UNIVERSIDAD MILITAR NUEVA GRANADA
FACULTAD DE INGENIERÍA
PROGRAMA EN INGENIERÍA MECATRÓNICA

SIMULACIÓN E IMPLEMENTACIÓN DE ALGORITMOS PARA GENERACIÓN DE TRAYECTORIAS EN EL SISTEMA DE ROBÓTICA MODULAR MECABOT.

Autor;
DANIEL ENRIQUE LÓPEZ NIÑO

Director:
ING. RICARDO ANDRÉS CASTILLO ESTEPA, PhD

Bogotá D.C, 2019

SIMULACIÓN E IMPLEMENTACIÓN DE ALGORITMOS PARA GENERACIÓN DE TRAYECTORIAS EN EL SISTEMA DE ROBÓTICA MODULAR MECABOT.

DANIEL ENRIQUE LÓPEZ NIÑO

Tesis de Pregrado para optar al título de Ingeniero Mecatrónico

Director: Ing. Ricardo Andrés Castillo Estepa, PhD

Universidad Militar Nueva Granada
Facultad de Ingeniería
Programa de Ingeniería Mecatrónica
Bogotá D.C
2019

Bogotá D.C (2019)

Nota de aceptación

_____
_____
_____
_____
_____
_____

_____
Firma del Presidente del Jurado

_____
Firma de Jurado

_____
Firma de Jurado

AGRADECIMIENTOS

A mi familia, por apoyarme, guiarme e instruir en mí que la responsabilidad es lo primero.

A mis compañeros, por hacer divertidos los momentos difíciles y por darme su apoyo.

A la Universidad Militar Nueva Granada, por disponer de los espacios y recursos necesarios para cumplir esta meta.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

El sistema robótico modular MECABOT, desarrollado por el grupo de investigación DAVINCI de la Universidad Militar Nueva Granada se encuentra actualmente en su quinta versión. Con ella, se pueden obtener arquitecturas de serpiente, oruga, rueda, hexápodo, y recientemente, se logró el desarrollo de una arquitectura cuadrúpeda, sobre la cual se realizará esta tesis. La locomoción de cada arquitectura es determinada por secuencias de acciones previamente definidas. El presente trabajo de grado se enfoca en la búsqueda, selección, programación e implementación de un algoritmo para generación de trayectorias que permita el desplazamiento del robot MECABOT 5.0 a través de terrenos con distintas configuraciones de obstáculos.

Se realiza un análisis de los métodos clásicos de planificación de trayectorias más desarrollados según [39] para una posterior comparación que logre identificar aquel método más apropiado para la aplicación presente según indicadores como seguridad, tiempo de ejecución, eficiencia, etc.

Al finalizar, el modelo virtual del robot es programado y probado en terrenos estructurados con varias configuraciones conocidas de los obstáculos. Su desempeño será medido a través del tiempo de ejecución de la trayectoria.

ABSTRACT

The modular robotic system MECABOT, developed by the DAVINCI research group from Universidad Militar Nueva Granada is currently on its fifth version. With it, several architectures can be achieved, like the serpent, caterpillar, wheel, hexapod and recently the development of a quadruped architecture was accomplished. This work will be based on the last one mentioned. The locomotion of each architecture is determined by sequences of previously defined actions. The current document is focused on the search, selection, programming, and implementation of an algorithm for trajectory generation that allows the displacement of the MECABOT 5.0 robot through terrains with various obstacle configurations.

An analysis is made of the more developed classic methods for trajectory planning according to [39] for a later comparison that manages to identify the most appropriate method for the present application according to indicators such as security, execution time, efficiency, etc.

In the end, the virtual model of the robot is programmed and tested on structured terrains with various known obstacle configurations. Its performance will be measured through the trajectory's execution time.

CAPÍTULO 1
INTRODUCCIÓN

Alrededor del año 380 A.C. Platón nos presenta en su dialogo socrático *Politeia* la siguiente afirmación: “La necesidad es la verdadera creadora, quien es la madre de nuestra invención” [1].

Siglos después, ese pensamiento se ve reflejado en el hecho de que la humanidad ha encontrado formas de lograr sus objetivos dado que la necesidad de ello es imperativa. [2] Ejemplo de esto es la Segunda Guerra Mundial, en donde el ejército alemán desarrolló un sin fin de mecanismos y artefactos robóticos para lograr su cometido.

Uno de estos robots es el *Goliath,* un pequeño vehículo con tracción por orugas, teledirigido y guiado por cable cuyo propósito era la demolición de fortificaciones, limpieza de campos minados o el ataque a otros vehículos. Aunque no tuvo bastante éxito en su tiempo, fundo las bases para el desarrollo de avances tecnológicos en vehículos a control remoto después del conflicto. [3]

Actualmente, existe un sinfín de aplicaciones para los robots teledirigidos, tal como lo es la exploración o la búsqueda y rescate. Los robots exploradores se diseñan con el propósito de, según lo indica su nombre, explorar y conocer terrenos desconocidos de forma que no se ponga en riesgo la vida de un ser humano, o acceder a lugares inalcanzables para él. Su funcionamiento se basa en la toma de imágenes que posteriormente se analizan y se toman decisiones a partir de los resultados obtenidos, algunos mecanismos lo hacen de forma autónoma puesto que cuentan con sistemas integrados de inteligencia artificial.

La presente tesis se enfoca en el mapeo, la planificación y el seguimiento de rutas para un sistema robótico modular de exploración, para ello se utilizará como fundamentos los distintos acercamientos clásicos a la resolución de los problemas de navegación que se presentarán más adelante. Para implementar la solución, se utilizará la quinta versión del sistema robótico modular MECABOT en su arquitectura cuadrúpeda desarrollado por el grupo de investigación DAVINCI de la Universidad Militar Nueva Granada.

Este documento está dividido en siete secciones, la primera hace referencia a la introducción, antecedentes, objetivos y las delimitaciones de la tesis. La segunda presenta el marco referencial entorno a la robótica modular y robots de exploración haciendo énfasis en los mecanismos de navegación utilizados.

La tercera sección muestra la definición, comparación y evaluación de los distintos métodos de navegación para la posterior selección del más adecuado según algunos parámetros definidos. La sección cuatro y cinco corresponde a la programación del algoritmo y la simulación de éste respectivamente. La sexta sección presenta la implementación del algoritmo en el sistema físico con los resultados de las pruebas físicas realizadas. Por último, la sección siete expone conclusiones y recomendaciones para trabajos futuros.

1.1 Planteamiento del problema

La robótica tele operada busca la unión de distintos sistemas robóticos que se puedan implementar en situaciones en donde no es conveniente involucrar a una persona debido al alto riesgo al que se pueden exponer; por esto, busca el desarrollo de herramientas y equipos que puedan ser manejados a distancia. [4]

Debido a su gran utilidad, se han implementado en varios campos en donde la principal ventaja es la preservación de la integridad física del usuario, tal como lo es el sector nuclear (mantenimiento de reactores), químico (manejo de sustancias), militar (detección, manipulación y desmantelamiento de cargas explosivas, desminado humanitario), exploración (espacial, inspección de tuberías, zonas de desastre), médico (tele cirugía), minero (excavaciones o manejo de explosivos) entre otros. [5]

El grupo DAVINCI, en pro de la robótica de exploración y de búsqueda y rescate ha diseñado y fabricado el robot modular MECABOT, el cual ha sido sometido a varias mejoras y actualizaciones estando actualmente en su quinta versión. Sobre este sistema solo se ha explorado su capacidad de formar distintas arquitecturas, sin embargo, poco se ha tratado de su mecanismo de navegación.

En el presente, todas las arquitecturas son controladas remotamente con secuencias de movimientos previamente definidos, la inclusión de un mecanismo de planeación y seguimiento de rutas aumenta considerablemente la autonomía del sistema y disminuiría el consumo de energía puesto que busca el camino más corto.

A nivel general, un algoritmo para la generación de trayectorias presenta un mayor acercamiento al objetivo final del sistema, de igual forma presenta una mayor flexibilidad en su desempeño puesto que evalúa cada posibilidad. De esta forma, la pregunta que se pretende responder con el desarrollo de esta tesis es: ¿Cómo mejora la navegación del MECABOT 5.0 la implementación física de un algoritmo para generación de trayectorias a comparación de un manejo manual?

1.2 Objetivo General

Desarrollar, simular e implementar un algoritmo para planeación de trayectorias en el sistema robótico modular MECABOT configurado en arquitectura cuadrúpeda, de forma que se permita la evasión de obstáculos en el desplazamiento de un punto a otro sobre un terreno conocido.

1.3 Objetivos Específicos

- Investigar distintos métodos de planeación de trayectorias específicos de la robótica modular, en caso de que sean escasos, se evaluarán algunos métodos clásicos de planificación de trayectorias en la robótica móvil, analizando indicadores de desempeño como completitud, complejidad, seguridad y estabilidad.

- Desarrollar un algoritmo que permita generar las trayectorias para el sistema robótico modular y posteriormente evaluar su desempeño utilizando el entorno de simulación Webots.

- Evaluar el funcionamiento del sistema modular virtual una vez implementado el algoritmo de planeación de trayectorias, al desplazarse sobre distintas configuraciones de obstáculos (forma, cantidad y ubicación) en un terreno, evaluando el tiempo de ejecución.

- Desarrollar una interfaz de usuario que permita la comunicación con los dispositivos físicos, la cual pueda ser utilizada en trabajos futuros para una implementación física.

1.4 Delimitación

La propuesta presente pretende enfocarse en la entrega del desarrollo de un algoritmo para generación de trayectorias implementado en el sistema robótico virtual MECABOT tal que se permita la evasión de obstáculos sobre un terreno conocido calculando indicadores de desempeño.

Cabe resaltar que modificaciones al diseño mecánico y electrónico de los módulos, así como a su configuración individual y a la arquitectura del sistema no son objetivo de esta propuesta de trabajo de grado.

1.5 Justificación

A diferencia de la navegación manual, la navegación autónoma promueve la productividad y calidad de las diversas tareas realizadas por los robots, con esto se pueden generar oportunidades de riqueza o bienestar social. [6] [7]

Aplicaciones como la paquetería y mensajería, la limpieza, la agricultura, la vigilancia, la construcción, el transporte y la guía de personas se benefician de una navegación autónoma. La búsqueda y rescate es otra aplicación que aprovecha el uso de este tipo de navegación puesto que, aunque la mayoría de los sistemas son tele operados, el otorgar la capacidad de navegar y buscar autónomamente permite que varios robots cubran la misma área en un menor tiempo, aumentando así, las probabilidades de rescatar a una persona presentando un mejor plan de rescate. [7]

Actualmente, la búsqueda y rescate tiene varios retos, como lo son la movilidad en terrenos con escombros, manejo de la energía para misiones de larga duración y la capacidad de identificar a las víctimas. Aunque cada aplicación presenta retos específicos, la navegación es un problema en común. Los robots deben desplazarse sin dañarse y alcanzar el objetivo en un ambiente no controlado, por eso la tarea presentada les es difícil. [7]

En Colombia, entre el 2006 y el 2014 se han presentado alrededor de 3181 muertos y 12,3 millones de afectados a causa de desastres naturales [8], y al 2012 se registraron 223 muertos y 1343 lesionados por atentados terroristas. [9] Dadas estas cifras, es necesaria la búsqueda del mantenimiento del derecho a la vida de los ciudadanos, por lo que la tarea de búsqueda y rescate, más aún en esta época, es imperativa.

El grupo DAVINCI ha presentado distintos prototipos del robot MECABOT, cada uno provee campos de investigación que permite su mejoramiento continuo. Con su enfoque principal siendo las posibles configuraciones del sistema, se ha dejado un poco de lado su mecanismo de navegación. Por esto, es necesario empezar a investigar sobre el tema si se quiere lograr el objetivo para el cual fue diseñado.

1.6 Metodología

La metodología implementada en el desarrollo de este trabajo se evidencia en la *Figura 1.1.* Primero, se revisa la literatura relacionada con la navegación en la robótica modular y los enfoques clásicos, así como trabajos anteriores relacionados con el MECABOT, posteriormente, según lo encontrado, se programa el procesamiento de las imágenes y el algoritmo de navegación, con sus respectivas simulaciones. En base a los resultados obtenidos de las simulaciones

y con la familiarización de la arquitectura cuadrúpeda del MECABOT 5.0 se implementa el algoritmo en el sistema físico y se realizan las pruebas de funcionamiento. De lo observado a lo largo de este proceso y de la experiencia con la programación de algoritmos de navegación, se realiza una retroalimentación y recomendaciones a trabajos futuros.

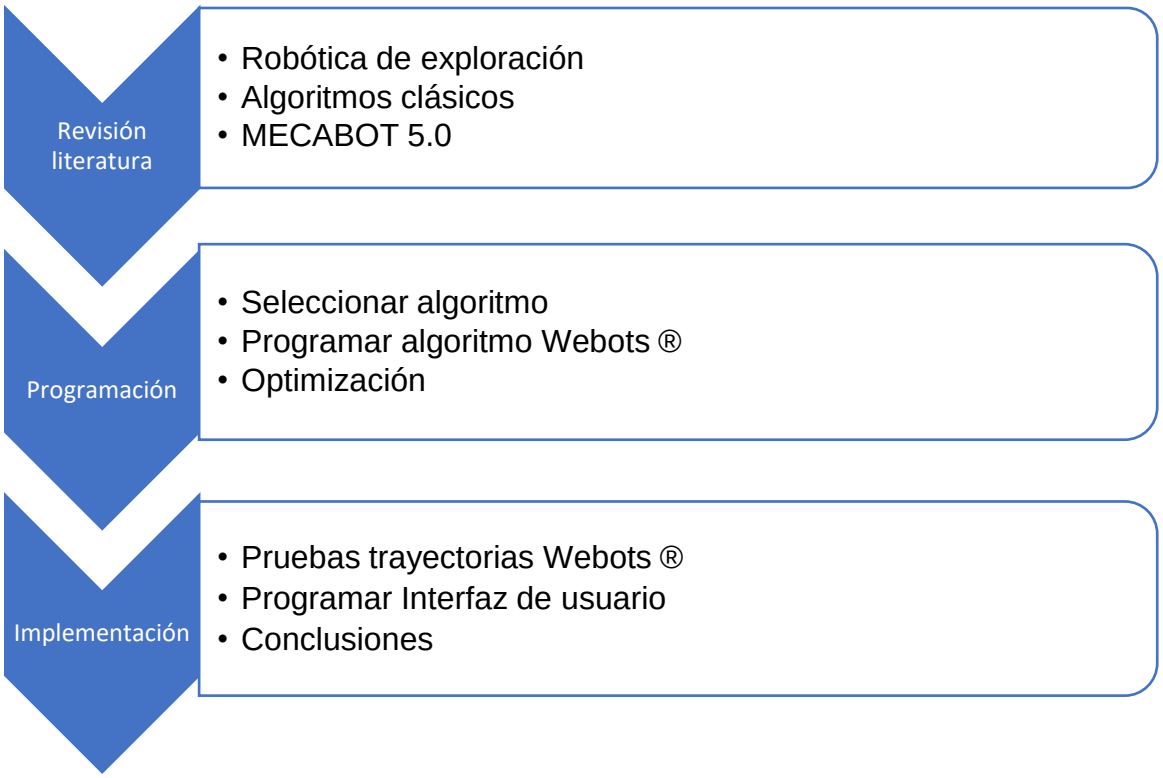


Figura 1. 1 Metodología

CAPÍTULO 2
MARCO REFERENCIAL

Esta sección describe brevemente los conceptos de robótica y robótica móvil para luego enfocarse en varios aspectos en relación con la robótica de exploración.

Adicionalmente, se presenta un breve trasfondo teórico asociado a la robótica modular, describiendo algunas arquitecturas comúnmente utilizadas.

Finalmente, se revisan los conceptos que fundamentan la navegación autónoma, y la resolución de los problemas que se presentan en ella.

2.1 Robótica

La palabra "*robot"* fue introducida por el novelista checo Karel Capek, en su obra "*Rassum's Universal Robots (RUR)"* en 1920; proveniente de *Robota,* checoslovaco para trabajador o sirviente. Alrededor de 1930 se popularizo el término gracias a varias novelas y cuentos cortos de ciencia ficción de Isaac Asimov, quien inventó y propuso las leyes de la robótica en 1942. [10] [11]

Aunque existen distintos planos, diseños y modelos de robots inspirados en humanos presentados por Leonardo da Vinci en los años 1500, hay registros que datan hasta el 85 d.C donde Herón de Alejandría propone mecanismos hidráulicos, poleas y palancas denominados *autómatas,* que imitan los movimientos de seres animados. [10] [45]

Lo que empieza como un auge industrial de la robótica en 1961 con la aparición del primer brazo manipulador programable de la compañía *Unimation,* se expande rápidamente generando nuevos campos de aplicación e investigación, como lo son la robótica móvil y la robótica modular, los cuales son tratados en este trabajo. [10]

2.2 Robótica móvil

La robótica móvil es aquella que cubre todos los robots que ruedan, caminan, vuelan o bucean, es decir todo robot que sea capaz de locomoción mediante algún actuador. [12]

En la robótica móvil existen tres preguntas que cada robot debe de responder para poder lograr su tarea: ¿Dónde estoy? ¿A dónde voy? y ¿Cómo llegar? Para dar respuesta, un robot debe de realizar mediciones, obtener un modelo del ambiente, localizarse a sí mismo y planificar un camino hacia su objetivo. Con relación a la tesis, únicamente se abordan los aspectos de la planificación y seguimiento de trayectorias. [12] [13]

Actualmente se presenta varias aplicaciones para los robots móviles, y su importancia en la industria continúa creciendo. La robótica móvil puede ser implementada en tareas de transportación, vigilancia, limpieza, entretenimiento, exploración, entre otras. [14]

2.3 Robótica de exploración

La robótica de exploración se trata fundamentalmente de enviar robots móviles dentro de ambientes naturales o artificiales en donde los seres humanos no pueden ingresar para obtener información valiosa o realizar alguna tarea física. Algunos de los diversos escenarios y aplicaciones en donde la robótica de exploración tiene el potencial para producir un impacto significativo son: Defensa y seguridad, búsqueda y rescate, arqueología, monitoreo ambiental, desmantelamiento nuclear, exploración espacial y la industria de gas y petróleo. [5] [15]

La robótica de exploración describe la mayoría de las aplicaciones de robot móviles que no cubre la robótica de servicio o de infraestructura. Algunos casos de robótica de infraestructura pueden ser descritos como robótica de exploración, pero a diferencia de ella sus escenarios son muy repetitivos. A comparación de la robótica de servicio, la robótica de exploración usualmente debe operar en ambientes que son menos estructurados y más desafiantes desde un punto de vista de locomoción. [15]

Debido a los ambientes desafiantes a los cuales se debe enfrentar la robótica de exploración, el sistema de locomoción es el principal enfoque. La locomoción bio-inspirada es, en ciertas ocasiones, beneficial, por lo que se relaciona con la investigación de control bio-robótico. El diseño de la plataforma y la integración de los sistemas también es una prioridad alta puesto que se requiere crear robots capaces de sobrevivir en ambientes hostiles lo suficiente para lograr su misión. [15]

Por lo general, los robots de exploración deben de ser pequeños por lo que se pueden beneficiar de los futuros procesos de manufactura. El propósito más común es adquirir información, por lo que se requiere de tecnologías avanzadas de sensado; al operar remotamente en ambientes complejos, obtienen beneficios de los avances realizados en redes y comunicaciones. De esta forma, aunque haya un humano involucrado, la robótica de exploración se vuelve cada vez más autónoma, desarrollando campos de investigación en inteligencia artificial. [15]

2.4 Robótica Modular

En las últimas décadas, la inspiración de la ingeniería por la mejora de la adaptabilidad, funcionalidad, confiabilidad y robustez de los sistemas robóticos dio

como resultado a los sistemas multi robot. Aunque estos sistemas puedan realizar tareas específicas correctamente, por lo general presentan limitaciones en la adaptación a ambientes impredecibles y en el cambio de tareas. Con la flexibilidad, adaptabilidad y autoorganización que presentan los sistemas biológicos multicelulares como referencia, la robótica se ha orientado a crear máquinas auto organizables que se adapten a condiciones inesperadas, de esto surgió la robótica modular. [16]

La robótica modular se enfoca en el diseño de robots conformados por partes simples o módulos, de forma que al unirse con otro robot de características similares o iguales se pueda generar una estructura robótica más compleja para realizar una tarea específica. Con esto se busca la habilidad de formar nuevos sistemas con distintos tipos de locomoción tal que pueda superar obstáculos en terrenos desconocidos o peligrosos. [17]

Los módulos se pueden percibir relativamente como estructuras simples, cada uno dispone de sistemas de actuación, sensores, interfaz de comunicación y de cálculo. Pueden poseer varias formas. Pese a que pueden actuar independientemente, los módulos poseen mecanismos de acoplamiento que les permiten conectarse con otros módulos, esto les da la propiedad de configurabilidad o auto configurabilidad, dependiendo si esta estructuración es externa o realizada de forma autónoma por los módulos. [17] [18] [45]

2.5 Clasificación de los robots modulares

Los robots modulares pueden ser clasificados según el método de locomoción y estructura, adicionalmente pueden ser clasificados según la conexión entre módulos adyacentes. [19]

Según el método de locomoción:

2.5.1 Configuración Móvil

Cada módulo es capaz de desplazarse de forma independiente permitiendo una interacción eficaz con el ambiente que lo rodea. (ver *Figura 2.1. Kilobots)* [45]

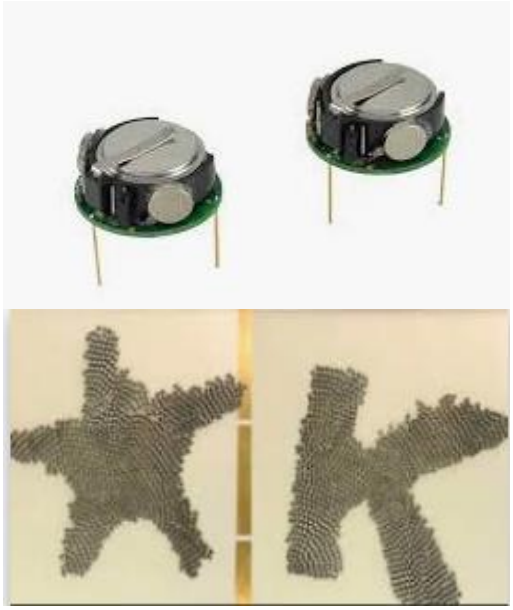


Figura 2. 1 Kilobots, capaces de formar una estrella y la letra K

Harvard University, (2014), Kilobots. Recuperado de [20]

2.5.2 Locomoción de cuerpo completo

Se requiere de más de un solo módulo para generar una arquitectura que permita la locomoción e interacción con el ambiente, ejemplo de esto es el Robot M-TRAN.

