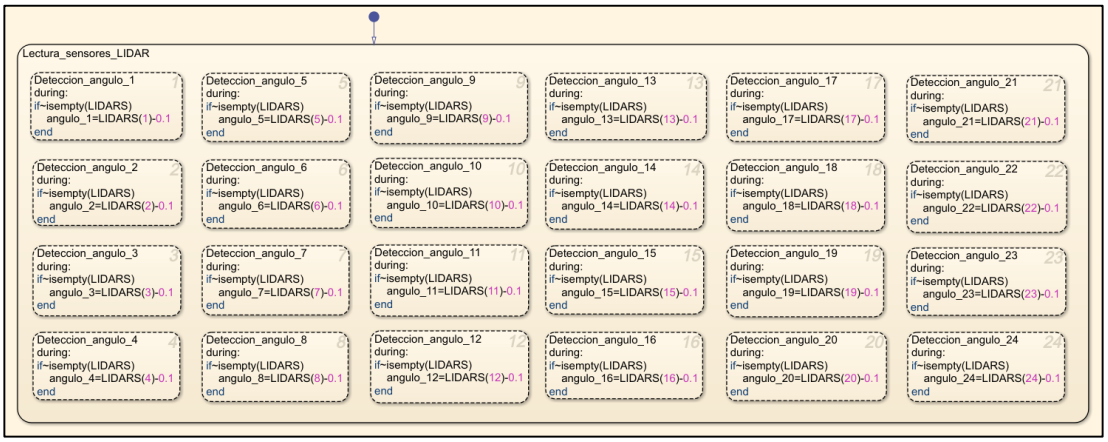
## ANEXO 96

## MÁQUINA DE ESTADOS PARA CÁLCULO DE LA VELOCIDAD ANGULAR DESEADA PARA EL ROBOT MÓVIL BASÁNDOSE EN EL ÁNGULO DESEADO PARA EL ROBOT MÓVIL EN ALGORITMO VFH.