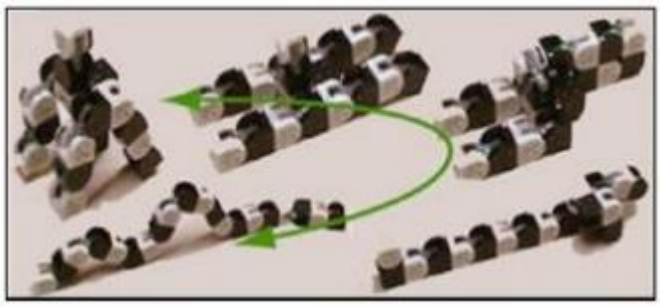


Figura 2. 2 Robot Modular M-TRAN

Cruz, V.(2018). Simulación e implementación de arquitectura cuadrúpeda utilizando sistema robótico modular MECABOT.[Figura] Recuperado de [45]

Según su estructura:

2.5.3 Arquitectura de red o lattice

Forma arquitecturas conectadas en un patrón 3D, el control y movimiento se ejecutan en paralelo, es computacionalmente más simple, y debido a que imita en

cierta forma la manera en la que los átomos se juntan para formar moléculas, son escalables a sistemas más complejos. [19] [21] [45]

Figura 2. 3 Robot ODIN

Mendoza, R. Robot Odin (2008). [Figura] Recuperado de [22]

2.5.4 Arquitectura de cadena o chain

Los módulos están conectados en línea, es decir, en topología de árbol; permite configuraciones simples y además es capaz de doblegarse para cambiar de plano de acción o formar una estructura en 3D. Su arquitectura serial estricta lo hace versátil aunque computacionalmente difícil de analizar y representar. [21] [45]

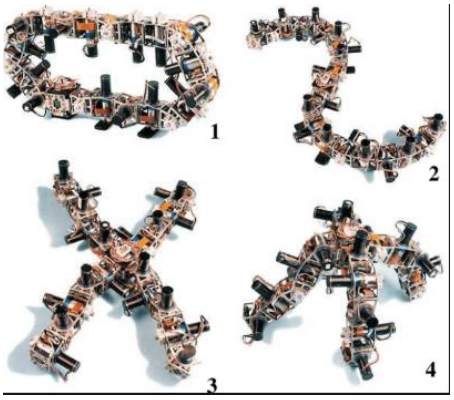


Figura 2. 4 Robot PolyBot

Gonzales, J. Ejemplo de autorreconfiguración probado con Polybot G2. [Figura] Recuperado de [23]

Existe también la arquitectura híbrida, que es una combinación de la de malla y la de cadena, lo que permite una mayor versatilidad en las configuraciones. [21]

Según la conexión entre módulos, existe la conexión *Pitch* en donde todos los módulos rotan alrededor del eje de *pitch.* La conexión *Yaw,* en donde todos los módulos rotan alrededor del eje de *yaw* y la conexión híbrida, en donde algunos módulos rotan alrededor del eje de *pitch* y otros alrededor del eje de *yaw.* [19]

2.6 Navegación Autónoma y el problema de la navegación

La navegación autónoma consiste en la habilidad de un vehículo para planear una ruta y desplazarse a lo largo de esta sin intervención humana, esto con el objetivo de desplazarse de un punto en el espacio a otro y realizar una tarea específica, esto mientras se evitan los obstáculos presentes en el ambiente. En algunos casos, la navegación remota ayuda con el proceso de planeación, mientras que en otros, la única información disponible para planear una ruta es la que proporcionan los sensores a bordo del vehículo, esta información puede ser de tipo visual, auditiva u olfatoria. Un vehículo autónomo no solo es capaz de mantener su estabilidad mientras se mueve, sino que también es capaz de planear sus movimientos. [6] [24]

Una vez la información de posición es adquirida, ya sea por señales trianguladas o percepción ambiental, cierta inteligencia debe de ser aplicada para traducir esta información en objetivos, hacia una trayectoria y plan de movimientos. En sistemas multi agente, dicho plan debe de acomodar las intenciones, comunicadas o estimadas, de los demás robots autónomos tal que se eviten colisiones mientras se toma en cuenta la dinámica de este. [24]

2.6.1 Rutas y planeación

Si se dispone de un mapa global, un vehículo autónomo puede planear su ruta desde se ubicación actual hasta su objetivo. Algunos métodos buscan el camino óptimo basado en reglas que tratan de minimizar tiempo de tránsito, consumo de combustible, exposición a amenazas, entre otros. Varias rutas son evaluadas basados en la selección de puntos disponibles, y aquellos que se acomoden más a las reglas son escogidos. A lo largo del desplazamiento se pueden dar amenazas inesperadas que causen la necesidad de la violación de las reglas, por lo que la ruta debe ser recalculada desde el punto donde se encuentra. Si ninguna ruta se acomoda a las reglas, entonces dichas reglas deben de ser más flexibles. Se tiene en cuenta que hay reglas que en ninguna circunstancia deben de ser cambiadas o violadas, por ejemplo, si una ruta requiere que el vehículo consuma más combustible del que dispone es, obviamente, una alternativa inaceptable.

Si no se tiene acceso a un mapa global, la ruta óptima no puede ser predicha, por lo que se utilizan técnicas de reconocimiento muerto y comportamientos de búsqueda-prevención. El reconocimiento muerto utiliza el tiempo transcurrido a cierta velocidad a lo largo de una dirección para extrapolar una nueva posición basado en la posición actual. Un ejemplo de reconocimiento muerto es la odometría, usualmente implementada en robots de fábricas para determinar la distancia recorrida a partir de las revoluciones de un disco de circunferencia conocida. La odometría visual también es posible a partir de robots aéreos en donde el desplazamiento de objetos en el suelo es notable, y, conociendo la altitud

a la que se encuentra el robot y su rango de visión, se puede deducir la distancia recorrida. [24]

2.6.2 El problema de la navegación

Los vehículos autónomos proveen soluciones superiores para varias aplicaciones, pero el reto más grande es el de proveer una navegación robusta en todas las situaciones. Para esto, un sistema de navegación debe tener en cuenta seis atributos: Objetivo (motivación para moverse), habilidad para percibir el ambiente (evasión de obstáculos), entendimiento de la ubicación actual, habilidad para planear un camino hacia el objetivo, movilidad auto accionada y la habilidad para replantear mientras se avanza (compensar situaciones inesperadas). [6] [24]

2.6.3 Componentes de la navegación

La navegación, como se mencionó previamente, es robusta si se tienen en cuenta seis atributos, pero cuando no se disponen de sensores, se reducen a 4, proveyendo así, una navegación básica pero funcional.

2.6.3.1 Localización

Este bloque busca estimar la posición actual de robot respecto a un sistema de coordenadas global. Normalmente se disponen de tres fuentes de información: Posición dada por odometría, las señales de los sensores y un mapa del entorno, el cual ha de ser realizado con anterioridad.

2.6.3.2 Mapeo

El mapeo consiste en la construcción de una representación útil del entorno, esta da soporte a la localización y a la planificación. Existen varios métodos para obtener dicha representación, que son los que se abordarán más adelante.

2.6.3.3 Planificación

Un planificador de trayectorias debe ser capaz de generar una ruta libre de obstáculos entre dos puntos cualquiera del mapa. Una trayectoria se puede definir como una secuencia de puntos en el mapa que unen el punto inicial y el punto objetivo, en donde la línea recta que une dos posiciones contiguas no contenga un obstáculo. [6]

2.6.3.4 Seguimiento

Una vez calculada la trayectoria, se generan velocidades de desplazamiento y de rotación que se traducen a velocidades en los motores, a partir de los puntos que la conforman.

A continuación se observa un diagrama de la forma en la que cada uno de estos atributos se interconecta. (ver *Figura 2.5. Diagrama de la arquitectura de la navegación autónoma)*

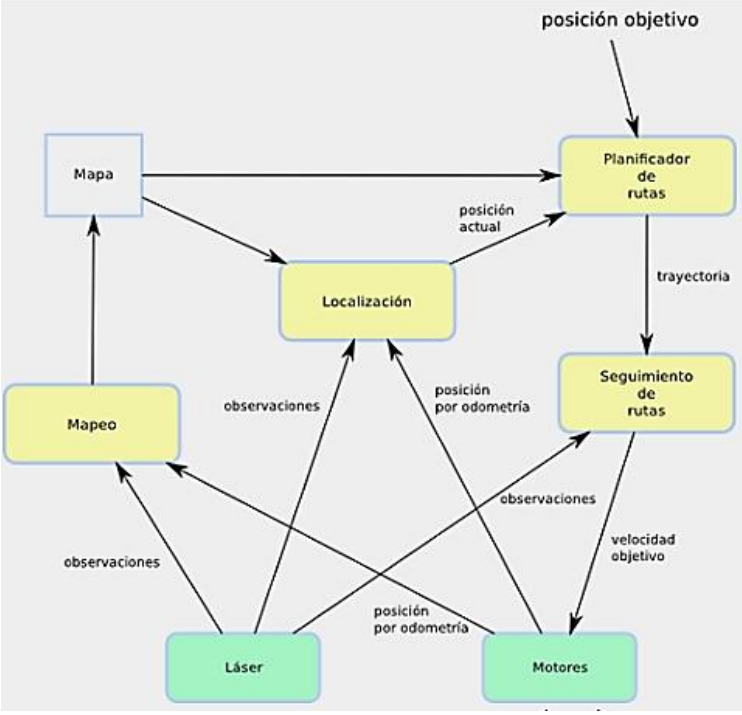


Figura 2. 5 Diagrama de la arquitectura de la navegación autónoma

Azkune, G. Navegación autónoma. [Figura] Recuperado de [6]

2.6.4 Trabajos realizados

- En [46], se presenta una arquitectura de control para la navegación semiautónoma de una plataforma robótica utilizando información sensorial proveniente de un sensor de mapeo 3D. En este trabajo, el sistema de control provee al robot de la habilidad de aprender y tomar decisiones respecto a que tareas de rescate se deberían llevar a cabo en un momento determinado sin comprometer la seguridad de la víctima, rescatistas y el robot mismo.

- Para [47], la inclusión de un módulo de evasión de obstáculos es requerido para lograr vehículos completamente autónomos. En su trabajo, presentan métodos de detección de colisiones y planeación de trayectorias, enfocándose en la diferencia entre planeadores locales y globales, describiendo las técnicas más comunes derivadas de la teoría clásica de búsqueda gráfica. Adicionalmente se realiza un análisis de métodos inteligentes como redes neuronales y algoritmos evolutivos.

- Según [48], un algoritmo de planeación de rutas o caminos es esencial para guiar vehículos no tripulados entre dos puntos o a lo largo de una trayectoria. Identifica el algoritmo A* como uno de los más eficientes para calcular una ruta segura con el menor costo de distancia. Menciona también las restricciones que este presenta debido a la resolución del mapa y la posibilidad de que no sea compatible con las restricciones no holonómicas del sistema.

- Varios estudios han sido realizados sobre este tema, como lo es uno presentado en [49], en donde se genera un análisis cuantitativo de la sensibilidad de la planeación por descomposición de celdas a factores como optimalidad y complejidad computacional , evaluando la influencia de las funciones de costo utilizadas en la fase de búsqueda. Otro estudio es el presentado en [50], en donde se genera una comparación entre los métodos clásicos y los métodos heurísticos.

2.7 MECABOT

El desarrollo de esta plataforma se da en el año 2013 por el grupo de investigación DAVINCI de la Universidad Militar Nueva Granada, con el fin de crear módulos compactos que tengan la posibilidad de acoplarse entre ellos para conformar estructuras más versátiles, en el aspecto cinemático, en terrenos irregulares, esto para aplicaciones de exploración y de búsqueda y rescate. [25] [45]

La primera versión consistía en módulos impresos en ABS, cada uno disponía de cinco servomotores, un controlador PIC24HJ12GP201 de Microchip, Driver Micro Maestro 6 de Pololu, un módulo de comunicación RF Transreceptor +0dBm, 6 sensores de proximidad y cuatro baterías de litio. [26] [27] [28] [45]

Esta primera versión logró un movimiento individual de los módulos, pero dado el tamaño y peso de estos, al momento de acoplarlos (el acople era mecánico, solo se acoplaba en los extremos, no en los laterales) los motores no tenían la capacidad de moverlo, por lo que el MECABOT 1.0 no podía llevar a cabo ninguna configuración propuesta. [26] [45]

En la versión 2.0 se cambió el acople mecánico por uno de imanes permanentes, se reemplazó el controlador por un Arduino Nano y la comunicación se llevó a cabo

mediante una Xbee Pro-serie 1. Aunque se conserva la cantidad de motores y el peso se redujo aproximadamente a la mitad, el robot es incapaz de desplazarse y mantener una configuración. [27] [45]

Figura 2. 6 Comparación MECABOT 1.0 (rojo) y MECABOT 2.0 (negro)

Cotera M. Simulación e Implementación de una Configuración de Robot Hexápodo Utilizando Sistemas de Robótica Modular para Evaluar su Locomoción. .[Figura]. Recuperado de: [28]

Teniendo en cuenta las falencias presentadas en los módulos hasta el momento, se rediseñaron una vez más. La tercera versión posee 6 servomotores (de 1.8 kg y 1.3 kg) con una tarjeta Teensy 3.2 para su comando. Para ese año (2016) se obtuvo la implementación de la configuración serpiente, permitiendo movimientos ondulatorios típicos de topologías 1D. Para lograr las configuraciones de rueda y serpiente se requería de una mayor fuerza de acople y mayor capacidad de levantamiento por parte de los motores. [28]

Figura 2. 7 MECABOT 3.0

Lancheros P, Sanabria L. Simulación e Implementación de Movimientos para Sistema Robótico Modular Considerando Diferentes Configuraciones. .[Figura]. Recuperado de: [27]

Luego en el 2017, se desarrolla la versión 4.0, la cual tiene un peso menor que su predecesor y cuenta con solo cuatro servomotores, esta vez de un torque mayor (4 kg), un regulador Step-Down para evitar sobre picos de corriente y dos baterías de litio ubicadas únicamente en el módulo trasero. Con esta versión se logra la configuración de rueda y oruga y la primera arquitectura con patas, el hexápodo. [28] [45]

Figura 2. 8 MECABOT 4.0

Cotera M. Simulación e Implementación de una Configuración de Robot Hexápodo Utilizando Sistemas de Robótica Modular para Evaluar su Locomoción. .[Figura]. Recuperado de: [28]

Un año después, con la quinta versión del MECABOT, se logra implementar la configuración cuadrúpeda.

Como se ha mencionado anteriormente, uno de los desafíos más significativos de la robótica autónoma es el de la planificación autónoma de movimiento. El objetivo es que se especifique una tarea en un lenguaje de alto nivel, es decir lenguaje cotidiano, y el robot sea capaz de compilar dicha tarea en un grupo de acciones de movimiento, o controladores, para realizarla sin necesidad de intervención humana. La tarea más común es la de encontrar un camino para el robot, ya sea un brazo manipulador o un robot móvil, desde una configuración a otra, mientras evade obstáculos. [30] [31]

Uno de los primeros problemas formulados fue el *Piano mover's problem,* o el problema del piano, el cual provee el diseño de un modelo preciso asistido por computadora de una casa y un piano como entrada a un algoritmo, el cual debe determinar cómo trasladar el piano de una habitación a otra sin chocar. [29] [30] [32] Desde este problema, la planificación del movimiento ha evolucionado para considerar una gran cantidad de variaciones, permitiendo aplicaciones en áreas tales como la animación de personajes digitales, planeación quirúrgica, verificación automática del diseño de fábricas, mapeo de ambientes inexplorados, navegación de ambientes cambiantes, secuencias de ensamblaje, diseño de medicamentos, asistencia para los discapacitados, entre otros. Cada aplicación nueva trae consigo nuevas consideraciones que deben ser tomadas en cuenta en el diseño de algoritmos de planeación de movimiento. [30] [31]

La planificación de movimiento en un robot se basa primordialmente en las traslaciones y rotaciones necesarias para mover el piano, ignorando (e.g.) la dinámica y varias limitaciones diferenciales. Mientras que una planificación de trayectorias refiere el problema de obtener la solución dada por un algoritmo de planeación de movimiento y determinar cómo moverse a lo largo de esta solución, de forma que se acomode a las limitaciones mecánicas del robot. [32]

2.8 Elementos primarios de la planeación

Aunque la naturaleza de los distintos métodos de planeación difiere entre sí, se constituyen de ciertos elementos primarios, según Steven M. LaValle, estos son:

2.8.1 Estado

Los problemas de planeación involucran un espacio de estados que contiene todas las situaciones posibles que puedan surgir. Por ejemplo, un estado puede representar la orientación y posición de un robot, la ubicación de las fichas en un rompecabezas, o la posición y velocidad de un helicóptero. Los espacios de estados se permiten ser discretos (finitos o infinitamente contables) y continuos (infinitamente incontables). Usualmente, el espacio de estados es representado implícitamente por un algoritmo de planeación. En la mayoría de los casos, el tamaño del espacio de estados (número de estados o complejidad combinatoria) es muy grande para ser representado explícitamente. [32]

2.8.2 Tiempo

Todos los algoritmos de planeación involucran secuencias de decisiones, las cuales han de ser implementadas a lo largo del tiempo. Este elemento temporal puede ser modelado explícitamente (p. e. Manejo de un vehículo a través de una pista de obstáculos en el menor tiempo posible) o implícitamente, visto en el hecho de que se deben seguir acciones de forma sucesiva (p. e. Solución a un cubo Rubik). [32]

2.8.3 Acciones

Un plan genera acciones que manipulan los estados. En alguna parte de la formulación de la planeación se ha de especificar como cambian los estados cuando se aplican acciones. Esto puede ser realizado a través de funciones de evaluación de estados (tiempo discreto) o a través de funciones diferenciales (tiempo continuo). Para la mayoría de las planeaciones de movimiento, se evita hacer referencia explícita del tiempo, especificando un camino a través de un espacio continuo de estados. Estos caminos pueden ser obtenidos a partir de la integración de ecuaciones diferenciales, pero no siempre es necesario. Para algunos problemas, las acciones son escogidas por ‘naturaleza’ e interfieren con los resultados y no están en control de la unidad de decisiones. Esto permite incertidumbre y predictibilidad. [32]

2.8.4 Estados iniciales y finales

Un problema de planeación involucra, usualmente, empezar en un estado inicial y tratar de llegar a un estado final específico, o a un estado dentro de un grupo de estados finales. Las acciones son escogidas de forma tal que se cumpla.

2.8.5 Criterio

Un criterio determina el resultado deseado de un plan en términos de los estados y acciones que son ejecutadas. Generalmente existen dos clases de aspectos basados en el tipo de criterio.

2.8.5.1 Factibilidad: Encontrar un plan que cause la llegada al estado final, independiente de su eficiencia.

2.8.5.2 Optimación: Encontrar un plan factible que optimice el desempeño en alguna forma especificada.

Para la mayoría de los problemas, lograr la optimación es considerablemente complejo, debido a que la sola formulación de un criterio de optimización es complicada. Incluso si el criterio deseado logra ser formulado, puede resultar imposible obtener un algoritmo práctico que compute planes óptimos. En estos casos, un plan factible es más deseado que no tener una solución en absoluto. [32]

2.8.6 Plan

De forma general, un plan impone cierta estrategia específica o comportamiento al momento de realizar decisiones. Un plan puede, en su forma más simple, especificar una secuencia de acciones a realizar, en varios casos se vuelve más complicado. Si no se pueden predecir estados futuros, entonces un plan especifica acciones como funciones de estado, así, independiente de los estados futuros, la acción apropiada es determinada. Esto permite realimentación o planes reactivos. También se puede dar el caso de que los estados no pueden ser medidos, por lo que la acción correcta puede ser determinada a partir de toda la información disponible en el momento, a esto se le refiere como ‘estado de información’, en donde las acciones de un plan están condicionadas. [32]

2.9 Algoritmos, planificadores y planes

A continuación se describe la diferencia entre lo que es un algoritmo, un planificador y un plan para que al ser referenciados más adelante, se conozca el tema a tratar.

2.9.1 Algoritmos

Para [32], un algoritmo puede ser percibido como una máquina de Turing, la cual es una máquina de estados finitos que lee y escribe a lo largo de una cinta infinitamente larga. La entrada para el algoritmo está codificada como una cadena de símbolos (generalmente binarios) que luego son escritas en la cinta, la máquina procede a leerlas, y decide si 'aceptarlas' o 'rechazarlas'. Esta representación es funcional para la toma de decisiones, aunque existen extensiones de esta que pueden traer consigo otros resultados deseados, como lo es un plan.
Existe una línea que divide el ambiente de la máquina, esta línea es arbitraria y varía dependiendo del problema en consideración. Una vez definida, la máquina obtiene información de entrada proveniente de los sensores durante su funcionamiento. Posteriormente, la máquina realiza acciones, que proveen actuación sobre el ambiente, esto puede alterar el ambiente de forma que nueva información podrá ser medida por los sensores. Esto es importante, puesto que si se utiliza la máquina de Turing como fundamento para los algoritmos, se debe de modelar cuidadosamente el mundo físico para que este pueda ser escrito en la cinta, leído por la máquina y esta pueda tomar decisiones. Si se presentan cambios en el ambiente durante la ejecución del algoritmo, no se poseerá certeza de lo que debería suceder. [32]

2.9.2 Planificadores

Un planificador es aquel que construye un plan, puede ser una máquina o un humano. Cuando es una máquina, se refiere a un algoritmo de planeación, en algunos casos, los humanos se pueden convertir en planificadores cuando construyen un plan que funciona en todas las situaciones posibles. Por ejemplo, un humano puede diseñar una máquina de estados que esté conectada a un ambiente, esto le da un modelo de planificación tal que el humano pueda desarrollar un plan. [32]

2.9.3 Planes

Una vez se determina un plan, existen tres maneras de usarlo:

2.9.3.1 Ejecución

Existen dos formas generales en la que una máquina ejecuta un plan. La primera, considera a la máquina programable y que puede aceptar varios planes posibles antes de ejecutarlo, se asume que una vez se vaya a ejecutar el plan, la máquina se vuelve completamente autónoma y ya no podrá interactuar con el planificador. Estas consideraciones no prohíben la actualización de los planes, aunque es preferible que un plan sea diseñado teniendo en cuenta información nueva que pueda ser obtenida durante su ejecución.

En la segunda forma de ejecución, el plan obtenido codifica una máquina entera, y está diseñado como una máquina de propósito especial para resolver las tareas específicas, dadas al planificador inicialmente. [32]

2.9.3.2 Refinamiento

Si un plan es utilizado para refinamiento, este debe ser aceptado como entrada a otro planificador. El plan resultante puede presentar más problemas a consideración, o simplemente ser más eficiente. El refinamiento puede ser aplicado varias veces, lo que busca generar una secuencia de planes mejorados hasta que el último sea ejecutado. Un ejemplo presentado por [32] es el de un robot móvil en interiores, en donde el primer plan es un camino libre de colisiones a través del edificio. El segundo plan, transforma dicho camino, en una ruta que satisfaga algunas limitaciones diferenciales, basadas en movimientos en las ruedas. El tercer plan considera como mover el robot a lo largo de la ruta a diferentes velocidades mientras se satisfacen consideraciones del momento. Finalmente, el cuarto plan incluye retroalimentación para asegurar que el robot se mantenga lo más cerca posible a la trayectoria planeada a pesar de comportamientos impredecibles. (Ver Figura 2.9)

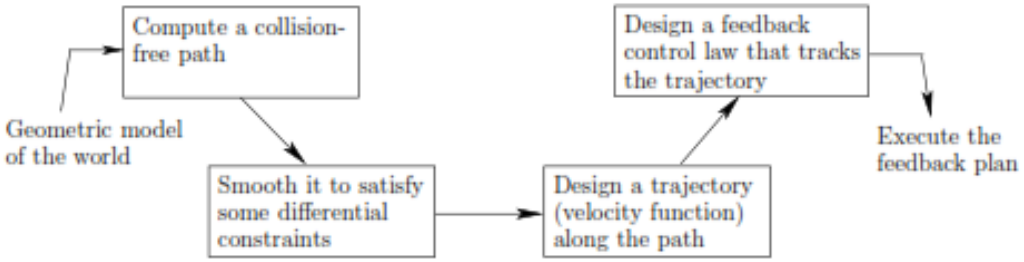


Figura 2. 9 Refinamiento del plan para un robot móvil de interiores.

Choset, H; Lynch, K; Hutchinson, S; Kantor,G; Burgard L y Thrun, S. Principles of Robot Motion. .[Figura]. Recuperado de [30]

2.9.3.3 Inclusión Jerárquica

La inclusión jerárquica incorpora un plan como una acción dentro de un plan mayor, puede considerarse al plan original como una subrutina. Para que esto tenga éxito, es necesaria la terminación del plan obtenido para que el plan mayor pueda realizar más acciones de ser necesario. La inclusión jerárquica puede ser realizada cualquier número de veces, resultando así en una planeación en ‘árbol’. [32]

2.10 Caracterización de un planificador de movimiento

La caracterización de un planificador de movimiento se da de acuerdo con la tarea a realizar, las propiedades del robot realizando la tarea y las propiedades del algoritmo.

2.10.1 *Tarea*

Una de las características más importantes de un planificador es el problema que resuelve, algunos de estos son: navegación, cobertura, localización y mapeo. [30] La navegación es el problema de encontrar un movimiento libre de colisiones para el robot, yendo de una configuración (o estado) a otra. La cobertura es el problema de pasar un sensor o herramienta a través de todos los puntos en un espacio, como en la minería o pintura. La localización es el problema de usar un mapa para interpretar la información de un sensor para determinar la configuración del robot. El mapeo es el problema de explorar y sensar un ambiente desconocido para generar una reconstrucción útil para la navegación, cobertura o localización. Existen otros varios problemas de planificación tales como: Navegación a través de obstáculos móviles, planeación de manipulación y agarre, planeación de ensamblaje y coordinación entre varios robots. [30] [32]

2.10.2 *Propiedades del robot*

La forma de un planificador de movimiento eficiente depende en gran medida de las propiedades del robot realizando la tarea en cuestión. Por ejemplo, el robot y el ambiente determinan el número de grados de libertad del sistema y la forma del espacio de configuraciones. Una vez se comprenda este espacio, se toma en consideración si el robot es capaz de desplazarse instantáneamente en cualquier dirección dentro de su espacio de configuraciones (omnidireccional), o si presenta restricciones de velocidad (no holonómico). Finalmente, el robot puede ser modelado utilizando ecuaciones cinemáticas, con velocidades como control, o ecuaciones dinámicas de movimiento, con fuerzas como control. [30]

2.10.3 Propiedades del algoritmo

Una vez se haya determinado la tarea a realizar y el sistema robótico, se procede a escoger el algoritmo según la forma en la que resuelven el problema. Por ejemplo, una pregunta que se puede plantear es: ¿El planeador encuentra movimientos que son óptimos en alguna manera, tales como en magnitud, tiempo de ejecución, o consumo energético? ¿O simplemente encuentra una solución que satisfaga las restricciones? Adicionalmente se pueden realizar preguntas sobre la complejidad computacional. [30]

Algunos algoritmos pueden ser descritos como ‘completos’ en el sentido de que siempre encontraran una solución al problema de planeación si existe, e indicara fallo en tiempo finito si no. A medida que aumenta el número de grados de libertad, las soluciones ‘completas’ serán computacionalmente intratables, debido a esto se pueden buscar otras formas de completitud, como lo son la ‘completitud en resolución’, es decir, si existe una solución a cierto nivel de resolución o discretización, el planeador la hallará, o la ‘completitud probabilística’, en donde la probabilidad de hallar una solución (si la hay) converge a 1 a medida que el tiempo tiende a infinito.

La optimización, completitud y complejidad computacional se relacionan entre sí, por ejemplo, si se desean planes de movimiento óptimos o completos, se debe estar dispuesto a aceptar una complejidad computacional mayor.

Otra característica considerable es si el planeador va a ser *offline* (construcción del plan a priori basado en un modelo del ambiente para luego ser ejecutado) u *online* (construcción incremental del plan a medida que se va ejecutando), este último es basa en sensores. De alguna forma, si un planeador *offline* se ejecuta lo suficientemente rápido, puede ser utilizado en un lazo de realimentación para replanificar continuamente cuando nueva información de los sensores genere un modelo actualizado del ambiente. La diferencia principal entre los planificadores *offline* y *online* es el tiempo de computación.

<table>
<tr><th>Tarea</th><th>Robot</th><th>Algoritmo</th></tr>
<tr><td>Navegación</td><td>Espacio de configuraciones, grados de libertad</td><td>Movimientos óptimos / no óptimos</td></tr>
<tr><td>Mapeo</td><td rowspan="2">Cinemática / dinámica</td><td>Complejidad computacional</td></tr>
<tr><td>Cobertura</td><td>Completitud (resolución, probabilística)</td></tr>
<tr><td rowspan="2">Localización</td><td rowspan="2">Omnidireccional o con restricciones de movimiento</td><td>Online / Offline</td></tr>
<tr><td>Basada en sensores / modelo del ambiente</td></tr>
</table>

Tabla 2. 1 Conceptos para caracterización de un planificador de movimientos

Choset, H; Lynch, K; Hutchinson, S; Kantor,G; Burgard L y Thrun, S. Principles of Robot Motion. .[Tabla]. Recuperado de [30]

Las secciones descritas a continuación, se basan en [32], donde se explica a fondo los fundamentos matemáticos de los algoritmos de planeación, sus variaciones, métodos de optimización y consideraciones necesarias. Las siguientes

explicaciones están encaminadas a resaltar los conceptos más importantes para tener en cuenta para el desarrollo de esta tesis.

2.11 Planificación discreta

Según [32], la planificación discreta se basa en modelos en espacio de estados. Las distintas situaciones en las que se encuentre el ambiente se llamarán estados, que se denotaran como *x,* y el grupo de posibles estados se llamará un espacio de estados, denotados como *X.* Para la planificación discreta es importante que *X* sea contable, por lo general serán finitos. En una aplicación dada, se requiere de una definición cuidadosa del espacio de estados para que no haya información irrelevante codificada dentro de cada estado.

El ambiente puede ser transformado a través de las acciones escogidas por un planificador. Cada acción, denotada como *u,* al ser aplicada desde el estado *x* actual, producirá un nuevo estado, $x' \in X$ de forma tal que cumpla una función de transición de estados, *f.* Así:

$$x' = f(x, u) \tag{2.1}$$

Cabe mencionar, que una acción *u* puede ser aplicada en múltiples estados, por lo cual existe un conjunto *U,* que contiene todas las acciones posibles sobre todos los estados, y el conjunto *U (x)* y *U (x')* no están necesariamente separados, por esto:

$$U = \bigcup_{x \in X} U(x) \tag{2.2}$$

También se vuelve necesario definir un grupo de estados finales, $X_G \in X$, debido a que el objetivo de un algoritmo de planeación es encontrar una secuencia de acciones, que al aplicarse, transformen un estado inicial, x_i, en un estado dentro de X_G.

La formulación presentada por el autor para una planificación discreta, factible, es la siguiente:

- Un espacio de estado *X*, no vacío, el cual es un grupo finito o infinitamente contable de estados.
- Un espacio finito de acciones *U (x),* para cada estado $x \in X$
- Una función de transición de estado *f,* que produce un estado $f\ (x, u) \in X$, por cada $x\ \in X$ y $u\ \in U$.
- Un estado inicial $x_i \in X$
- Un grupo de estados final $X_G\ \subset X$

Se indica también, que un requerimiento para cada algoritmo de búsqueda es que sea sistemático, es decir, si el ambiente es finito, el algoritmo debe recorrer cada estado alcanzable, lo que le permite declarar correctamente si existe o no, una solución en tiempo finito. Para cumplir con esta condición, el algoritmo debe mantener cuenta de los estados ya visitados de forma que no se presente una búsqueda cíclica a través de estados ya visitados.

En cualquier punto a lo largo de la búsqueda de un plan factible, existirán tres tipos de estados:

- **Sin visitar:** Son todos los estados que no han sido visitados. Inicialmente, esto es cada estado a excepción de x_i.

- **Muertos:** Estados que ya han sido visitados y para los que cada estado siguiente posible también ha sido visitado. De cierta forma, se define 'muerto' puesto que ya no contribuyen nada más a la búsqueda de un pan factible.

- **Vivos:** Son los estados que han sido encontrados y posiblemente, algunos estados adyacentes que no han sido visitados. Inicialmente, el único estado 'vivo' es x_i.

El grupo de estados 'vivos' es almacenado en una lista prioritaria para la que se debe definir una función de prioridad. La única diferencia entre varios algoritmos para la búsqueda es la particularidad de la función de prioridad.

Esto será importante tener en cuenta para la sección 3.10, puesto que para algunos algoritmos expresados en la sección 3.8, se requieren de algoritmos de búsqueda que su funcionalidad varía según la forma en la que se maneja la lista prioritaria.

2.12 Espacio de configuraciones

Para la creación de planes de movimiento, se debe ser capaz de especificar la posición del robot en todo momento a lo largo de una trayectoria, más específico, se debe indicar la posición de cada punto del robot, dado que se debe asegurar que ninguno llegue a colisionar con algún obstáculo. Con esto se introduce el concepto de *espacio de configuraciones,* también denominado el espacio C de un robot. Para [31]:

"*Dado un objeto rígido A (el robot), moviéndose en un espacio de trabajo físico W, en donde se define un eje de referencia fijo F_W en W, un eje de referencia móvil F_A en A y una configuración q de A que especifica la posición y orientación de F_A respecto a F_W, el espacio de configuraciones de A es el espacio C de todas las*

posibles configuraciones q de A. Una configuración única en C, escogida arbitrariamente se llama configuración de referencia de A, denotada q=0."

El número de grados de libertad del robot definirá las dimensiones de este espacio. Por ejemplo, si se considera un robot móvil circular que se desplaza, sin rotar, sobre un plano, este se puede representar simplemente con la ubicación de su centro geométrico relativo a un eje coordenado fijo. Si se tiene el radio del robot, se puede determinar el espacio ocupado por el robot en una configuración. Este espacio se denomina *R (q)*. Según esto, para una configuración en un punto (x, y):

$$R(q) = \{x', y' | (x - x')^2 + (y - y')^2 \leq r^2\} \qquad (2.3)$$

Se puede observar que con solo los parámetros (x, y) se puede determinar la configuración del robot móvil.

Los robots tienden a moverse a lo largo de un espacio Euclidiano (aquel en donde se cumplen los postulados euclidianos) bidimensional o tridimensional, este ambiente es referido como el espacio de trabajo. [30]

2.13 Obstáculos en el espacio de configuraciones

En la formulación y solución de un problema de planeación de movimiento se requiere la definición y manipulación de modelos geométricos de los cuerpos de un sistema en el espacio. Existen varias formas de realizar un modelo geométrico, su selección depende de la aplicación y dificultad del problema. En general existen dos alternativas: representación sólida y representación de límites. [32]

Primero, se debe definir un mundo, *W*, el cual puede ser: 2D,$W = R^2$, o 3D, $W = R^3$. Se conoce que en el mundo contiene, generalmente, dos tipos de entidades:

- **Obstáculos:** Porciones del mundo que están ocupados permanentemente.

- **Robots:** Cuerpos modelados geométricamente, controlables por medio de un plan de movimiento.

Tanto obstáculos como robots se consideran subgrupos cerrados de *W*. La región de todos los puntos que conforman los obstáculos se denominará *O*, el cual es un subconjunto propio de *W*.

El problema de la planificación de trayectorias es la de crear un plan de movimiento continuo que contenga las configuraciones del robot a lo largo de esta, tal que ninguna configuración cause una colisión con otro robot o con un obstáculo. Para esto se ve necesario definir el grupo de configuraciones para las cuales ocurre una

colisión. Para esto se genera un espacio de configuraciones de obstáculos CO_i, el cual representa las configuraciones en las que el robot interseca un obstáculo *WO,* en el espacio de trabajo. [30]

$$CO_i = \{q \in Q \mid R(q) \cap WO \neq \emptyset\} \quad (2.4)$$

Y el espacio de configuraciones libre de obstáculos será:

$$C_{free} = C \setminus \bigcup_i CO_i \quad (2.5)$$

Con esto, se puede definir un camino libre como un mapeo continuo que no permite el contacto entre el robot y los obstáculos, y un camino semi-libre como aquel que permite contacto leve con obstáculos sin colisionar con ellos (p. e. Rozamiento).

Si se considera el robot móvil circular mencionado anteriormente en un ambiente donde solo existe un obstáculo poligonal, se pueden 'mapear' las limitaciones que impone el obstáculo sobre las configuraciones del robot, 'deslizando' el robot a lo largo de este. (Ver Figura 2.10)

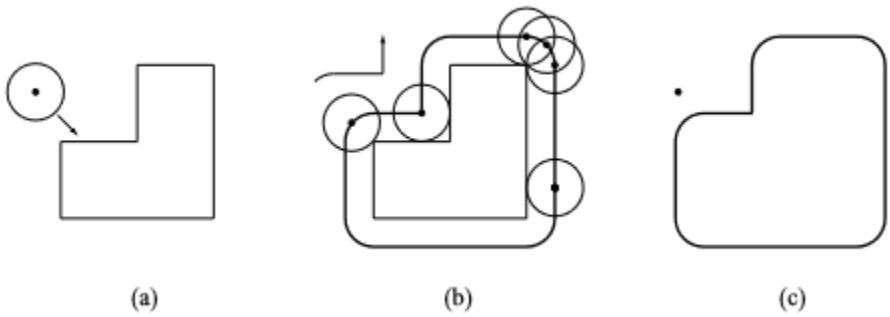


Figura 2. 10 a) El robot se acerca al obstáculo del espacio de trabajo. b) Al deslizar el robot a lo largo del obstáculo, teniendo en cuenta la línea trazada por el punto de referencia, se construye el obstáculo del espacio de configuraciones. c) Obstáculo resultante en el espacio de configuraciones

Choset, H; Lynch, K; Hutchinson, S; Kantor,G; Burgard L y Thrun, S. Principles of Robot Motion. Recuperado de [30]

Con esto, el problema de planeación de movimiento para un robot circular se convierte en un problema de planeación de movimiento para un punto en el espacio de configuraciones. En ciertas ocasiones es más fácil conceptualizar el movimiento de un punto a través del espacio que de un cuerpo con volumen. [30] [31]

2.14 Dimensión del espacio de configuraciones

Los robots en la realidad son modelados (generalmente) como un grupo de cuerpos rígidos conectados por articulaciones, o por un único cuerpo rígido

(mayoría de robots móviles). El robot para utilizar en este trabajo es un cuerpo rígido que puede trasladarse y rotar sobre un plano. Si se tienen tres puntos A, B y C fijos sobre el cuerpo del robot, y se escoge libremente la posición del punto A, al momento de escoger la posición del punto B se está sujeto a que se debe mantener una distancia $d(A,B)$, desde A.

$$d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \tag{2.6}$$

Según (2.*6),* el punto B debe de estar ubicado a lo largo de una circunferencia de radio $d(A,B)$, centrado en (x_A, y_A), y el único parámetro arbitrario es el ángulo θ entre A y B. Ahora, para escoger el punto C se está sujeto a dos restricciones:

$$d(A,C) = \sqrt{(x_A - x_c)^2 + (y_A - y_c)^2} \tag{2.7}$$

$$d(B,C) = \sqrt{(x_B - x_C)^2 + (y_B - y_C)^2} \tag{2.8}$$

Por lo que automáticamente, al escoger el punto (x_A, y_A, θ) quedan definidos los demás puntos, y así, quede una buena representación de la configuración dada en ese punto. [30] [31]

Dado que el robot puede trasladarse en un plano (x, y) y rotar a lo largo de una circunferencia *S* sobre este, se dice que la dimensión del espacio de configuraciones es de $R^2\ x\ S^1$. Estos conceptos quedan a discusión, puesto que se estima un modelo simplificado del robot, asumiendo su región de inclusión como una circunferencia, lo que resulta más práctico debido a que no se toma en cuenta la dinámica propia del robot, sino que se asegura que todo movimiento estará excluido de una colisión, así como lo muestra la siguiente figura:

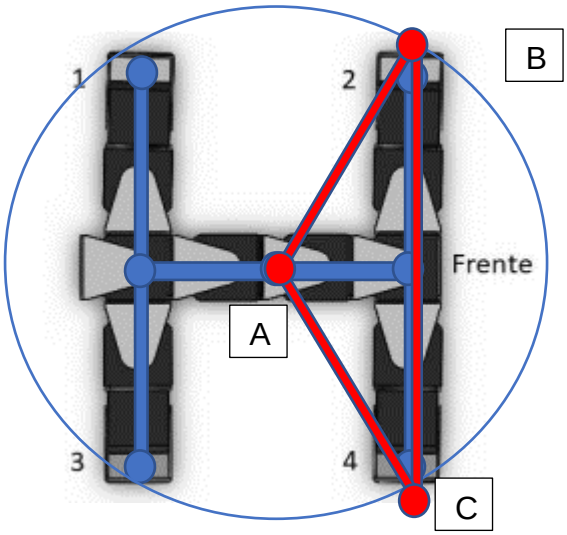


Figura 2. 11 Modelo simplificado del robot

Claro está, que si se desean resultados más exactos a la forma en la que se comporta el robot en la realidad, se puede utilizar un modelo más complejo (ver Figura 2.12), lo cual aumentaría las dimensiones del espacio de configuraciones, lo que significa un aumento en el número de restricciones y con esto la complejidad computacional.

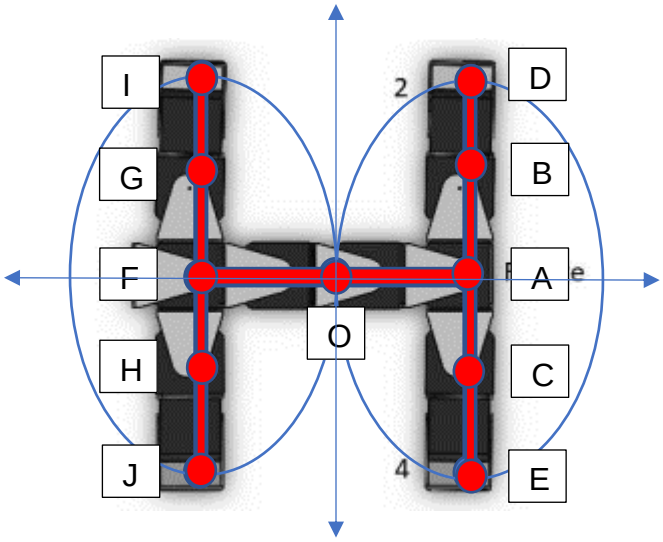


Figura 2. 12 Modelo desarrollado del robot

Con el modelo presentado en la Figura 2.12, se deben cumplir las siguientes restricciones una vez se defina el punto O, (x_o, y_o, θ_{OA}) y el punto D, (x_D, y_D, θ_{AD}) (nótese que θ_{OA} es el ángulo entre O y A, que a su vez es el ángulo entre F y O):

$$d(A,B) = d(A,C) = d(B,D) = d(C,E)$$

Estas restricciones son equivalentes puesto que los módulos que conforman el robot son homogéneos.

Si se observa, el segmento ABD es una cadena cinemática de articulaciones, por lo que una vez obtenida la posición del punto A, y con el punto D previamente definido, se obtiene un problema de cinemática inversa para hallar el ángulo entre A y B, y el ángulo entre B y D, y, con ciertas consideraciones geométricas y por la secuencia de locomoción del sistema (ver Figura 2.14), se pueden generar los siguientes enunciados:

- ***Enunciado 1:*** *El sistema a la izquierda de O es un reflejo de la parte derecha en el eje vertical relativo.*

- ***Enunciado 2:*** *El sistema ACE y FHJ son un reflejo del sistema ABD y FGI respectivamente, en el vértice de un eje coordenado que pasa por A y F respectivamente (ver Figura 2.13)*

Si se analiza, estos enunciados son válidos si se dispone de un segundo eje coordenado, relativo al sistema, lo que permite definir la orientación general del robot, y si está sujeto a cierta rotación de la sección central (curvatura). Ejemplo de esto se puede observar en la siguiente figura:

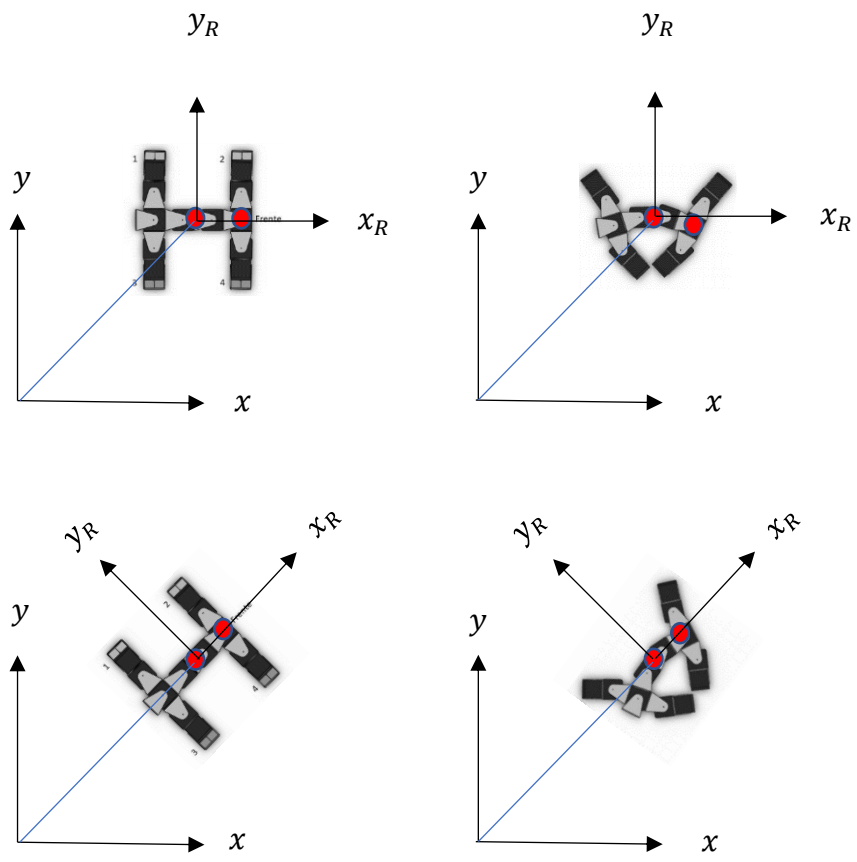


Figura 2. 13 Representación de la condición de validez de los enunciados.

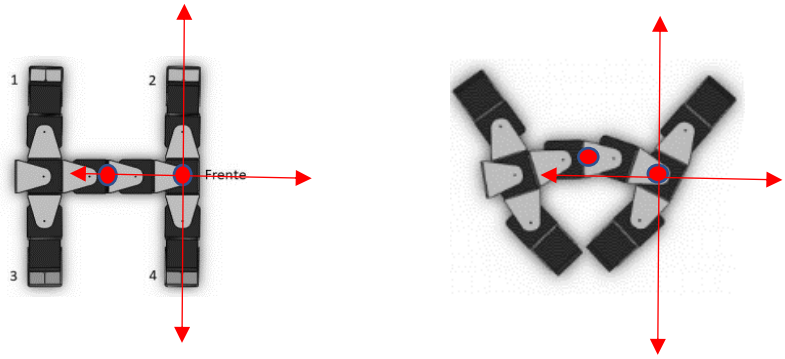


Figura 2. 14 Representación del enunciado 2

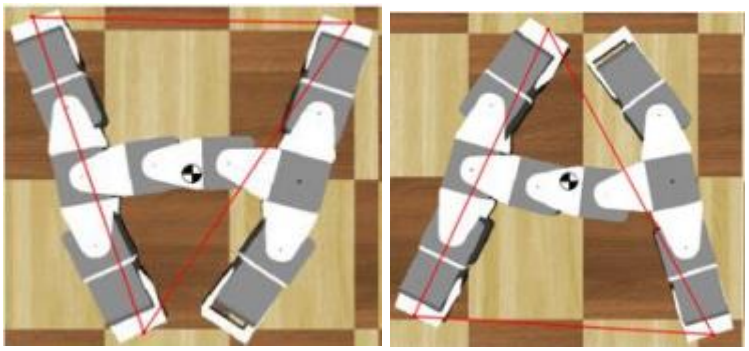


Figura 2. 15 Representación del enunciado 1.