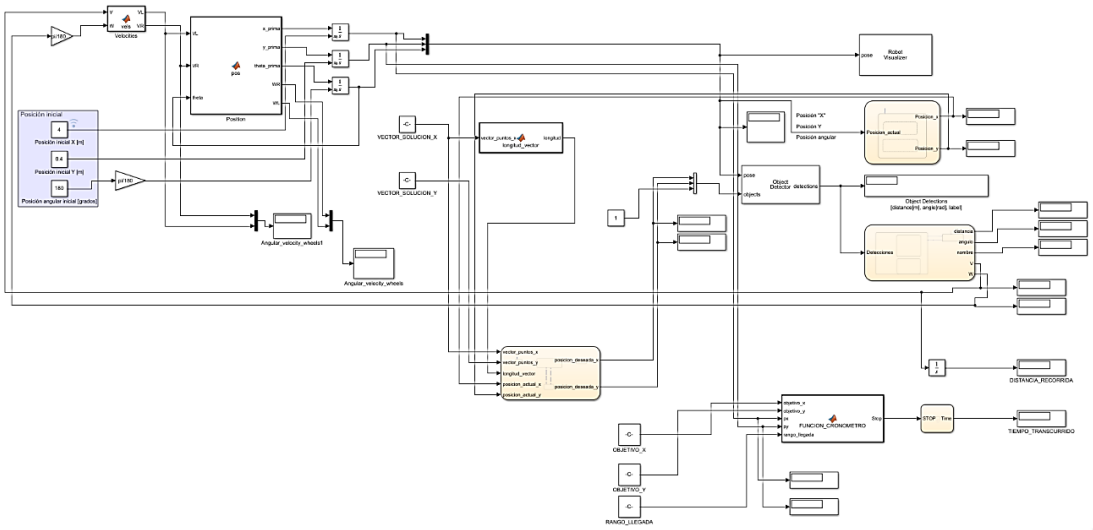
## ANEXO 97

## MÁQUINA DE ESTADOS PARA LECTURA DE SENSORES LIDAR EN ALGORITMO VFH.



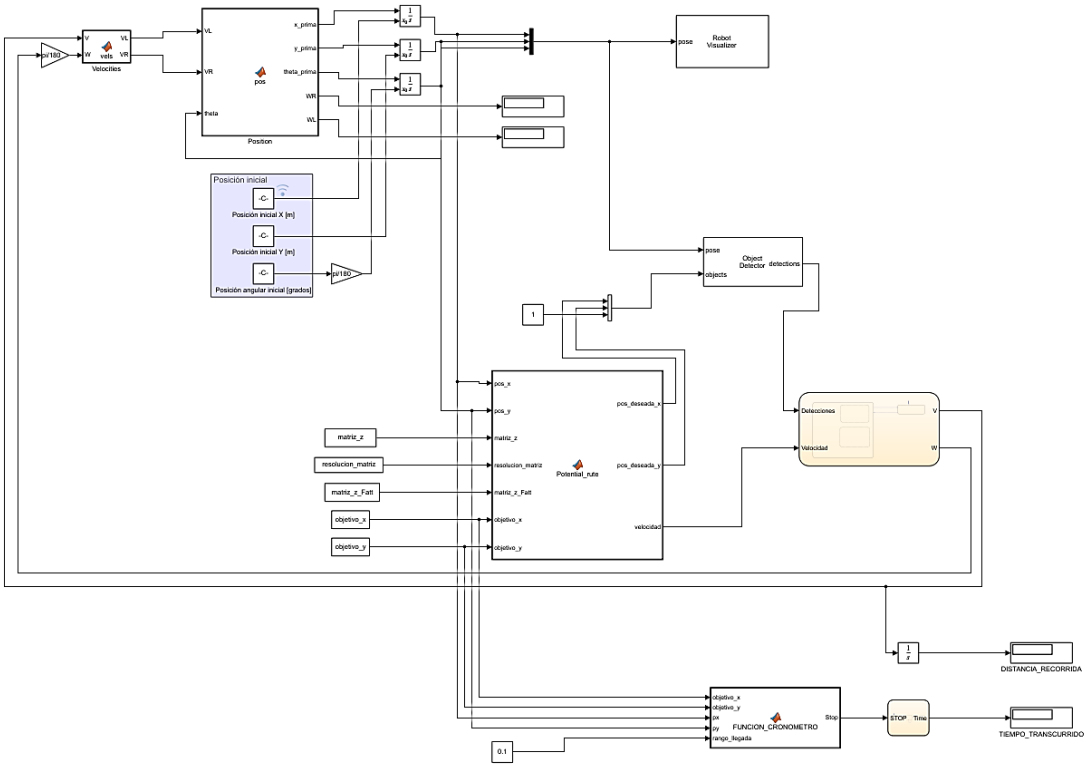

## ANEXO 98

## SISTEMA DE BLOQUES DE SIMULINK PARA SIMULACIÓN DE ALGORITMOS RRT & RRT*.



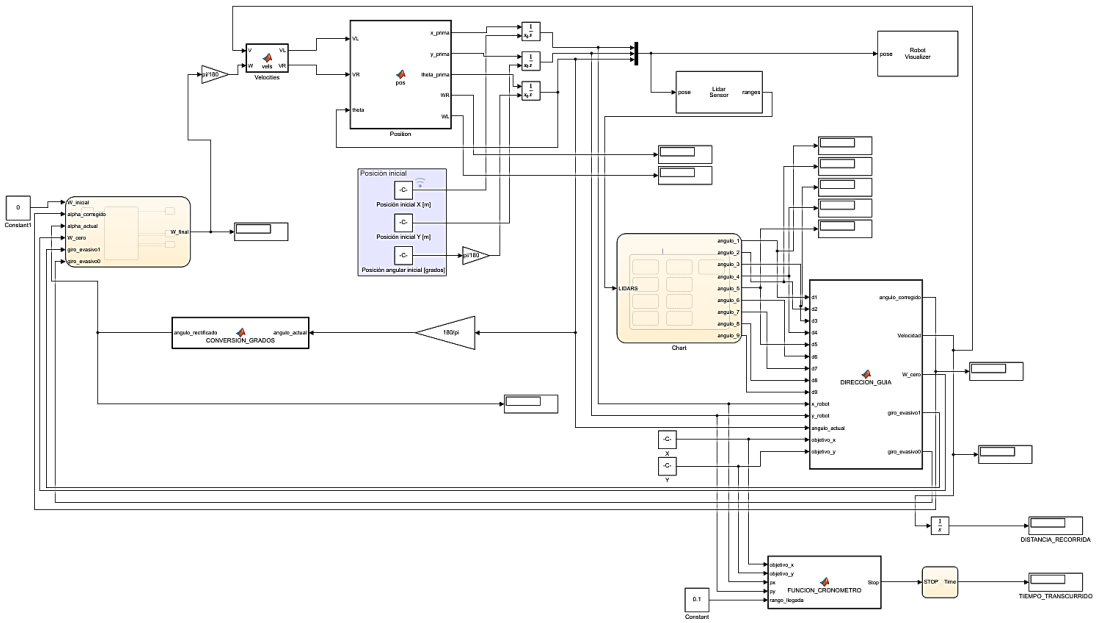
## ANEXO 99

## SISTEMA DE BLOQUES DE SIMULINK PARA SIMULACIÓN DE ALGORITMO PFPP.




## ANEXO 100

## SISTEMA DE BLOQUES PARA IMPLEMENTACIÓN Y SIMULACIÓN DE ALGORITMO VFF.



## ANEXO 101

## SISTEMA DE BLOQUES PARA IMPLEMENTACIÓN Y SIMULACIÓN DE ALGORITMO VFH.