Cruz, V.(2018). Simulación e implementación de arquitectura cuadrúpeda utilizando sistema robótico modular MECABOT.[Figura] Recuperado de [45]

Cabe notar que ya no es suficiente la sola definición del punto A, puesto que la distancia AF, aunque varía muy poco, no es del todo constante.

Se estima que con la definición del punto O, (x_o, y_o, θ_{OA}) y el punto D, (x_D, y_D, θ_{AD}), se puede hallar los punto A y B (por método de cinemática inversa), se refleja según la Figura 2.13 para hallar los puntos C y E, y la unión DBACE se refleja según la Figura 2.14 para hallar la unión IGFHJ. Quedando así definida en su totalidad, la configuración del robot.

Ya definido un modelo más complejo, se observa que las dimensiones el espacio de configuraciones se duplica, quedando como resultado un espacio de $R^4 \, x \, S^2$, puesto que para representar una configuración es necesaria la posición de dos puntos ($R^2 \, x \, R^2$) y su orientación ($S^1 \, x \, S^1$).

2.14.1 Representación Matricial

Resulta conveniente la representación de la posición y orientación de un cuerpo rígido utilizando una matriz *m x m*. Una de las ventajas que presenta esta representación es que se pueden multiplicar para obtener otra matriz que representa otra configuración en el espacio (siempre y cuando se cumplan un grupo de ecuaciones de igualdad de forma que se cumplan las restricciones planteadas por el espacio de trabajo).

Para [30], la matriz *SE (n)* (n= 2 o 3) representa la posición y orientación de un cuerpo rígido relativo a un eje de referencia fijo. Si:

$$p = [p1 \quad p2 \quad p3]^T \in R^3$$

Es el vector que va desde el origen del eje estacionario hasta el eje de referencia sobre el cuerpo, y *R* es una matriz de rotación, entonces se representa la posición y la orientación como una matriz de transformación *T*:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in SE\,(2)$$

Dado el eje de referencia fijo en la Figura 2.15, se obtiene una matriz general para la aplicación presente:

$$Tq_i, q_j = \begin{bmatrix} cos\theta & -sin\theta & (x_j - x_i) \\ sin\theta & cos\theta & (y_j - y_i) \\ 0 & 0 & 1 \end{bmatrix}$$

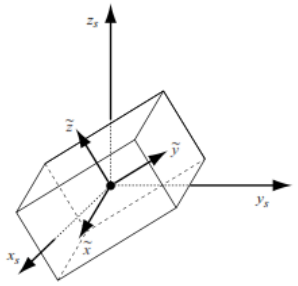


Figura 2. 16 Eje de referencia fijo (x, y, z). Eje de referencia del cuerpo (x˜,y˜,z˜)

Choset, H; Lynch, K; Hutchinson, S; Kantor,G; Burgard L y Thrun, S. Principles of Robot Motion.[Figura] Recuperado de [30]

Donde θ es el ángulo entre x y $\tilde{x}$. Puesto que el sistema en cuestión solo presenta rotación a lo largo de un eje, la parametrización de la matriz de rotación no es

necesaria, puesto que los valores de los ángulos respecto a los ejes X y Y son constantes.

Según el trabajo presentado en [30], la representación matricial puede ser utilizada para:

- Representar configuraciones de un cuerpo rígido.

- Cambiar el eje de referencia de la representación de una configuración o un punto.

- Mover una configuración o un punto.

En el trabajo presente, se utilizará para la representación de configuraciones, por lo que la matriz será denominada *cuadro.* [30]

2.15 Planificadores basados en mapas

Los siguientes métodos exploran el espacio libre de forma incremental mientras buscan por un camino que logre conectar el punto inicial con el punto objetivo. [30]

En las siguientes secciones solo se definirá el funcionamiento y las características de los planificadores basados en mapas, en relación con el objetivo de este trabajo, por esto, los métodos y estrategias de programación serán excluidos de este capítulo.

2.15.1 Roadmaps

La idea general de este método es el de capturar la conectividad del espacio libre del robot en C, en la forma de una red de curvas unidimensionales. Una vez obtenido, un *roadmap Rm,* es utilizado como un conjunto de caminos estandarizados, por lo que el problema de la planificación se reduce a conectar la configuración inicial y final a través de *Rm.* [31]

Para [30], un *roadmap* debe mantener las siguientes características:

- **Accesibilidad:** Existe un camino desde $q_{start} \in C_{free}$ a algún $q'_{start} \in Rm$.

- **Desprendimiento:** Existe un camino desde algún $q'_{goal} \in Rm$ hasta $q_{goal} \in C_{free}$.

- **Conectividad:** Existe un camino en Rm entre q'_{start} y q'_{goal}.

A continuación se desarrollarán dos tipos de *roadmaps,* los mapas de visibilidad, del cual se trabajarán los grafos de visibilidad y los métodos por retracción, de los cuales se trabajarán los diagramas generalizados de Voronoi.

2.15.1.1 Grafos de visibilidad

Este método consta en la construcción de un camino semi-libre a partir de líneas que conectan la configuración inicial y la configuración final a través de vértices en el espacio de configuraciones de obstáculos. Ejemplo de esto puede ser observado en la siguiente figura:

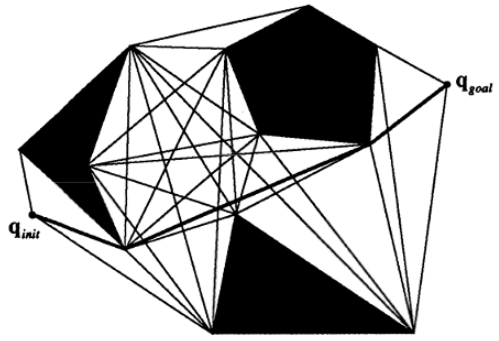


Figura 2. 17 Grafo de visibilidad de una región con obstáculos poligonales simples. Nótese que los bordes de los obstáculos también hacen parte de las conexiones entre vértices del camino a seguir.

Latombe, J. Robot Motion Planning. [Figura]. Recuperado de [31]

Existe cierta consideración al momento de realizar un grafo de visibilidad *G* de un espacio de configuraciones *C*, esta es: Los vértices son visibles si, y solo si existe una línea que los conecta a través de C_{free} o si son los vértices de un borde. [30] [31] [34]

Según Lozano-Perez y Wesley (1979), un algoritmo para la creación de grafos de visibilidad requiere de un tiempo $O(n^3)$, en donde *n* es el número de vértices en el espacio de obstáculos. [30]

Con la definición de un grafo de visibilidad, se propone que es suficiente para un planificador considerar únicamente el conjunto de líneas que forman un camino semi-libre, y si este existe, contiene el camino más corto (según la métrica Euclidiana en R^2). [31] [34]

2.15.1.2 Diagramas generalizados de Voronoi

Este método es desarrollado en los métodos de planificación por retracción, los cuales consisten en el mapeo continuo del espacio libre del espacio de

configuraciones, en curvas unidimensionales ubicadas dentro del mismo espacio libre. [30] [31] [34] [35]

Un diagrama generalizado de Voronoi es conformado por el conjunto de puntos en donde la distancia a los dos obstáculos más cercanos es la misma. Una región de Voronoi F_i es el grupo de puntos más cercanos a un obstáculo. Así, una región se define como:

$$F_i = \{q \in C_{free} \,|\, d_i(q) \leq d_h(q), \forall h \neq i\} \tag{2.9}$$

En donde $d_i(q)$ es la distancia a un obstáculo QO_i desde q, es decir:

$$d_i(q) = min_{c \in CO_i} d(q, c) \tag{2.10}$$

Para la construcción de un diagrama generalizado de Voronoi (GVD, por sus siglas en inglés), [30] propone una superficie duo-equidistante, denominada $S_{ij} = \{x \in C | d_i(q) = d_j(q)\}$, dado que esta superficie atraviesa obstáculos, se restringe al grupo de puntos que son equidistantes a CO_i y CO_j y a su vez tienen a CO_i y CO_j como sus dos obstáculos más cercanos. Esta estructura restringida es denotada por:

$$F_{ij} = \{q \in S_{ij} \mid d_i(q) \leq d_h(q), \forall h\} \tag{2.11}$$

La unión de estas estructuras conforma el GVD:

$$GVD = \bigcup_i \bigcup_j F_{ij} \tag{2.12}$$

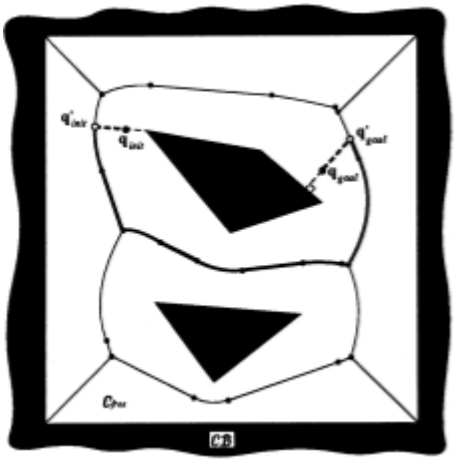


Figura 2. 18 GVD de una región con obstáculos poligonales simples limitada por fronteras.

Latombe, J. Robot Motion Planning. [Figura]. Recuperado de [31]

Como se observa en la Figura 2.18, el GVD cumple con las propiedades de accesibilidad, desprendimiento y conectividad.

2.15.2 Descomposición en celdas

Los métodos por descomposición en celdas consisten en descomponer el espacio libre del robot en regiones más simples denominadas *celdas,* de forma tal que un camino entre dos configuraciones cualquiera dentro de su respectiva celda pueda ser obtenido. Para este método se requiere de la construcción de una gráfica que represente la relación de adyacencia entre celdas, esta gráfica se conoce como la *gráfica de conectividad.* Los nodos en esta gráfica se obtienen a partir del espacio libre, y dos nodos se conectan si, y solo si sus celdas correspondientes son adyacentes. Al final de la búsqueda se obtiene una secuencia de celdas denominadas *canal.* [30] [31] [36]

Para [31], dentro de los métodos por descomposición en celdas se encuentran dos categorías principales:

- **Descomposición exacta:** Descomposición del espacio libre en celdas cuya unión es exactamente (haciendo referencia al problema de planificación y no al problema físico real) el espacio libre. Los límites de una celda son definidos por alguna característica de criticalidad como por ejemplo un cambio repentino en las restricciones de movimiento del robot (e. g. La presencia de un obstáculo).

- **Descomposición aproximada:** Estos métodos producen celdas con forma determinada (p. e. Rectángulos) cuya unión está estrictamente definida en el espacio libre. Los límites de una celda aproximada no representan discontinuidad y no tienen significado en físico.

Ejemplos de una descomposición exacta y aproximada se muestran a continuación:

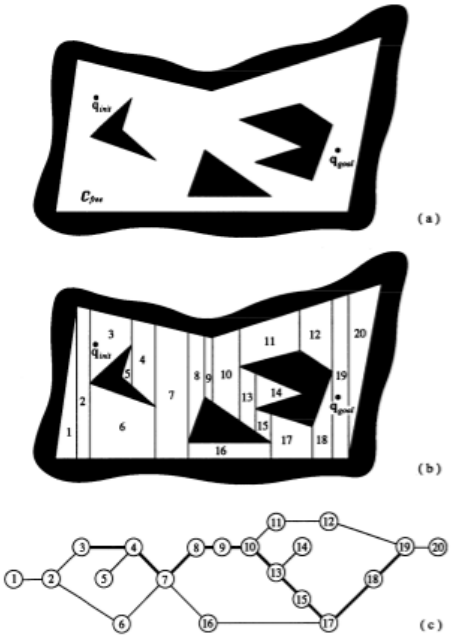


Figura 2. 19 a) Espacio libre limitado externamente por un polígono, e internamente por tres polígonos, b) Descomposición del espacio libre en celdas triangulares y trapezoidales, c) Construcción de la gráfica de conectividad que representa la adyacencia entre celdas.

Latombe, J. Robot Motion Planning. [Figura]. Recuperado de [31]

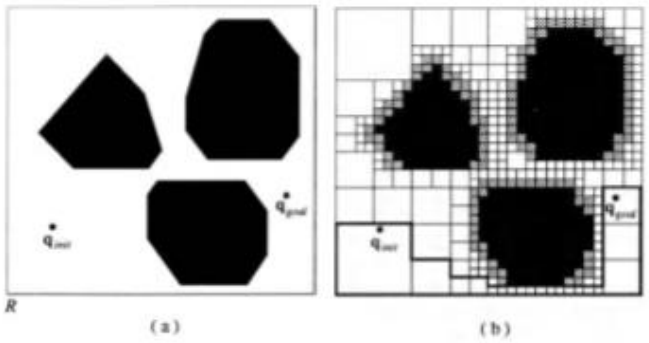


Figura 2. 20 a) Espacio libre limitado externamente por un rectángulo e internamente por tres polígonos, b) El rectángulo R se descompone en cuatro rectángulos idénticos, si el interior de un rectángulo yace completamente en espacio libre, o en espacio CO ya no es descompuesto, de no ser así es recursivamente descompuesto hasta que se obtenga alguna resolución definida.

Latombe, J. Robot Motion Planning. [Figura]. Recuperado de [31]

Dado que se dispongan de técnicas de búsqueda y de computación numérica exacta apropiadas, la descomposición en celdas exactas es más completa (garantizan hallar un camino libre cuando existe, e indican error si no). Los métodos de descomposición aproximada no son del todo completos, pero pueden ser configurados para que la precisión de la aproximación sea arbitrariamente pequeña (a cambio de un tiempo de computo mayor), por lo que estos métodos se

dicen son completos por resolución. Por otro lado, exceptuando algunos casos simples, los métodos exactos son matemáticamente más complejos, por lo que los métodos aproximados son más fáciles de implementar. [31] [30]

Para [30] y [36], la descomposición exacta, en el peor de los casos, tardaría alrededor de $O(n^4 \log(n))$, en donde n es el número de vértices en los obstáculos, para realizar la descomposición. Esto podría ser reducido utilizando métodos de barrido y por consideración de eventos, resultando, en el mejor de los casos, en $O(n\,(\log n)^4)$. La descomposición aproximada, considerando que es matemáticamente más simple, espera tener un tiempo menor que los métodos exactos, que aumenta a medida que aumenta la resolución.

2.15.3 Campos potenciales

Estos métodos se basan en funciones potenciales, de valor real y diferenciables denominadas $U = R^m \rightarrow R$, el valor de estas funciones puede ser visto como energía, por lo que el gradiente de estas es fuerza. El gradiente ∇U es un vector que apunta en la dirección que incrementa, localmente, a *U*. Se utiliza para crear un campo vectorial que asigna un vector a cada punto dentro del espacio. Este método dirige el robot como si fuera una partícula desplazándose dentro de un campo de vectores de gradiente. Esto puede ser representado como fuerzas actuando sobre una partícula cargada positivamente (robot), la cual es atraída hacia el objetivo (cargado negativamente). Los obstáculos tienen carga positiva, lo que forma una fuerza repulsiva dirigiendo el robot lejos de ellos. La combinación de fuerzas de atracción y repulsión busca dirigir el robot desde la ubicación de inicio hasta el objetivo evadiendo obstáculos. [30] [31] [32]

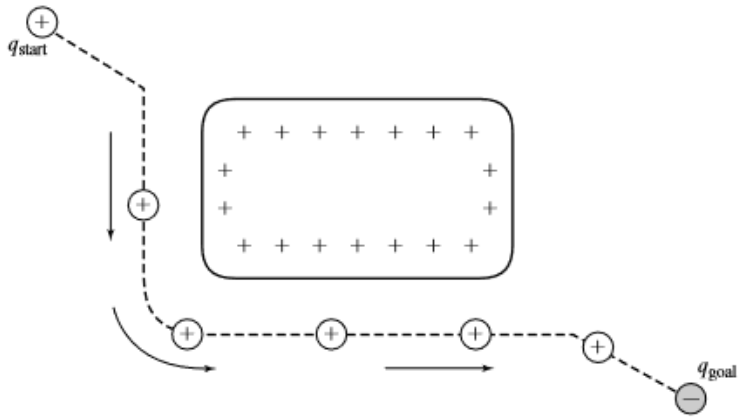


Figura 2. 21 La carga negativa atrae al robot, la carga positiva lo repele, resultando en un camino que rodea el obstáculo.

Choset, H; Lynch, K; Hutchinson, S; Kantor,G; Burgard L y Thrun, S. Principles of Robot Motion. .[Figura]. Recuperado de [30]

Las funciones potenciales pueden ser vistas como un ambiente en el que el robot va de un estado de alto nivel a uno de bajo nivel, siguiendo el gradiente negativo de la función potencial, esto es denominado *descenso de gradiente*. [30]

$$\dot{c}(t) = -\nabla U(c(t)) \qquad (2.13)$$

El robot terminará de moverse cuando encuentre un punto en el que el gradiente se vuelva nulo, es decir un estado q* en donde $\nabla U(q^*) = 0$. Este punto es denominado *punto crítico,* y puede ser un máximo, mínimo o un punto de ensilladura. Para determinar qué tipo de punto crítico es, se utiliza la segunda derivada, también conocida como la matriz Hessiana para funciones de valor real. No es necesario la determinación de estos puntos en métodos de descenso de gradiente, porque generalmente el movimiento del robot termina en un mínimo local. Dado que se disminuye el valor de U, no se puede llegar a un máximo local, a no ser que el robot empiece en uno, en dicho caso, la más mínima perturbación induce movimiento en el robot, liberándolo del punto crítico. Llegar a un punto de silla también es poco probable puesto que son inestables. Los mínimos locales por otro lado son bastante estables, puesto que en caso de que se dé una perturbación, el campo regresará el robot al mínimo local, por esto, el único punto (e. g.) en donde el robot puede terminar es en un mínimo local. Esto presenta un problema, puesto que dicho punto no siempre puede resultar siendo el punto objetivo, por lo que se dice que no todas las funciones potenciales llevan a un planificador completo. [30] [31]

Para [30], una de las funciones potenciales más simples es la de atracción/repulsión, en donde el objetivo atrae al robot mientras los obstáculos lo repelen, la combinación de estos efectos genera un camino libre de obstáculos. Puede ser representado como:

$$U(q) = U_{att}(q) + U_{rep}(q) \qquad (2.14)$$

Para el potencial de atracción U_{att}, se requiere que cumpla ciertas condiciones, la primera es que incremente (de forma monótona) con la distancia al objetivo (es decir, a mayor distancia mayor fuerza ejerce el campo). Existen varias funciones que cumplen este criterio, pero también se requiere que sea diferenciable en todos sus puntos para que no se presenten inconsistencias en el modelo (p. e. El gradiente de la función de potencial cónica es inversamente proporcional a la distancia, por lo que llegando al objetivo se presenta una incertidumbre). Por esto, [30] propone una función que crece cuadráticamente con la distancia:

$$U_{att}(q) = \frac{1}{2}\xi d^2(q, q_{goal}) \tag{2.15}$$

En donde ξ es un parámetro de escala para el efecto del potencial de atracción. El gradiente resulta:

$\nabla U_{att} = \frac{1}{2}\xi \nabla d^2(q, q_{goal})$

$$\nabla U_{att} = \xi(q - q_{goal}) \tag{2.16}$$

Una alternativa que expone es la de combinar un potencial cónico con uno cuadrático, de forma que al alejarse q de q_{goal}, el potencial cónico atraiga al robot con una velocidad segura (puesto que el potencial cuadrático crece sin límites a medida que aumenta la distancia, resultando en una velocidad muy alta, provocando así un *overshoot*), y que al acercarse al objetivo, el potencial cuadrático atraiga también al robot. Para esto es necesario definir los límites en los que el planificador cambia de potencial cónico a cuadrático.

$$U_{att}(q) = \begin{cases} \frac{1}{2}\xi d^2(q, q_{goal}), d(q, q_{goal}) \leq d^*_{goal} \\ d^*_{goal}\xi d(q, q_{goal}) - \frac{1}{2}\xi (d^*_{goal})^2, d(q, q_{goal}) > d^*_{goal} \end{cases} \tag{2.17}$$

De esto se obtiene que el gradiente es:

$$\nabla U_{att}(q) = \begin{cases} \xi(q - q_{goal}), d(q, q_{goal}) \leq d^*_{goal} \\ \frac{d^*_{goal}\xi(q - q_{goal})}{d(q, q_{goal})}, d(q, q_{goal}) > d^*_{goal} \end{cases} \tag{2.18}$$

En donde d^*_{goal} es la distancia límite que indica el cambio entre potencial cónico y cuadrático. Obsérvese que en este punto, el potencial cónico es equivalente al potencial cuadrático, por lo que el gradiente está definido en todos sus puntos. [30]

El potencial de repulsión, por otro lado, depende de la proximidad del robot a un obstáculo, por lo que se define de la siguiente forma:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2, & D(q) \leq Q^* \\ 0, & D(q) > Q^* \end{cases} \tag{2.19}$$

Donde Q^* es un factor que le permite al planificador ignorar obstáculos que estén bastante alejados del robot, $D(q)$ es la distancia al obstáculo más cercano y η es el factor de escala. Con esto, el gradiente resulta:

$$\nabla U_{rep}(q) = \begin{cases} \eta\left(\frac{1}{Q^*} - \frac{1}{D(q)}\right)^2 \frac{1}{D^2(q)} * \nabla D(q), & D(q) \leq Q^* \\ 0, & D(q) > Q^* \end{cases} \qquad (2.20)$$

Dada la posibilidad de que existan puntos equidistantes a dos o más obstáculos, se redefine el potencial de repulsión en función de obstáculos individuales, por lo que cada obstáculo posee una función de potencial:

$$U_{rep_i}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d_i(q)} - \frac{1}{{Q_i}^*}\right)^2, & D(q) \leq {Q_i}^* \\ 0, & D(q) > {Q_i}^* \end{cases} \qquad (2.21)$$

En donde $d_i(q)$ es la mínima distancia entre q y el obstáculo. De esto, si se define cl como el punto sobre el obstáculo más cercano a q:

$$\nabla d_i(q) = \frac{q - c}{d(q, c)} \qquad (2.22)$$

Por lo que el gradiente del potencial de repulsión es:

$$\nabla U_{rep_i}(q) = \begin{cases} \eta\left(\frac{1}{d_i(q)} - \frac{1}{{Q_i}^*}\right)^2 \frac{1}{{d_i}^2(q)} * \nabla d_i(q), & d_i(q) \leq Q^* \\ 0, & d_i(q) > Q^* \end{cases} \qquad (2.23)$$

Y la función potencial queda definida como:

$$U(q) = U_{att}(q) + \sum_{i=1}^{n} U_{rep_i}(q) \qquad (2.24)$$

Cabe notar que estas funciones potenciales presentan el problema de mínimos locales distintos al objetivo, y aunque existen varios métodos de corrección, e incluso otro tipo de funciones potenciales que solo poseen un mínimo asegurado

en el objetivo, no serán presentados en este capítulo. Si al final del capítulo 3 se escoge este método, se entrará más en detalle.

2.16 Algoritmo de búsqueda

Solo el algoritmo de planeación no es suficiente para generar una trayectoria, lo que este presenta es un conjunto de configuraciones válidas del sistema sobre el mapa; con esto, hace falta un algoritmo que genere un subconjunto de dichas configuraciones, que de ser seguidas en orden, presentarán la trayectoria deseada. Dicho esto, se retoma lo presentado en la sección 2.11, en donde se menciona la planificación discreta, y como el manejo de un subconjunto de configuraciones (lista prioritaria) define el algoritmo en sí.

Dentro de la literatura existen varios algoritmos de búsqueda, en [32] están:

- ***Breadth-first* o Amplitud primero:** Basado en una arquitectura FIFO (*First in-First out)*, genera una frontera de búsqueda con crecimiento uniforme.

- ***Depth-first* o Profundidad primero:** Basado en una arquitectura LIFO (*Last in-First out),* genera una exploración agresiva de la transición entre estados.

- **Algoritmo Dijkstra:** Este algoritmo presenta el concepto de '*costos'* al momento de ir de un estado a otro, con esto, el plan óptimo es aquel cuya suma de costos sea menor a comparación de los demás planes posibles.

- **Algoritmo A*:** Como variación del algoritmo Dijkstra, este algoritmo también presenta una función de costo, la diferencia es que adiciona una función heurística al momento de calcular los costos, lo que permite una exploración más cerrada (no visita tantos estados).

Se reconoce que el algoritmo A* presenta mejores resultados a diferencia de los anteriormente mencionados, según [40], este algoritmo es la opción más utilizada para la generación de trayectorias dado que es bastante flexible y puede ser utilizado en un amplio rango de aplicaciones. Dado que existen otros algoritmos que no se exponen en este capítulo, y no son tomados en cuenta al realizar dicha consideración, queda abierto a discusión en trabajos futuros.

*2.16.1 Algoritmo A**

El algoritmo A* se basa en dos funciones de costo, una denominada *g(n),* que representa el costo exacto desde el punto actual a cualquier otro punto *n.* La función *h(n)* representa el costo estimado por una heurística para llegar del punto *n* al punto objetivo o punto final. El algoritmo, a lo largo de su ejecución busca balancear ambas funciones mientras se mueve del punto inicial al punto final. Por

cada iteración, escoge el punto *n* que presente el costo *f(n) = g(n) + h(n)* menor. [40]

La función heurística para implementar depende del tipo de ejercicio a realizar, por ejemplo, si el desplazamiento del sistema se da a partir del seguimiento de líneas horizontales y verticales, la función heurística en este caso podría ser la distancia Manhattan:

$$H(n) = \left|x_{start} - x_{obj}\right| + \left|y_{start} - y_{obj}\right| \qquad (2.25)$$

Esta heurística ignora movimientos diagonales y todo obstáculo que se encuentre a lo largo de esta diagonal.
Otra función heurística es la de distancia Euclidiana, esta es más exacta a diferencia de la distancia Manhattan puesto que favorece un camino en línea recta al objetivo, aunque es más lenta puesto que debe analizar un área mayor del espacio para encontrar un camino.[41]

$$H(n) = \sqrt{\left(x_{start} - x_{obj}\right)^2 + \left(y_{start} - y_{obj}\right)^2} \qquad (2.26)$$

CAPÍTULO 3
DESARROLLO DEL ALGORITMO EN WEBOTS ®

En este capítulo se presenta el método de selección del algoritmo a utilizar, así como los distintos métodos de procesamiento de imágenes necesarios para la generación de un mapa del espacio de configuraciones a partir del espacio de trabajo, posteriormente se identificarán las celdas presentes en dicho mapa para identificar los puntos por los cuales el sistema tiene la posibilidad de transitar. Una vez identificados, se implementará el algoritmo de búsqueda que permitirá obtener la secuencia de puntos que generarán la trayectoria deseada.

3.1 Selección del algoritmo

En esta sección se evaluarán los planificadores desarrollados en la sección inmediatamente anterior a partir de los aspectos listados a continuación y expresados en la sección **2.10,** según su importancia. A cada uno se le asignará un sistema de puntuación de 1 a 5 según su correspondencia con la aplicación a desarrollar. Esto corresponde a parte de la caracterización del planificador y las propiedades del algoritmo.

- **Completitud:** Hace referencia a que el algoritmo es capaz de identificar un camino, si lo hay, e indicar error si no lo hay. También se toma en cuenta la completitud probabilística y de resolución.

  1: No es completo.
  3: Es completo probabilísticamente o por resolución.
  5: Es completo.

- **Seguridad:** Hace referencia a que el camino hallado es libre o semilibre, es decir, la preservación de la integridad del robot.

  1: El camino choca con un obstáculo.
  3: El camino es semi-libre.
  5: El camino es libre

- **Complejidad:** Hace referencia a la complejidad matemática y de programación.

  1: Es bastante complejo
  3: Requiere de trabajo pero no es muy complejo
  5: No es complejo

- **Optimización:** Hace referencia a si es necesario algún método de optimización posterior a lo que se ha expuesto en la sección 3.8 debido a casos excepcionales o carga computacional.

1: Requiere de varios métodos de optimización
3: Requiere de métodos de optimización simples en implementación
5: La implementación de métodos de optimización es opcional.

- **Estabilidad:** Hace referencia a que la trayectoria descrita no oscile a lo largo de sí misma y que la más mínima perturbación no afecte el resultado.

1: La trayectoria es muy inestable
3: La trayectoria es estable
5: La trayectoria es muy estable

A partir de estos elementos se propone la siguiente matriz de puntaje:

		Completitud		**Seguridad**		**Complejidad**	
		Descripción	**Pt**	**Descripción**	**Pt**	**Descripción**	**Pt**
Roadmaps	**Grafos de visibilidad**	Por definición es suficiente para hallar un camino, y posee el más corto.	5	El camino es semi-libre.	3	Su programación posee algo de complejidad en su desarrollo	3
	Diagramas generalizados de Voronoi	Es capaz de hallar un camino, pero no es el más corto.	3	El camino es libre.	5	El algoritmo requiere de un desarrollo matemático muy extenso	1
Descomposición en celdas	**Celdas exactas**	Garantizan la completitud y se puede encontrar el más corto de los posibles.	5	El camino es libre.	5	Su programación posee algo de complejidad en su desarrollo	3
	Celdas aproximadas	Son completos por resolución.	3	El camino es semi-libre.	3	Su programación posee algo de complejidad en su desarrollo	3

Campos potenciales	Son completos por probabilidad.	3	El camino es libre.	5	Su programación posee algo de complejidad en su desarrollo	3

Tabla 3. 1 Matriz de comparación para los aspectos de completitud, seguridad y complejidad. Continúa

		Optimización		**Estabilidad**	
		Descripción	**Pt**	**Descripción**	**Pt**
Roadmaps	**Grafos de visibilidad**	La implementación de métodos de optimización es opcional.	5	El camino hallado es estable.	5
	Diagramas generalizados de Voronoi	Se requiere la implementación de varios métodos de optimización.	1	El camino hallado presenta oscilaciones.	3
Descomposición en celdas	**Celdas exactas**	La implementación de métodos de optimización es opcional.	5	El camino hallado es estable.	5
	Celdas aproximadas	Se requiere la implementación de algunos métodos de optimización.	3	El camino hallado es estable.	5
Campos potenciales		Se requiere la implementación de varios métodos de optimización.	1	El camino hallado presenta oscilaciones.	3

Tabla 3. 2 Matriz de comparación para los aspectos de optimización y estabilidad.

De acuerdo con lo expresado en la tabla 3.2, se obtienen los siguientes resultados:

Método de planificación de caminos	Puntaje total
Grafos de visibilidad	21
Diagramas generalizados de Voronoi	13
Celdas exactas	23
Celdas aproximadas	17
Campos potenciales	15

Tabla 3. 3 Resultados de la puntuación.

Con los resultados obtenidos en la tabla 3.3 el método de planeación de trayectorias a utilizar es el de celdas exactas.

Para el desarrollo de este algoritmo se presenta el siguiente diagrama de flujo:

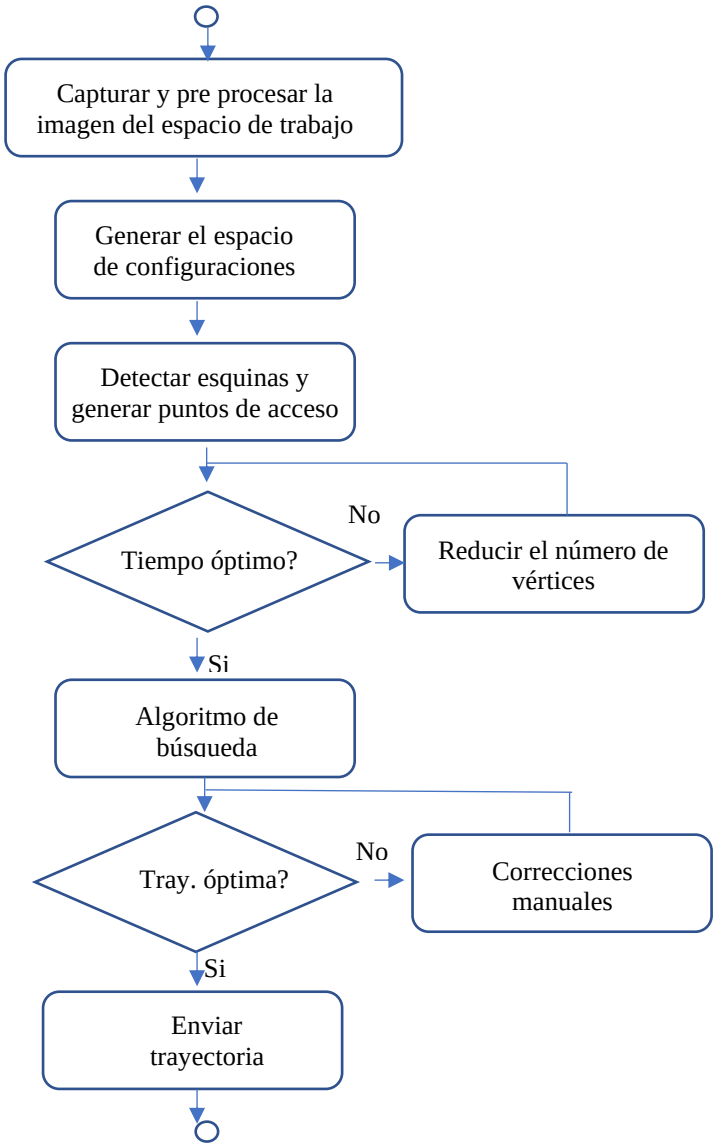


Figura 3. 1 Diagrama de flujo (Proceso general)

3.2 Procesamiento de imágenes

Para las pruebas iniciales se generará un espacio de trabajo sobre una arena rectangular, a la cual se le cambia la textura para obtener un fondo blanco. Para la cámara se utilizará el modelo propio de Webots® por lo que la resolución de las imágenes estará sujeta a aquella de esta cámara. Los obstáculos serán representados con cajas sin masa, esto resultará en elementos inamovibles. El espacio estará delimitado por una frontera fija. Ejemplo de esto se muestra en las siguientes figuras:

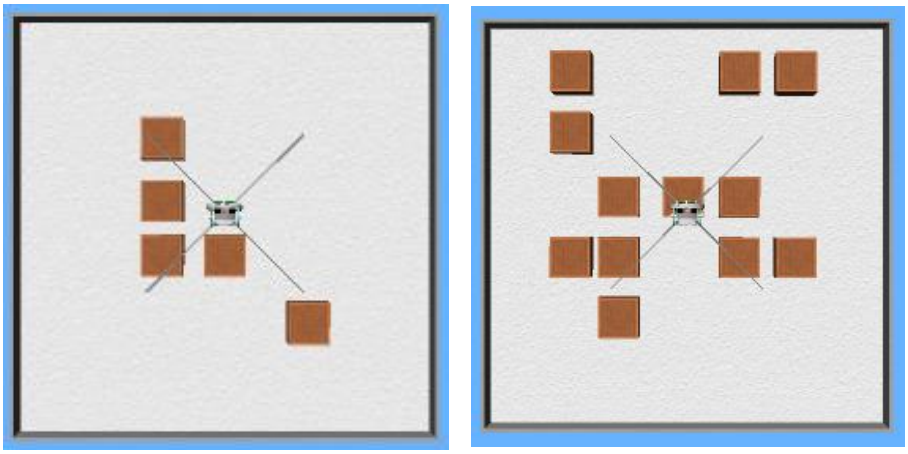


Figura 3. 2 a) (Izquierda) Espacio de trabajo con un arreglo de obstáculos simple. b) (Derecha) Espacio de trabajo con un arreglo mayor de obstáculos.

A continuación se muestra el diagrama de flujo a seguir:

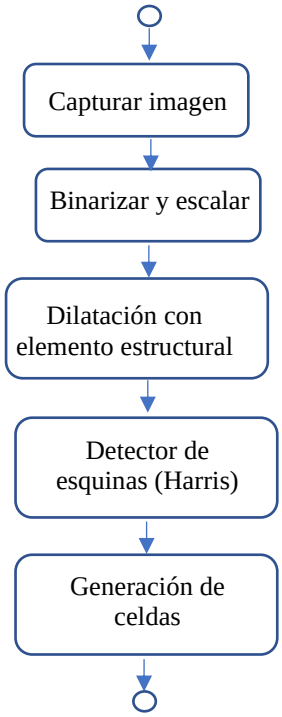


Figura 3. 3 Diagrama de flujo procesamiento de imágenes

Para la generación del mapa del espacio se requiere de imágenes binarias por lo que en esta sección se utilizará el comando *im2bw* para binarizar la imagen tomada, no se variarán los parámetros dado que con el límite de binarización predeterminado (0.5) es suficiente para obtener una imagen binaria correcta. No

se utilizará el comando recomendado por MATLAB® (*imbinarize*) dado que la versión del controlador en Webots® no la reconoce. [43]

Una vez obtenida la imagen binaria, se redimensiona a 240p x 240p debido a que la resolución de origen es muy baja (35p x 35p), posteriormente se procede a la generación del espacio de configuraciones. Para esto, se realizó un barrido por píxeles sobre toda la imagen y se determinaba cuales correspondían a obstáculos (si es un 0 en la matriz de la imagen). Si el píxel resultaba ser parte de un obstáculo, se sobreponía una matriz de ceros de tamaño $((2 * r) + 1)x\left((2 * r) + 1\right)$ con centro en dicho píxel, donde *r* es el radio de la circunferencia que encierra el sistema (ver **figura 2.11**) convertida a su equivalente en píxeles a partir de la medida horizontal del espacio 'real' indicada por consola (este valor es arbitrario en esta fase de la programación, se utilizó un valor de 3 m).

Cuando se ejecutaba el programa se encontró que la imagen resultante era de un tamaño mayor a la inicial, por lo que en iteraciones posteriores se generaba un error de indexación. Para solventar este problema, se generaron sentencias condicionales que variaban el tamaño de la matriz a sobreponer según la posición del píxel evaluado respecto a los márgenes de la imagen.

Más adelante se observó que el algoritmo se ejecutaba en un tiempo mayor a 1 segundo (ver **tabla 3.4**) y los resultados obtenidos no eran muy precisos (comparado a lo expuesto en la **figura 2.10**) por lo que se planteó un cambio en el método de identificación de obstáculos. En vez de realizar un barrido a lo largo de toda la imagen se utilizó una herramienta de identificación de bordes. A los píxeles que hacen parte de estos, se les aplica la superposición de matriz manteniendo los condicionales dado que los bordes de la frontera retienen el problema mencionado anteriormente.

Evaluando los resultados obtenidos, se observa que el espacio de configuraciones no está en concordancia con lo expuesto en la sección 2.14, por lo que para mejorar los resultados, se decide cambiar el método de creación del espacio de configuraciones.

Se decide utilizar un 'elemento estructural morfológico' o 'vecindad de valor binario' para realizar una operación de dilatación sobre el espacio de trabajo. En esta operación, los pixeles de valor *true* son incluidos en la computación morfológica (para dilatación) y los de valor *false* no.
Dicho elemento estructural será un rectángulo de medidas (2*r) x (2*r). Se probaron otros elementos estructurales, sin embargo, el elemento rectangular presenta mejores resultados. [42]

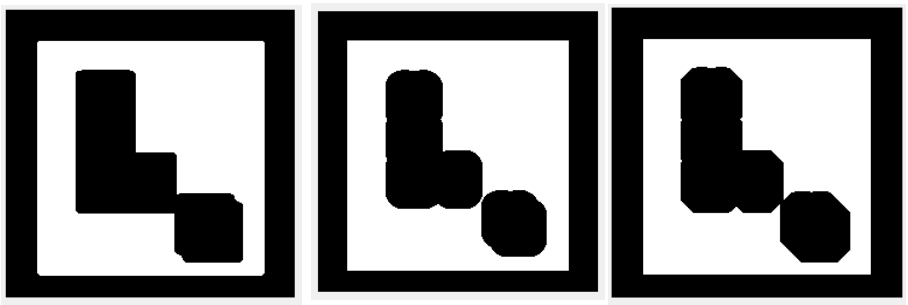


Figura 3. 4 a) (Izquierda) Elemento rectangular. b) (Centro) Elemento disco c) (Derecha) Elemento octogonal

		Tiempo de ejecución (s)	
Método		**Webots®**	**MATLAB®**
Barrido por píxel		>2	1.630918
Detección de bordes		>1	0.221759
Elemento estructural	Rectangular	2.146751	0.045575
	Disco	2.169929	0.012166
	Octagonal	2.314711	0.103294

Tabla 3. 4 Tiempos de ejecución de las variaciones del programa para generar el espacio de configuraciones. Nótese que se comparan los tiempos obtenidos a través del entorno Webots® con aquellos obtenidos directamente desde MATLAB®, esto con el propósito de dar a entender que desde el simulador Webots® se dan ciertas acciones adicionales (ej. como acceso a los archivos de raíz) que retrasan la ejecución del código. Estas mediciones se realizaron con la misma imagen (figura 4.1 a), del mismo tamaño y contabilizando el tiempo de ejecución de las mismas líneas de código. Las consideraciones siguientes serán realizadas a partir de los resultados obtenidos directamente desde MATLAB® puesto que con esta herramienta es la que se trabajará el modelo real.

De los tres elementos estructurales presentados, el elemento disco se ejecuta en un tiempo menor que el elemento rectangular, no obstante, como se expresa en la sección 3.1, la funcionalidad tiene mayor prioridad que el costo computacional.

3.3 Detección de esquinas (Detector Harris)

Una vez se disponga del espacio de configuraciones se procede a la generación de celdas, esto se logra proyectando los puntos clave de un objeto dentro del espacio hasta hallar otro objeto o frontera. Dado que los objetos dentro del espacio a trabajar consisten en formas geométricas, los puntos clave de estos son sus vértices.

Para la identificación de los vértices se utilizará el detector de características Harris, este consiste en un operador matemático que encuentra características independientes de rotaciones, escalas y variaciones de iluminación. Para [44], es bastante popular debido a su facilidad de implementación y velocidad de cómputo.

El objetivo de este detector es identificar "ventanas" (secciones) de la imagen que generan altas variaciones cuando se desplazan en pequeñas cantidades. Matemáticamente se da que:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \qquad (3.1)$$

Donde:

- *E* es la diferencia entre la ventana original y la ventana que se ha movido
- *u* es el desplazamiento en la dirección *x*
- *v* es el desplazamiento en la dirección *y*
- $w(x,y)$ es la posición de la ventana en (x,y). Es una máscara que asegura que solo la ventana deseada sea evaluada.
- I es la intensidad de la ventana en la posición (x,y)
- $I(x+u, y+v)$ es la intensidad de la ventana desplazada

Lo que se busca son ventanas que generen un valor de E alto, por esta razón se maximiza la diferencia de intensidades. Después de una expansión por series de Taylor y la expansión del término cuadrático se da que:

$$E(u,v) \approx \sum_{x,y} w(x,y)\,[u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2] \qquad (3.2)$$

Si:

$$M = \sum w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \qquad (3.3)$$

Donde M es la matriz de segundo momento.

Entonces:

$$E(u,v) = [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.4}$$

Con los eigen-valores de la matriz M se puede determinar la viabilidad de una ventana. Para esto se le asigna un puntaje denominado *R,* calculado para cada ventana. Todas las ventanas con un valor R mayor a un umbral determinado pueden ser consideradas como vértices. [44]

Dado que MATLAB® dispone de una función que implementa el operador Harris para la detección de esquinas, no es necesaria la programación de este. Los parámetros que utiliza se basan en el desarrollo matemático presentado. Su funcionamiento se observa en la siguiente imagen:

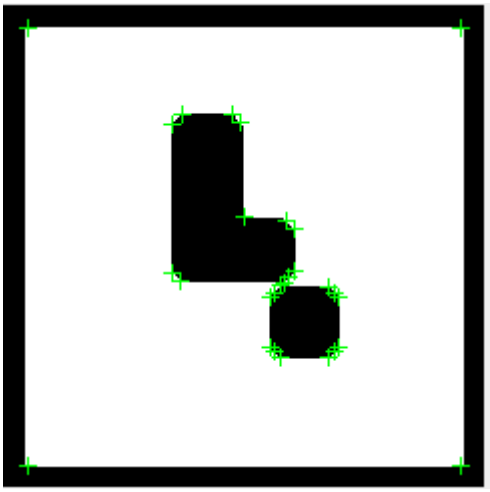


Figura 3. 5 Identificación de esquinas utilizando los comandos del toolbox de MATLAB®

Como se observa en la **figura 3.5** se presentan más vértices de los necesarios, de continuar así, se generarían celdas adicionales que resultarían en un retraso en el tiempo de ejecución del programa. Por esto, se propone seguir los siguientes pasos para solucionar este problema:

1. Obtener la distancia entre todos los vértices hallados.
2. Definir una distancia mínima aceptable
3. Obtener una lista de aquellos vértices cuya distancia entre sí sea menor a la distancia mínima aceptable
4. Elegir un vértice como referencia y eliminar aquellos que pertenezcan a la lista.

Una vez implementados se obtienen los siguientes resultados:

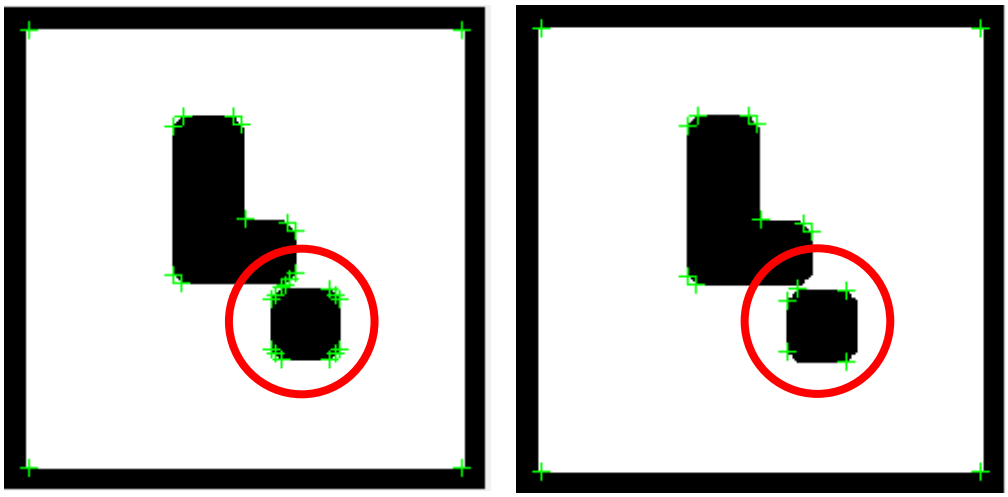


Figura 3. 6 Reducción en la cantidad de vértices para la generación de celdas, aunque la diferencia no sea tan notoria al principio, el tiempo de ejecución en la obtención de celdas pasó de 0.226525 s a 0.042908 s, significando una reducción del 81%.

Una vez reducido el número de vértices a utilizar, se busca, a partir de estos, los puntos de acceso a las celdas. La obtención de estos puntos se formula siguiendo el diagrama de flujo que se muestra a continuación para cada vértice válido (un vértice válido implica que existe espacio libre ya sea hacia arriba o hacia debajo de este):

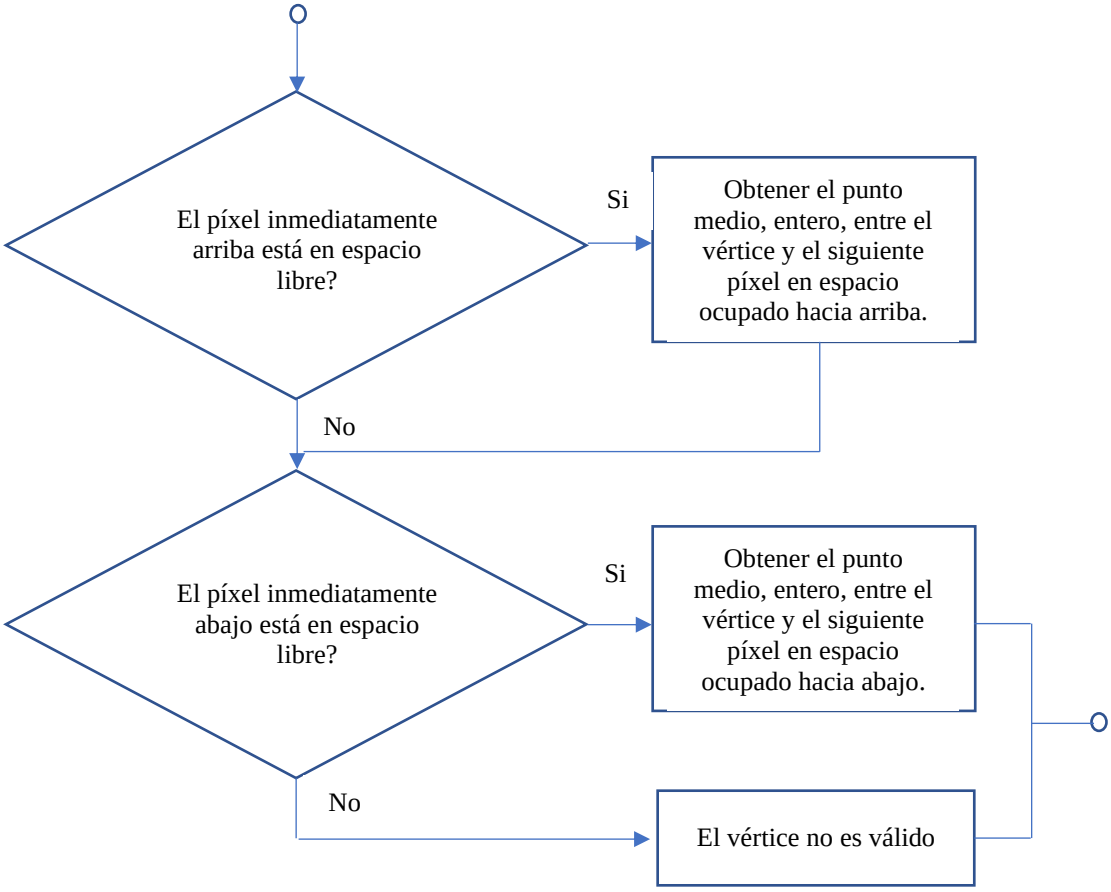
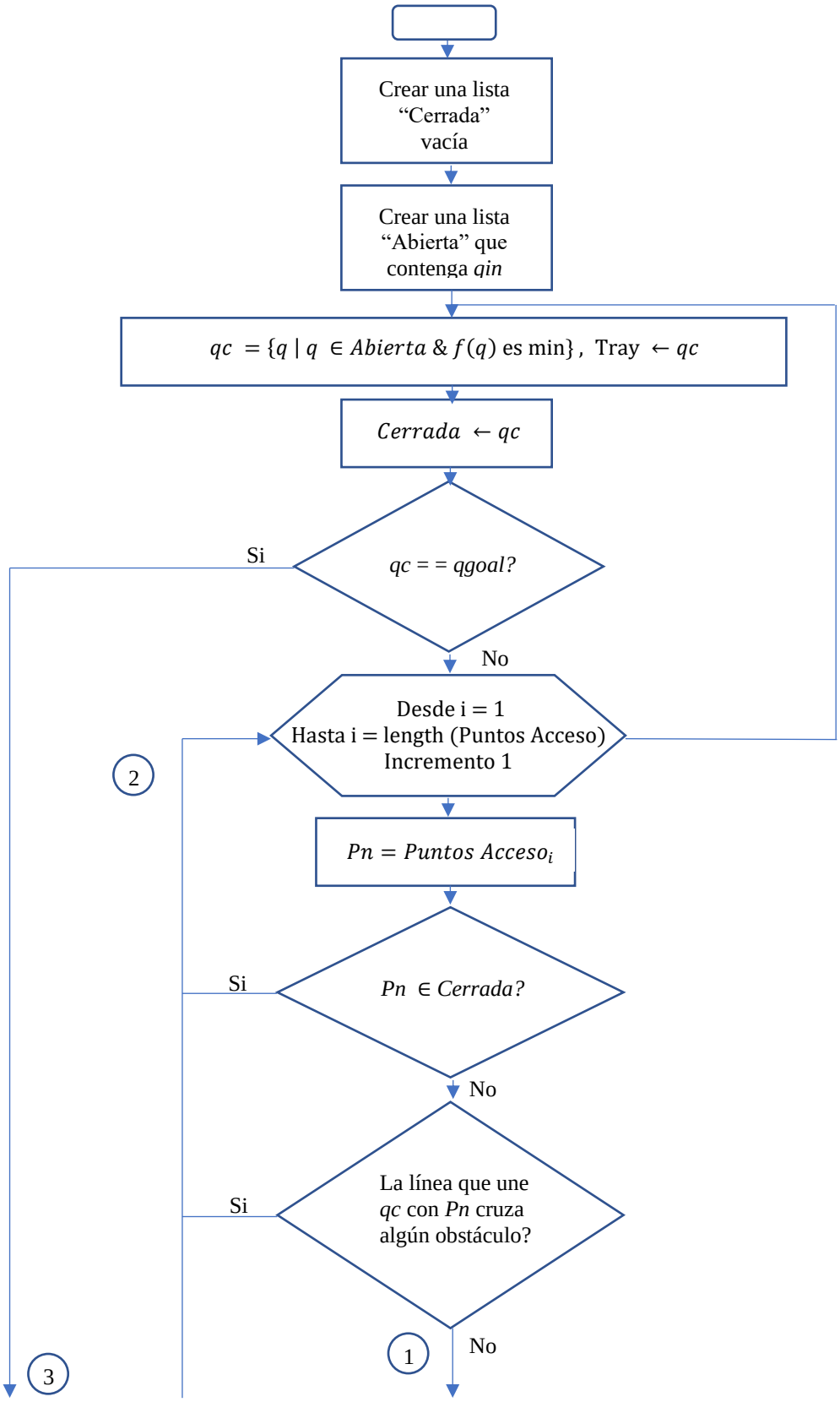


Figura 3. 7 Diagrama de flujo para la obtención de los puntos de acceso a las celdas.

3.4 Algoritmo de búsqueda

Una vez obtenidos los puntos de acceso a las celdas se programa el algoritmo de búsqueda (en este caso A*) con el cual se generará una secuencia de puntos de acceso que permitirá el desplazamiento seguro del sistema a través del espacio hacia un punto objetivo.

Para esto, se presenta el siguiente diagrama de flujo:


Crear una lista "Cerrada" vacía
Crear una lista "Abierta" que contenga qin
$qc = \{q \mid q \in Abierta \ \& \ f(q) \text{ es min}\}$, Tray $\leftarrow qc$
$Cerrada \leftarrow qc$
qc = = qgoal?
Si
No
Desde i = 1
Hasta i = length (Puntos Acceso)
Incremento 1
2
$Pn = Puntos\ Acceso_i$
Pn ∈ Cerrada?
Si
No
La línea que une qc con Pn cruza algún obstáculo?
Si
No
1
3

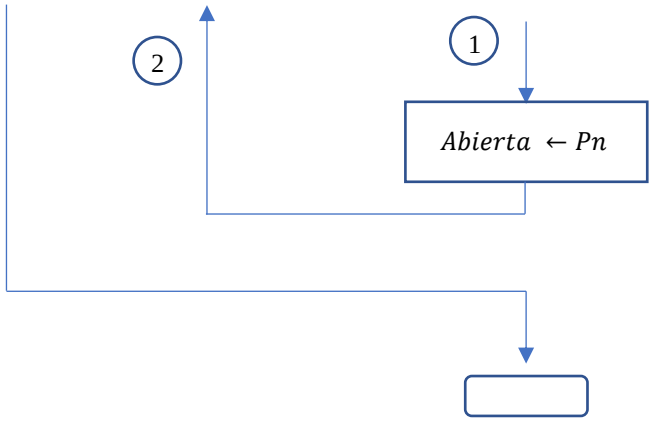


Figura 3. 8 Diagrama de flujo para la obtención de la trayectoria a seguir.

Los resultados obtenidos se muestran a continuación:

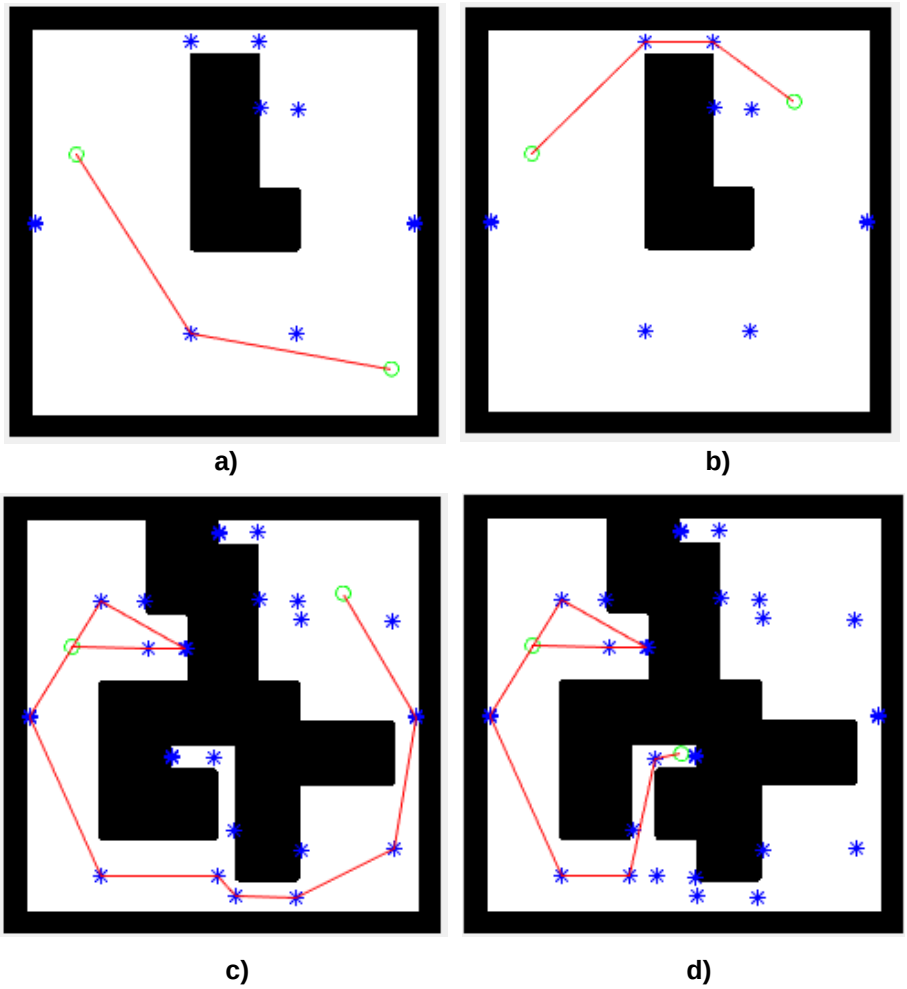


Figura 3. 9 a) Primer espacio de trabajo con un obstáculo simple. b) Primer espacio de trabajo con un punto objetivo diferente. c) Segundo espacio de trabajo con un obstáculo complejo. b) Segundo espacio de trabajo con un punto objetivo diferente. Resultados obtenidos en distintos espacios de trabajo con diferentes puntos objetivo.

Como se observa en la **figura 3.9,** el algoritmo encuentra una trayectoria hasta el punto objetivo a través de los puntos de acceso a las celdas, nótese que en algunos casos se genera una trayectoria cruzada, esto se debe a que algunos puntos, aunque no sean eficientes, se encuentran en espacio libre y más cercanos al punto objetivo que otros puntos, por lo que su valor *f* (ver sección 2.16.1) es menor, lo cual lo convierte en parte de la trayectoria.

Debido a lo mencionado anteriormente se programó una secuencia de corrección manual que le permite al usuario elegir qué puntos de la trayectoria obtenida desea descartar. Resultado de esto se observa en las siguientes figuras:

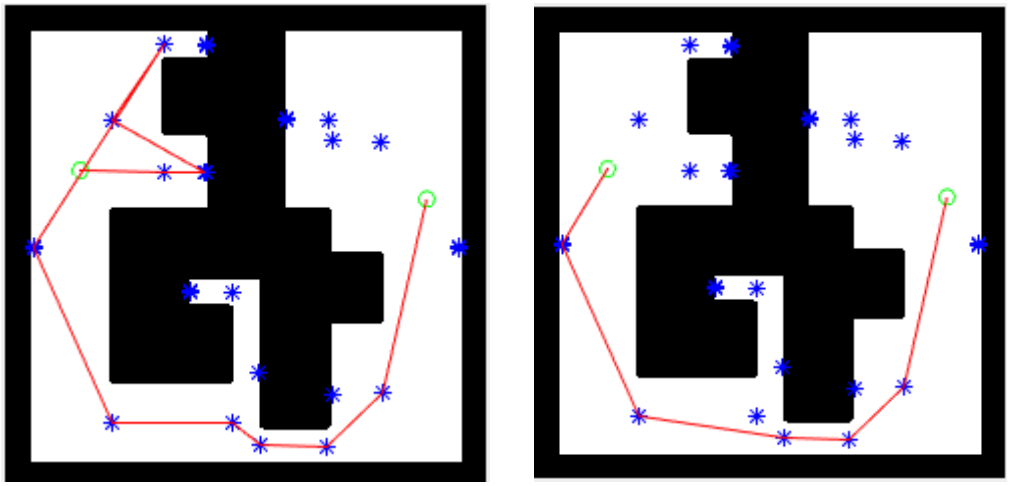


Figura 3. 10 Trayectoria sin correcciones (Izquierda). Trayectoria con correcciones (Derecha)

Cabe resaltar que el uso de la secuencia de corrección manual es completamente opcional, con o sin correcciones se llega al mismo punto, pero el uso de estas permite una distancia recorrida menor.

3.5 Simulación

Para el proceso de simulación se implementaron varios algoritmos, clasificados en dos grupos: Algoritmos de comunicación y algoritmos de locomoción.

3.5.1 Algoritmos de comunicación

Para la comunicación entre las dos entidades (cámara y cuadrúpedo) se dispuso de una topología emisor-receptor, siguiendo un protocolo de comunicación serial síncrono (el paso de simulación funciona como el reloj interno) siendo la cámara el emisor y el cuadrúpedo el receptor. La comunicación es unidireccional. Una vez el emisor envía los datos (posición del objeto verde y azul), en paquetes de 8 bits,

a baudios infinitos (esto quiere decir que es inmediata la transmisión), el receptor sigue el siguiente diagrama de flujo:

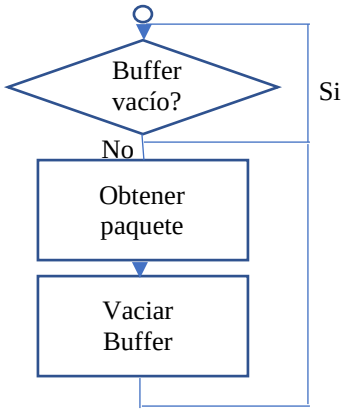


Figura 3. 11 Diagrama de recepción de datos

La primera vez que se ejecuta el algoritmo, el emisor enviará un paquete de datos que contiene la trayectoria final una vez determinada, después de esto procederá a enviar la configuración actual del robot.
La cámara dispone de la capacidad para identificar de objetos y al cuadrúpedo se le asignan tres puntos identificables tal como lo representa la **figura 3.3.** Para esto se incluyeron tres objetos esféricos a una distancia fija del centro, cada uno con un color distinto para ser identificados por la cámara y así disponer de la configuración actual del sistema, como lo muestra la siguiente figura:

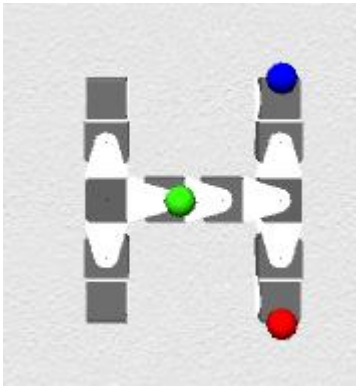


Figura 3. 12 Sistema final con modelo de identificación por 3 puntos

El algoritmo para la comunicación se puede observar en los anexos **A y B.**

3.5.2 Algoritmos de locomoción

Para la locomoción se dispuso de dos algoritmos de movimiento principales, seguidos de un algoritmo de control 'semi-difuso'.

- **Locomoción en línea recta:** Este algoritmo fue dispuesto por [45] y adaptado para ajustarse al lenguaje de programación a utilizar (MATLAB) y para disminuir algunos tiempos de ejecución. (**Ver Anexo C**)

- **Locomoción giro cerrado:** Este algoritmo fue recreado a partir de los diagramas de flujo presentados en [45] y algunas características útiles del **algoritmo para locomoción recta,** lo cual, aunque **no es parte del alcance de esta tesis,** tuvo que ser desarrollado para poder continuar. (**Ver Anexo D**)

- **Control Semi-difuso:** Dícese 'semi-difuso' debido a que basa su funcionalidad en reglas condicionales que relacionan ciertas entradas con un respectivo estado de salida. A diferencia de un control difuso convencional, este no hace uso de funciones de defusificación para hallar un valor de salida. El funcionamiento (entre cada punto de la trayectoria final) se ve planteado en el siguiente diagrama:

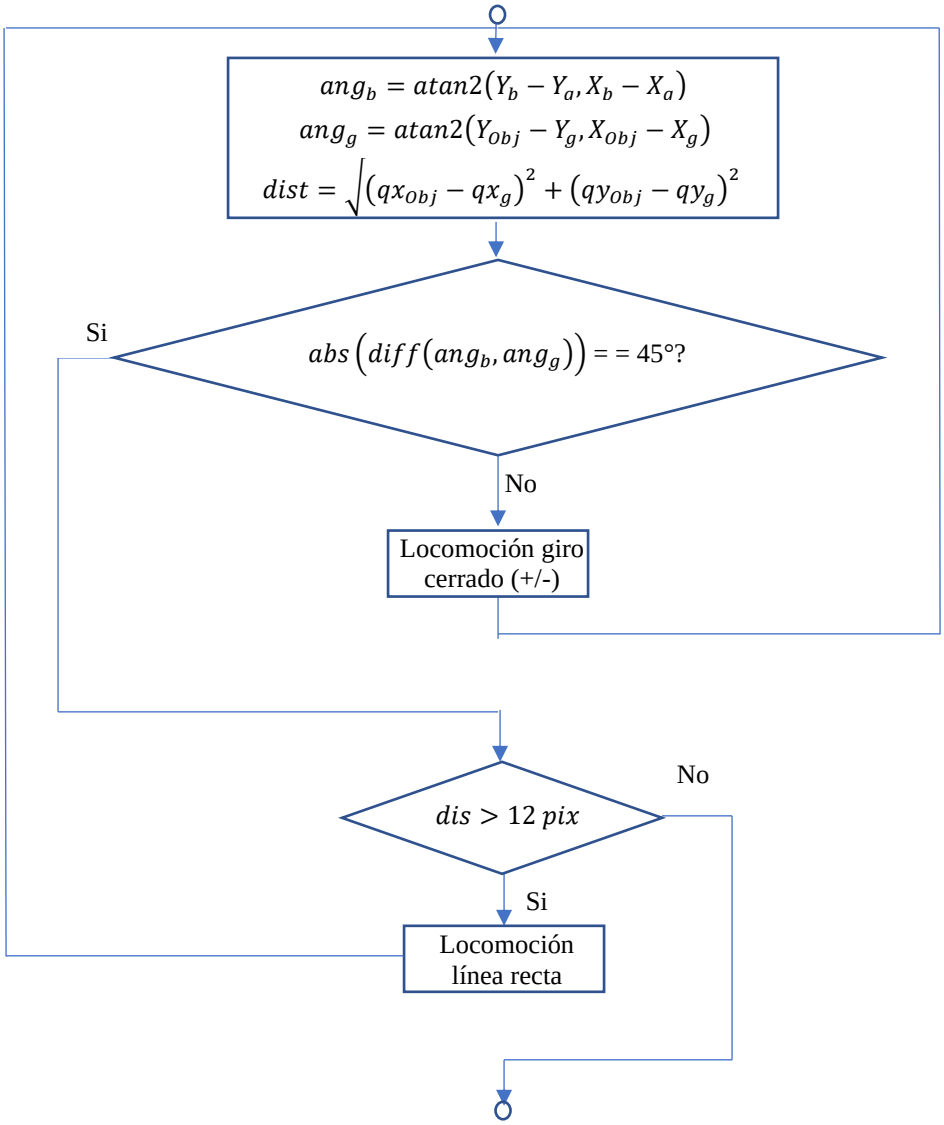


Figura 3. 13 Diagrama de flujo control semi-difuso.

En donde:

- ang_g es el ángulo entre el elemento central del sistema y el punto objetivo.
- ang_b es el ángulo entre el elemento central del sistema y el elemento esférico azul.
- *dis* es la distancia entre el elemento central del sistema y el punto objetivo

Cabe denotar que el sentido de giro en la locomoción de giro cerrado se da a partir del cálculo de la distancia de giro mínima. La condición $abs\left(diff(ang_b, ang_g)\right) = = 45°$ proviene del hecho de que el elemento esférico azul está ubicado (intencionalmente) a 45° con respecto al elemento esférico central (verde), con un radio de 0.25 m (radio nominal del sistema) convertido a píxeles, de aquí es que

se deriva la segunda condición, $dis > 12$. En el **Anexo E** se puede observar la forma en la que se programó lo previamente mencionado.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

En este capítulo se presentarán las distintas pruebas realizadas con sus resultados correspondientes. Las pruebas se dividen en dos categorías: Pruebas para el algoritmo de generación de trayectorias, y pruebas de ejecución. Para realizar estas pruebas se dispondrán de distintos espacios de trabajo, en donde la configuración de los obstáculos varía según la forma, cantidad y ubicación:

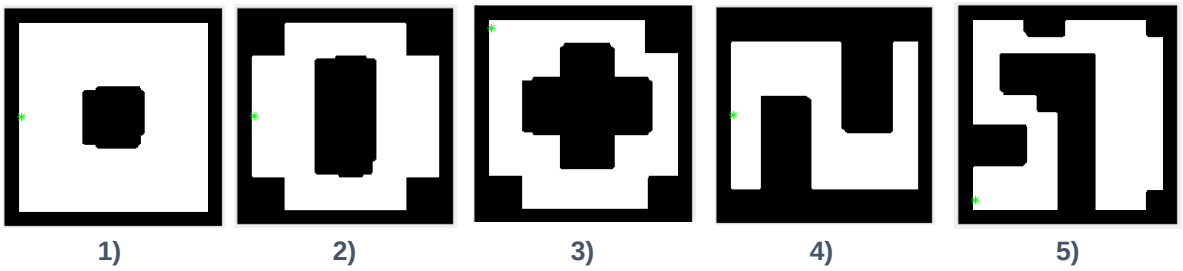


Figura 4. 1 Espacios de configuraciones para obstáculos cuadrados

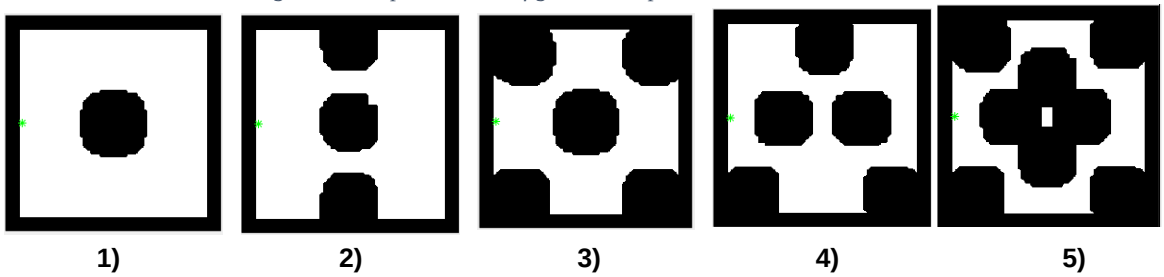


Figura 4. 2 Espacios de configuraciones para obstáculos circulares

4.1 Pruebas para el algoritmo de generación de trayectorias

Para estas pruebas se evaluará el tiempo de cómputo al igual que la cantidad de correcciones necesarias para obtener una trayectoria óptima.

4.1.1 Obstáculos cuadrados:

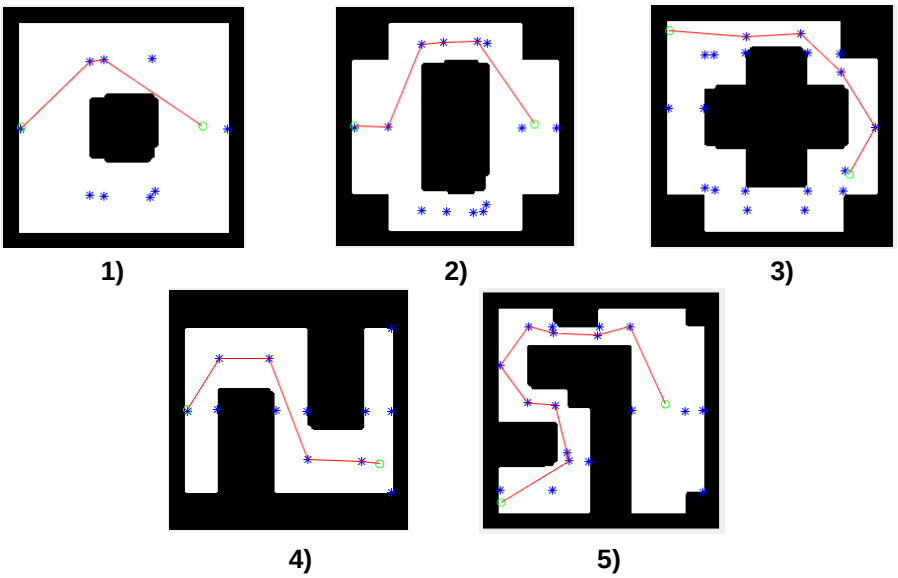


Figura 4. 3 Trayectorias generadas

		Correcciones		
Espacio de configuraciones	**Tiempo de cómputo (s)**	**Sí**	**No**	**Número de correcciones**
1	3.6886		✓	N/A
2	2.1132	✓		1
3	2.6639	✓		3
4	1.8453	✓		1
5	2.4937	✓		2

Tabla 4. 1 Resultados obtenidos

4.1.2 *Obstáculos circulares*:

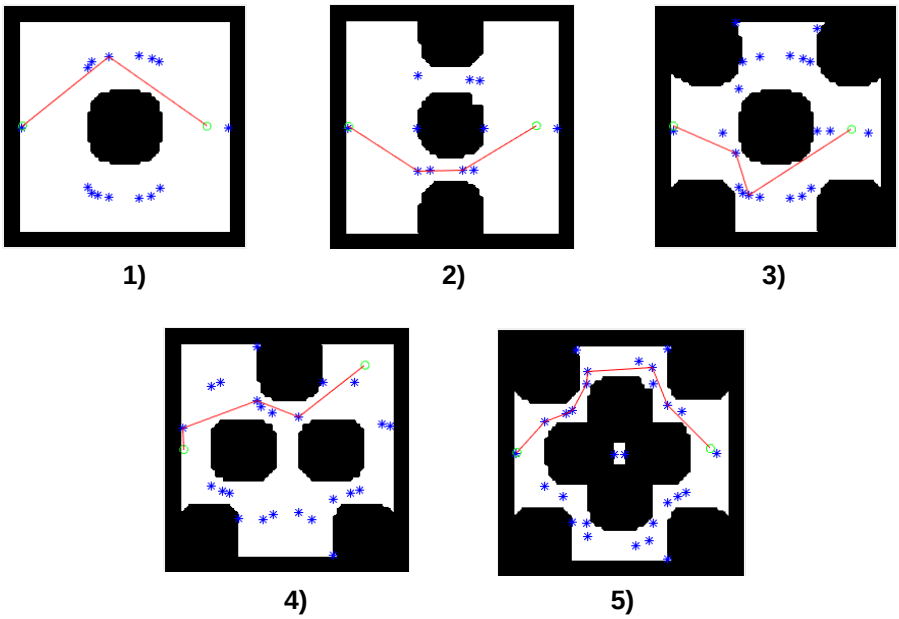


Figura 4. 4 Trayectorias generadas.

		Correcciones		
Espacio de configuraciones	**Tiempo de cómputo (s)**	**Sí**	**No**	**Número de correcciones**
1	1.9921	✓		3
2	2.0477	✓		2
3	2.4359	✓		3
4	2.5739	✓		1
5	3.4676		✓	N/A

Tabla 4. 2 Resultados obtenidos

4.2 Pruebas para los algoritmos de locomoción

Para realizar estas pruebas, se dispuso de las trayectorias generadas en las figuras 4.3 y 4.4 sobre su espacio de trabajo correspondientes, evaluando el tiempo que tarda el sistema en recorrer la trayectoria completa. De esta forma se obtuvieron los siguientes resultados.

Espacio de Configuraciones	Obstáculos cuadrados min:seg:ms	Obstáculos circulares min:seg:ms
1	01 :03:296	00:52:416
2	01:03:616	01:03:872
3	01:20:576	00:58:240
4	01:07:840	01:17:056
5	01:25:325	01:21:600

Tabla 4. 3 Resultados obtenidos

4.3 Interfaz de usuario en MATLAB

Con el propósito de que en trabajos futuros se logre la implementación de estos algoritmos, se dispuso de una interfaz programada en MATLAB (Ver Figura 4.5 a Figura 4.12).

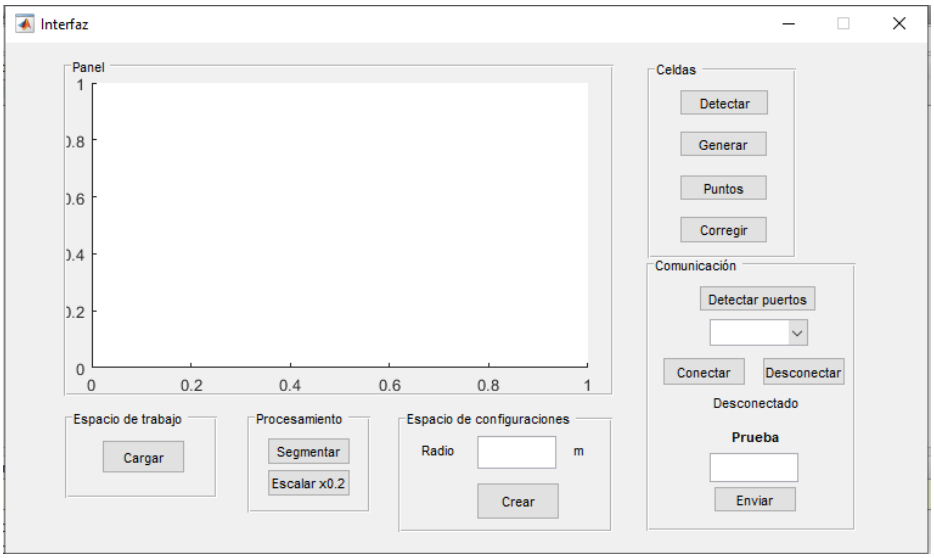


Figura 4. 5 Interfaz en MATLAB

Por el momento la interfaz realiza el procesamiento de imágenes a partir de imágenes guardadas en un directorio, por lo que de ser utilizada, se ha de guardar la imagen del espacio de trabajo en dicho directorio.

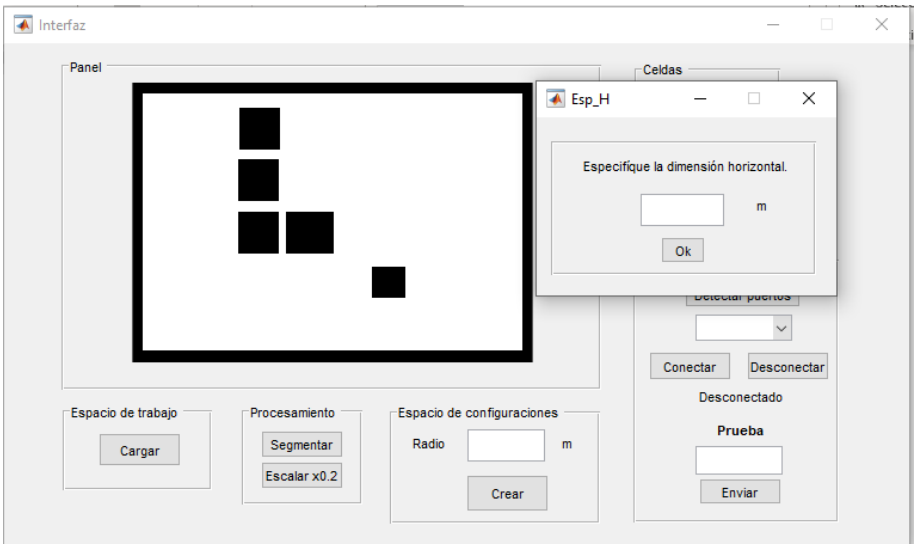


Figura 4. 6 Carga de imágenes

Para realizar pruebas, se generó un espacio con una herramienta de gráficos (Paint), una vez cargada se genera un cuadro donde se solicita ingresar la dimensión horizontal del espacio para realizar los cálculos subsecuentes. Claro está, se busca que en un futuro, la toma de imágenes se pueda realizar en tiempo real o por medio de acceso directo a cámaras.
También se dispone de una sección de procesamiento que busca mejorar la imagen a utilizar para la generación del espacio de configuraciones.

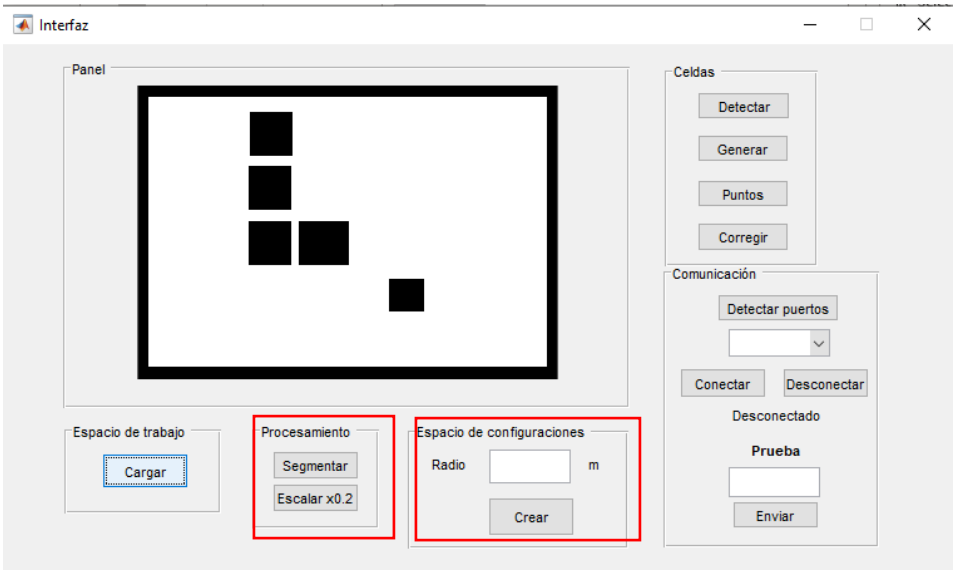


Figura 4. 7 Secciones de procesamiento y generación del espacio de configuraciones

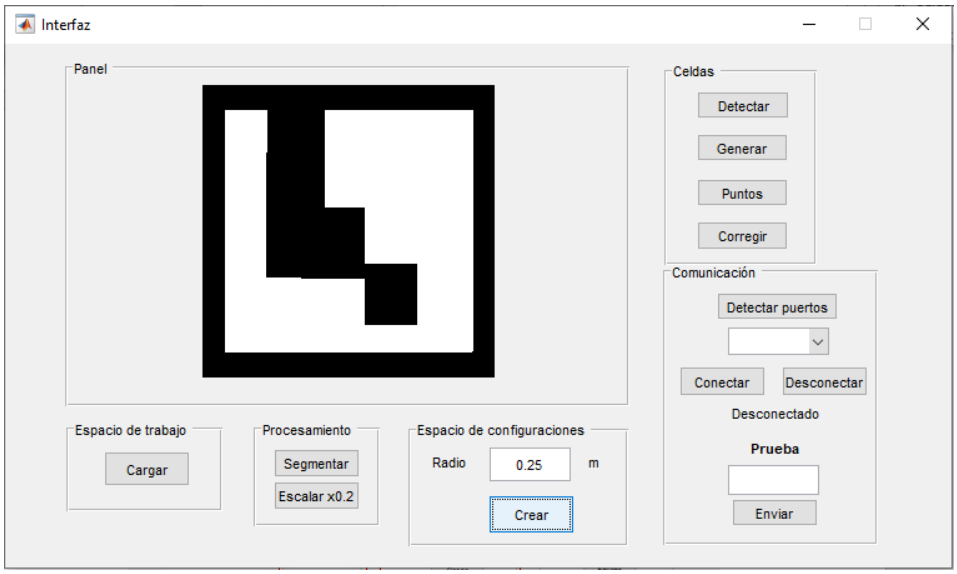


Figura 4. 8 Resultados del espacio de configuraciones

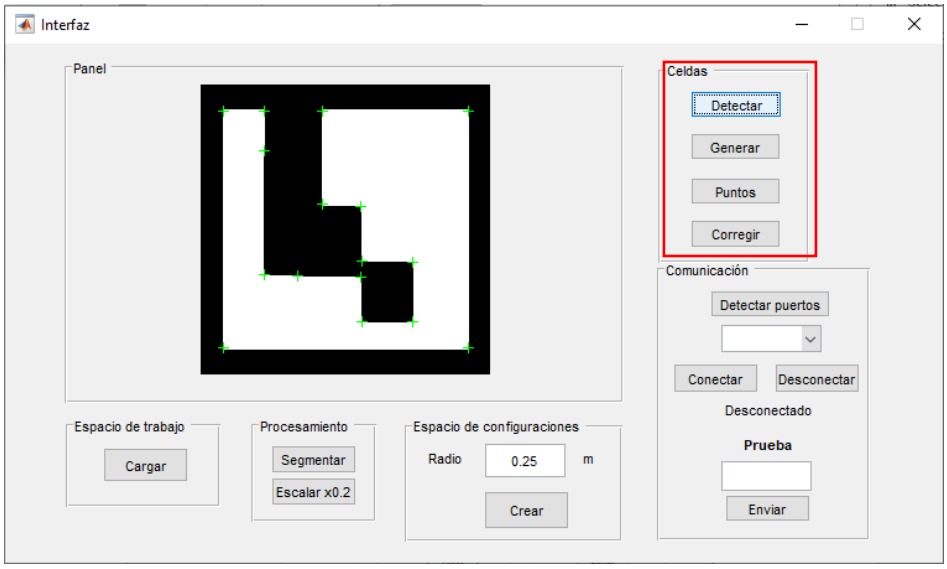


Figura 4. 9 Identificación de esquinas

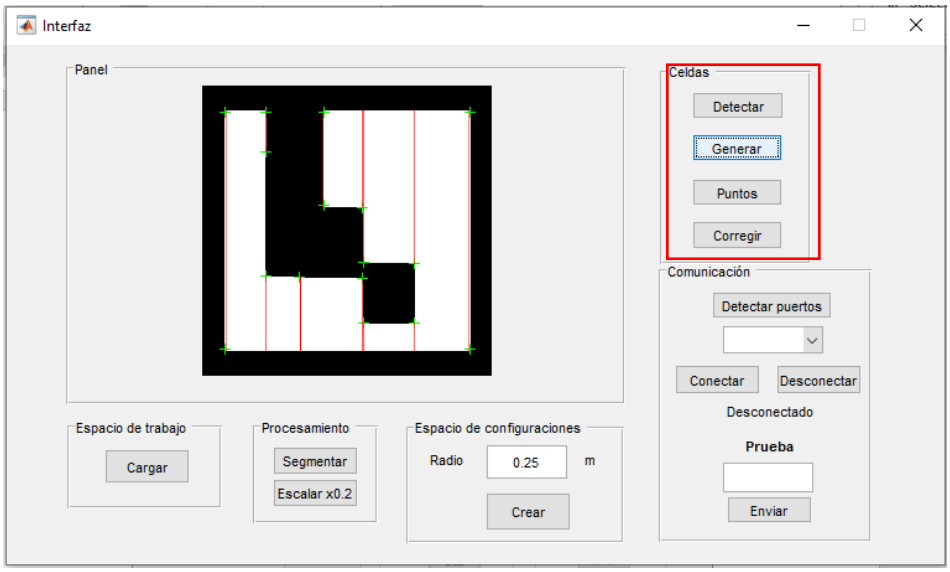


Figura 4. 10 Generación de celdas

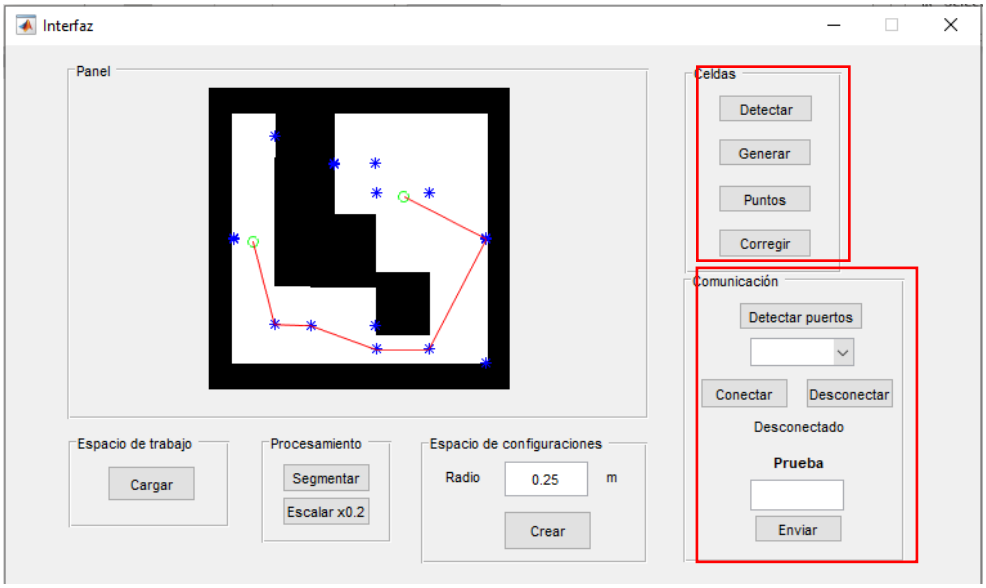


Figura 4. 11 Generación de trayectoria

Una vez obtenida la trayectoria, se dispone de una sección de comunicación que permite detectar el puerto en el cual se encuentra conectado el módulo XBee, conectarse a los módulos del sistema modular, y realizar pruebas de movimiento para verificar el funcionamiento de los módulos independientes, o en conjunto, según el comando enviado [45]. Para estas pruebas se utilizó un módulo XBee configurado igual a aquellos presentes en el sistema real, como se expresa en [45] y conectado mediante XCTU (aplicación para interactuar con los módulos) en modo *Router*. La interfaz de MATLAB funciona como coordinador (Envío de datos).

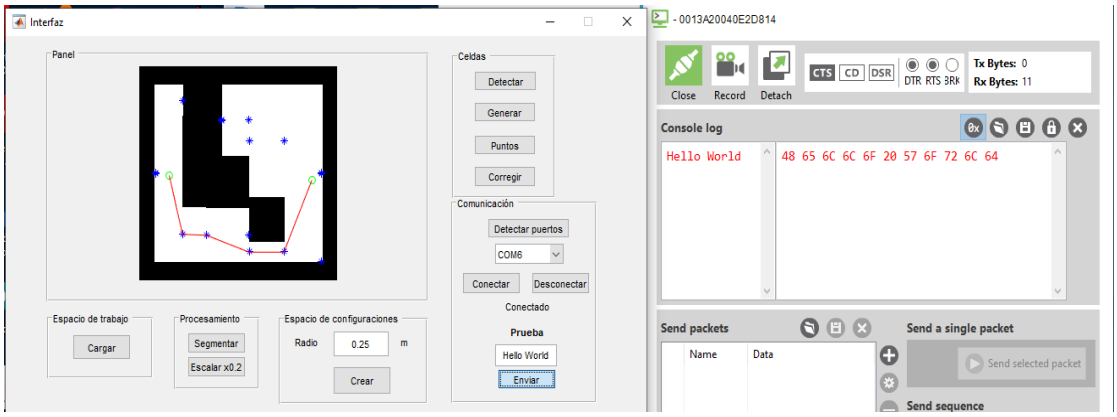


Figura 4. 12 Prueba conexión XCTU

4.4 Análisis de resultados

- La imagen obtenida por la cámara presenta una calidad muy baja, por lo que es necesario realizar operaciones de transformación para obtener mejores resultados. Esto afecta el movimiento del sistema al momento de ejecutar los algoritmos de locomoción puesto que las coordenadas de los puntos de la trayectoria están referenciadas respecto a la imagen transformada, por lo que al momento de referenciarlas respecto a la imagen original se pierde información, y en una imagen de tan baja calidad, un píxel puede representar grandes distancias.

- El número de correcciones no aparenta depender de la complejidad del espacio de configuraciones, sino de la posición del punto objetivo respecto a los distintos puntos de acceso a las celdas, ya que de esto depende el comportamiento del algoritmo de búsqueda.

- El tiempo de cómputo de la trayectoria parece depender de la complejidad del espacio de configuraciones, no porque existan varios obstáculos, sino por la cantidad de conexiones que existan entre un punto y otro, ya que el algoritmo evalúa cada punto con respecto a los demás, escoge aquellos que sean válidos y a partir de eso genera los valores correspondientes para el algoritmo de búsqueda. También, se debe tener en cuenta el hecho de que el software Webots® requiere de ciertas operaciones adicionales para el acceso a las librerías de MATLAB.

- La calidad de la identificación de esquinas depende de la imagen tomada, ya que como es observa en la figura 4.1, imagen 1, aunque el obstáculo sea un contenedor cuadrangular, se generan deformaciones en la imagen debido a las características de la cámara que obtiene las imágenes. Esto genera más esquinas para detectar, con esto más celdas para procesar, lo que explica el aumento del tiempo de procesamiento utilizando la interfaz Webots (ver tabla 3.4)

CAPÍTULO 5
CONCLUSIONES Y RECOMENDACIONES PARA TRABAJOS FUTUROS

A continuación se presentan las conclusiones obtenidas y algunas recomendaciones para aquellos trabajos que utilizaran el desarrollo aquí expuesto como referencia para algún trabajo futuro.

5.1 Conclusiones

- Se determinó que el algoritmo de descomposición en celdas exactas es el más adecuado a partir del resultado obtenido en la evaluación de los parámetros mencionados anteriormente. Para el algoritmo de búsqueda se analizan algunos de los distintos métodos que existen en la literatura. Se implementa el algoritmo A* debido a su flexibilidad y amplio rango de aplicaciones.

- Se desarrolló un algoritmo de planeación de trayectorias por descomposición del espacio de configuraciones en celdas exactas, donde posteriormente se ejecuta un algoritmo de búsqueda que permite hallar una trayectoria al punto objetivo. Se determinó que algunos de los métodos de procesamiento de imágenes inicialmente implementados generaban un resultado poco apropiado y en un tiempo mayor a los obtenidos posteriormente, debido a esto se implementaron algoritmos de optimización (opcionales) con lo que se logró disminuir el tiempo de procesamiento de la imagen y el tiempo de cómputo empleado para hallar la trayectoria mencionada.

- Se desarrolló un controlador “semi-difuso” para lograr simular el movimiento del sistema, puesto que este no dispone de sensores, y los movimientos rectilíneos tienden a desviarse de la trayectoria inicial a lo largo del tiempo, debido a la naturaleza del movimiento. Algunas condiciones del controlador presentan inconvenientes puesto que generan inconsistencias debido a la forma en la que el controlador halla los ángulos entre los distintos elementos del sistema. Debido a esto, se tuvo que implementar un rango de desviación que permita cierta flexibilidad en el controlador.

- Se realizaron simulaciones del sistema en el entorno Webots®, monitoreando el tiempo de cómputo y el tiempo de ejecución. Se determinó que el tiempo de cómputo varía según la complejidad del espacio de trabajo, de la disponibilidad de memoria del computador y de las operaciones de acceso a librerías de MATLAB utilizadas por el entorno.

- El tiempo de ejecución depende de los parámetros de velocidad y amplitud en los movimientos rectilíneos y rotacionales. También depende de cuanta flexibilidad se le haya permitido al controlador. Un control poco flexible no presenta buenos resultados puesto que la naturaleza del movimiento a realizar

exige cierto grado de flexibilidad. Por el otro lado, un control muy flexible resulta en desviaciones bastante considerables, lo que genera posibles choques del robot y los obstáculos dentro del espacio de trabajo.

- El uso continuo del entorno, en algunas ocasiones presentaba problemas de compatibilidad con las librerías internas de MATLAB, lo que resulta en problemas de licencia, por lo que era necesario desinstalar y reinstalar MATLAB. Debido a esto, se deben analizar alternativas para migrar el proyecto a plataformas como ROS o Gazebo.

5.2 Recomendaciones

- El algoritmo de búsqueda funciona según la evaluación de la validez de todos los puntos respecto al punto actual, por lo que esto incrementa el tiempo de cómputo, de ser posible, optimizar el algoritmo de forma que solo se evalúen los puntos de acceso contiguos al actual.

- Como se mencionó anteriormente, se recomienda buscar una migración a una plataforma distinta, esto debido a que si se va a procesar grandes cantidades de información o a simular sistemas complejos, el funcionamiento de Webots® con MATLAB resulta poco, cercano a nada óptimo, debido a que el acceso a librerías ralentiza el proceso y si se requiere realizar cambios, se debe reiniciar la plataforma y volver a cargar las librerías, resultando en una gran cantidad de tiempo perdido.

- Antes de realizar propuestas, asegurarse que se dispone de completa y transparente comunicación con aquellos individuos cuyos trabajos están siendo utilizados como base para el desarrollo del trabajo a realizar. Refiérase a estudiantes que han trabajado con el sistema y disponen de los controladores necesarios.

- Los algoritmos de planificación de trayectorias están sujetos a cambios que se consideren pertinentes, por lo que si se determina que hay un mejor algoritmo para el desarrollo del trabajo no existen restricciones a parte del tiempo requerido para programar y simular. Lo mismo sucede con los algoritmos de búsqueda.

REFERENCIAS

[1] CAMBRIDGE DICTIONARY: Necessity is the mother of invention. [en línea]. Cambridge. [Consultado: 8 julio de 2019]. Disponible en https://dictionary.cambridge.org/dictionary/english/necessity-is-the-mother-of-invention

[2] OXFORD ENGLISH DICTIONARY. [sitio web]. Oxford. [Consultado: 3 de julio de 2019]. Disponible en http://www.oed.com/

[3] 2GM. Goliath (vehículo teledirigido). [en línea]. España. [Consultado: 11 de julio de 2019]. Disponible en: https://www.lasegundaguerra.com/viewtopic.php?t=11634

[4] GONZÁLEZ, Anexa. Robots para exploradores. [en línea]. Prezi: Argentina. (24 de abril de 2015). [Consultado 11 de julio de 2019]. Disponible en https://prezi.com/ddzng8_pcl3s/robots-para-exploradores/

[5] CERÓN CORREA, Alexander. Sistemas robóticos teleoperados. En: *Ciencia e Ingeniería Neogranadina*. [en línea]. Bogotá D.C.: Universidad Militar Nueva Granada, noviembre de 2005. Nro. 15. p. 62-72. [Consultado: 13 de julio de 2019]. Disponible en: http://www.umng.edu.co/documents/63968/74791/r15_05.pdf

[6] AZKUNE, Gorka. Navegación autónoma. [en línea]. Cuentos cuánticos. España. (12 de noviembre de 2011). [Consultado: 14 de julio de 2019]. Disponible en: https://cuentos-cuanticos.com/2011/11/12/navegacion-autonoma/

[7] ZAMORA, Erik. Robots Autónomos: Navegación. [en línea]. Notas sobre Robótica Autónoma y Aprendizaje Automático. México. (8 de noviembre de 2015). [Consultado: 14 de julio de 2019]. Disponible en: https://ezamorag.wordpress.com/2015/11/08/robots-autonomos-navegacion/

[8] DEPARTAMENTO NACIONAL DE PLANEACIÓN: 3.181 muertos y 12,3 millones de afectados: las cifras de desastres naturales entre 2006 y 2014. [en línea]. DNP: Colombia. (25 de mayo de 2015). [Consultado: 14 de julio de 2019]. Disponible en: https://www.dnp.gov.co/Paginas/3-181-muertos,-21-594-emergencias-y-12,3-millones-de-afectados-las-cifras-de-los-desastres-naturales-entre-2006-y-2014-.aspx

[9] CENTRO NACIONAL DE MEMORIA HISTÓRICA: Estadísticas del conflicto armado en Colombia. [en línea]. Bogotá D.C. (2012). [Consultado: 15 de julio de 2019]. Disponible en: http://www.centrodememoriahistorica.gov.co/micrositios/informeGeneral/estadisticas.html

[10] WILLIAMS, Bob. An Introduction to Robotics. [en línea]. Ohio: Dr. Bob Productions. 2019. [Consultado: 17 de julio de 2019]. Disponible en: https://www.ohio.edu/mechanical-faculty/williams/html/PDF/IntroRob.pdf

[11] KAPILA, Vikram. Introduction to Robotics. [en línea]. Engineering NYU. New York. [Consultado: 17 de julio de 2019]. Disponible en: http://engineering.nyu.edu/mechatronics/smart/pdf/Intro2Robotics.pdf
[12] GOLDBERG, Steve. An Introduction to Mobile Robotics. [en línea]. [Consultado: 17 de julio de 2019]. Disponible en: https://www.uio.no/studier/emner/matnat/ifi/INF3480/v14/undervisningsmatriale/intro-to-mobile-robots.pdf
[13] KULICH, Miroslav. Introduction to Mobile Robotics. [en línea]. Czech Technical University. Praga. [Consultado: 20 de julio de 2019]. Disponible en: https://cw.fel.cvut.cz/old/_media/courses/ae3m33mkr/intro-i.pdf
[14] NEHMZOW, Ulrich. Mobile Robotics: Research, Applications and Challenges. [en línea]. Department of Computer Science. The University of Manchester. Manchester M13 9PL. United Kingdom. 2001. [Consultado: 21 de julio de 2019]. Disponible en: https://pdfs.semanticscholar.org/820f/06d1a4f4dff304f71464a0166792e147d344.pdf
[15] UNIVERSITY OF LEEDS. Exploration Robotics. [en línea]. Yorkshire. [Consultado: 21 de julio de 2019]. Disponible en: https://robotics.leeds.ac.uk/research/field-robotics/exploration/
[16] AHMADZADEH, Hossein; MASEHIAN, Ellips y ASADPOUR, Masoud. Modular Robotic Systems: Characteristics and Applications. [en línea]. Springer Science+Business Media Dordrecht. Teherán. (18 de junio de 2015). [Consultado: 22 de julio de 2019]. Disponible en: https://link.springer.com/article/10.1007/s10846-015-0237-8
[17] TEINCO. Robótica Modular. [en línea]. Studylms. Bogotá D.C. (16 de marzo de 2018). [Consultado: 22 de julio de 2019]. Disponible en: https://teinco.edu.co/index.php/2018/03/16/robotica-modular/
[18] VERGARA PULGAR, Andrea Alejandra. Diseño y fabricación de robots modulares blandos. [en línea]. Memoria para optar al título de ingeniero civil mecánico. Santiago de Chile. Universidad de Chile. Facultad de ciencias físicas y matemáticas. Departamento de ingeniería mecánica, 2015. 89 p. [Consultado: 22 de julio de 2019]. Disponible en: Repositorio Universidad de Chile. http://repositorio.uchile.cl/bitstream/handle/2250/132941/Diseno-y-fabricacion-de-robots-modulares-blandos.pdf?sequence=1&isAllowed=y
[19] IEA ROBOTICS. A general classification of the modular robots. [en línea]. (9 de octubre de 2006). [Consultado: 24 de julio de 2019]. Disponible en: http://www.iearobotics.com/personal/juan/publicaciones/art14/html/node2.html

[20] HARVARD UNIVERSITY. Kilobots. [en línea]. Self-Organizing Systems Research Group. Cambridge. [Consultado: 24 de julio de 2019]. Disponible en: https://ssr.seas.harvard.edu/kilobots

[21] ROCKEL, Sebastian. An Introduction To Modular Robots. [en línea]. Universität Hamburg. [Consultado: 29 de julio de 2019]. Disponible en: https://tams.informatik.uni-hamburg.de/lehre/2009ws/seminar/ir/PDF/ModRobotsPres.pdf

[22] MENDOZA, Ricardo. Robot Odin. [en línea]. Reserach Gate. [Consultado: 29 de julio de 2019]. Disponible en: https://www.researchgate.net/figure/Odin-robot-with-21-modules-in-a-closely-packed-lattice-cubic-closed-packed_fig6_224339766

[23] GONZALEZ, Juan. Ejemplo de autorreconfiguración probado con Polybot G2. [en línea]. Research Gate. [Consultado: 29 de julio de 2019]. Disponible en: https://www.researchgate.net/figure/Figura-216-Ejemplo-de-autorreconfiguracion-probado-con-Polybot-G2-1-El-robot-avanza_fig1_259195011

[24] MICHELSON, Robert. Autonomous Navigation. [en línea]. AccessScience. McGraw-Hill Education. (2000). [Consultado: 29 de julio de 2019]. Disponible en: https://www.accessscience.com/content/YB000130

[25] UNIVERSIDAD NUEVA GRANADA. Grupo de investigación DAVINCI. [sitio web]. Bogotá D.C. [Consultado: 30 de julio de 2019]. Disponible en: https://www.umng.edu.co/programas-academicos/facultad-ingenieria/pregrados/ingenieria-mecatronica/davinci

[26] RUBIANO MONTAÑA, Oscar Gerardo y HURTADO ERASSO**,** Camilo Andrés. DISEÑO Y SIMULACIÓN DE UN ROBOT MODULAR RECONFIGURABLE. [en línea]. Trabajo de Grado para Optar al Título de Ingeniero en Mecatrónica. Bogotá D.C. Universidad Nueva Granada. Facultad de Ingeniería. Programa de Ingeniería en Mecatrónica, 2013. 205 p. [Consultado: 4 de agosto de 2019]. Disponible en: https://repository.unimilitar.edu.co/handle/10654/10150

[27] LANCHEROS GUZMÁN, Paola Natalia y SANABRIA GALVIS, Laura Beatriz. Simulación e Implementación de Movimientos para Sistema Robótico Modular Considerando Diferentes Configuraciones. [en línea]. Trabajo de Grado para Optar al Título de Ingeniero en Mecatrónica. Bogotá D.C. Universidad Nueva Granada. Facultad de Ingeniería. Programa de Ingeniería en Mecatrónica, 2016. 157 p. [Consultado: 2 de agosto de 2019]. Disponible en: https://repository.unimilitar.edu.co/handle/10654/15382

[28] COTERA BERDUGO, Mateo. Simulación e Implementación de una Configuración de Robot Hexápodo Utilizando Sistemas de Robótica Modular para Evaluar su Locomoción. [en línea]. Trabajo de Grado para Optar por el Título de

Ingeniero en Mecatrónica. Bogotá D.C. Universidad Nueva Granada. Facultad de Ingeniería. Programa de Ingeniería en Mecatrónica, 2017. 135 p. [Consultado: 4 de agosto de 2019]. Disponible en: https://repository.unimilitar.edu.co/bitstream/handle/10654/16120/CoteraBerdugoMateo2017.pdf?sequence=3

[29] DAVENPORT, James. A Piano Movers Problem. [en línea]. ACM Sigsam Bulletin. [Consultado: 2 de agosto de 2019]. Disponible en: https://www.researchgate.net/publication/255678007_A_''piano_movers'_'_problem

[30] CHOSET, Howie; LYNCH, Kevin; HUTCHINSON, Seth; KANTOR, George; BURGARD, Wolfram. KAVRAKI, Lydia y THRUN, Sebastian. Principles of Robot Motion. The MIT Press. Cambridge & Londres. 2005, 615 p. ISN 0-262-03327-5.

[31] LATOMBE, Jean-Claude. Robot Motion Planning. Springer Science+Business Media New York. 1991. 667 p. ISBN 978-1-4615-4022-9.

[32] LAVALLE, Steven. Planning Algorithms. Cambridge University Press. 2006, 1008p. ISBN 978-0521862059.

[33] STOYAN, Y.G y YAKOLEV, S.V. Configuration Space of Geometric Objects. [en línea]. (24 de septiembre de 2018). [Consultado: 10 de agosto de 2019]. Disponible en: https://link.springer.com/article/10.1007/s10559-018-0073-5

[34] CHOSET, Howie. Robotic Motion Planning: Roadmap Methods. [en línea]. Robotics Institute. [Consultado: 12 de agosto de 2019]. Disponible en: https://www.cs.cmu.edu/~motionplanning/lecture/Chap5-RoadMap-Methods_howie.pdf

[35] ARRAS, Kai; BENNEWITZ, Mares; BURGARD, Wolfram y STACHNISS, Cyrill. Robot Motion Planning. [en línea]. Universidad de Friburgo [Consultado: 12 de agosto de 2019]. Disponible en: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf

[36] CHOSET, Howie. Robotic Motion Planning: Cell Decompositions (with some discussion on coverage and pursuer/evader). [en línea]. Robotics Institute 16-735. [Consultado: 12 de agosto de 2019]. Disponible en: https://www.cs.cmu.edu/~motionplanning/lecture/Chap6-CellDecomp_howie.pdf

[37] BOSCARIOL, Paolo; GASPARETTO, Alessandro; LANZUTTI, Albano y VIDONI, Renato. Path Planning and Trajectory Planning Algorithms: A General Overview. Mechanisms and Machine Science. [en línea]. 2015. [Consultado: 12 de agosto de 2019]. Disponible en: https://www.researchgate.net/publication/282955967_Path_Planning_and_Trajectory_Planning_Algorithms_A_General_Overview

[38] GETIAL, Jesús y PANTOJA, Andrés. A Quantitative Comparison of Path Planning Methods in Mobile Robotics. [en línea]. IEEE 3rd Colombian Conference on Automatic Control (CCAC), At Cartagena – Colombia. 2017. [Consultado: 15 de agosto de 2019]. Disponible en: https://www.researchgate.net/publication/322675531_A_Quantitative_Comparison_of_Path_Planning_Methods_in_Mobile_Robotics

[39] MASEHIAN, Ellips Masehian, y SEDIGHIZADEH, Davoud. Classic and Heuristic Approaches in Robot Motion Planning – A Chronological Review. En: *Open Science Index, Mechanical and Mechatronics Engineering*. [en línea]. Teherán: Universidad Tarbiat Modares, 2007, Vol 1, Nro 5. 6 p. [Consultado: 16 de agosto de 2019]. Disponible en: https://waset.org/publications/10300/classic-and-heuristic-approaches-in-robot-motion-planning-a-chronological-review-

[40] PATEL, Amit. Introduction to A*. [en línea]. Red Blob Games. 2019. [Consultado: 16 de agosto de 2019]. Disponible en: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[41] ABIY, Thaddeus; PANG, Hannah y TILIKSEW, Beakal. A* Search. [en línea]. Brilliant.org. [Consultado: 18 de agosto de 2019]. Disponible en: https://brilliant.org/wiki/a-star-search/

[42] MATHWORKS. strel. [en línea]. The MathWorks, Inc. [Consultado: 18 de agosto de 2019]. Disponible en: https://www.mathworks.com/help/images/ref/strel.html

[43] MATHWORKS. im2bw. [en línea]. The MathWorks, Inc. [Consultado: 18 de agosto de 2019]. Disponible en: https://www.mathworks.com/help/images/ref/im2bw.html

[44] SINHA, Utkarsh. Harris Corner Detector. [en línea]. AI Shack. 2016. [Consultado: 18 de agosto de 2019]. Disponible en: http://aishack.in/tutorials/harris-corner-detector/

[45] CRUZ CARBONELL, Vanessa. [en línea]. Simulación e implementación de arquitectura cuadrúpeda utilizando sistema robótico modular MECABOT. Trabajo de grado para optar por el título de ingeniero en mecatrónica. Bogotá D.C.: Universidad Nueva Granada. Facultad de Ingeniería. Programa de Ingeniería en Mecatrónica, 2018. 148 p. [Consultado: 20 de agosto de 2019]. Disponible en: https://repository.unimilitar.edu.co/bitstream/handle/10654/18007/CruzCarbonellVanessa2018.pdf?sequence=1&isAllowed=y

[46] B. Doroodgar, M. Ficocelli, B. Mobedi and G. Nejat, "The search for survivors: Cooperative human-robot interaction in search and rescue environments using semi-autonomous robots," 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, 2010, pp. 2858-2863.

[47] Polvara, R., Sharma, S., Wan, J., Manning, A., & Sutton, R. (2018). Obstacle Avoidance Approaches for Autonomous Navigation of Unmanned Surface Vehicles. Journal of Navigation, 71(1), 241-256. doi:10.1017/S0373463317000753

[48] Rui Song, Yuanchang Liu, Richard Bucknall, Smoothed A* algorithm for practical unmanned surface vehicle path planning, Applied Ocean Research, Volume 83, 2019, Pages 9-20, ISSN 0141-1187, https://doi.org/10.1016/j.apor.2018.12.001.

[49] R. Gonzalez, M. Kloetzer and C. Mahulea, "Comparative study of trajectories resulted from cell decomposition path planning approaches," 2017 21st International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, 2017, pp. 49-54.

[50] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, Robin De Keyser, Heuristic approaches in robot path planning: A survey, Robotics and Autonomous Systems, Volume 86, 2016, Pages 13-28, ISSN 0921-8890, https://doi.org/10.1016/j.robot.2016.08.001.

ANEXOS

Anexo A: Comunicación (Emisor)

```
emitter = wb_robot_get_device('emitter');
channel = wb_emitter_get_channel(emitter);
if (channel ~= COMMUNICATION_CHANNEL)
 wb_emitter_set_channel(emitter, COMMUNICATION_CHANNEL);
end

  Objects=wb_camera_recognition_get_objects(camera);
 if (size(Objects)==[1 3])
  Obj1=Objects(1);
  Obj2=Objects(2);
  Obj3=Objects(3);
  Obj1_ant=Obj1;
  Obj2_ant=Obj2;
  Obj3_ant=Obj3;
 else
  Obj1=Obj1_ant;
  Obj2=Obj2_ant;
  Obj3=Obj3_ant;
 end

  setdatatype(Obj1.colors,'doublePtr',1,3);
  setdatatype(Obj2.colors,'doublePtr',1,3);
  setdatatype(Obj3.colors,'doublePtr',1,3);

  Color1=Obj1.colors.Value;
  Color2=Obj2.colors.Value;
  Color3=Obj3.colors.Value;

  if(Color1==[0.223 1 0.07])
   Mecabot=Obj1;
  elseif(Color2==[0.223 1 0.07])
   Mecabot=Obj2;
  else
   Mecabot=Obj3;
  end
```

```
if(Color1==[0 0 1])
 Ref_Point=Obj1;
elseif(Color2==[0 0 1])
 Ref_Point=Obj2;
else
 Ref_Point=Obj3;
end
qc=double(Mecabot.position_on_image);
Pqx=qc(1)/qx; Pqy=qc(2)/qy;
qc=[round(w*Pqx) round(hi*Pqy)];

qb=double(Ref_Point.position_on_image);
Pqx_b=qb(1)/qx; Pqy_b=qb(2)/qy;
qb=[round(w*Pqx_b) round(hi*Pqy_b)];

success=wb_emitter_send(emitter,[qc qb]);
```

Anexo B: Comunicación (Receptor)

```
receiver = wb_robot_get_device('receiver');
channel = wb_receiver_get_channel(receiver);
if (channel ~= COMMUNICATION_CHANNEL)
 wb_receiver_set_channel(receiver, COMMUNICATION_CHANNEL);
end
wb_receiver_enable(receiver, TIME_STEP);

if(wb_receiver_get_queue_length(receiver) > 0)
  TrayX=round(wb_receiver_get_data(receiver,'double'),2);
  wb_receiver_next_packet(receiver);
  TrayY=round(wb_receiver_get_data(receiver,'double'),2);
  wb_receiver_next_packet(receiver)
  Tray=[TrayX TrayY];
  QX = TrayX(PuntoTray); QY=TrayY(PuntoTray);
  Visual=1;
end
```

Anexo C: Locomoción línea recta

```
phase = phase + TIME_STEP / 1000.0 * FREQUENCY * pi; %Calculo fase onda principal
H_phase = H_phase + TIME_STEP / 1000.0 * FREQUENCY *2* pi; %Calculo fase primer armonico
%Angulos de columna
  angu_motorC1= (-pi/90*cos(phase-(pi/8)));
  angu_motorC2= (-2*pi/45*cos(phase-(pi/8)));
  angu_motorC3= (-pi/12*cos(phase-(pi/8)));
  %Asignacion de valores
  wb_motor_set_position(column{1},angu_motorC1);
  wb_motor_set_position(column{2},angu_motorC2);
  wb_motor_set_position(column{3},angu_motorC3);

  %Angulos de motores de hombros (piernas)
  angu_motor1= (amplitud * sin(phase))+((amplitud/2)* sin(H_phase));
  angu_motor2= (amplitud * sin(phase+bias2))+((amplitud/2)* sin(H_phase+(bias2*2)));
  angu_motor3= (amplitud * sin(phase+bias2-bias1))+((amplitud/2)* sin(H_phase+(bias2*2)-(bias1*2)));
  angu_motor4= (amplitud * sin(phase+bias1))+((amplitud/2)* sin(H_phase+(bias1*2)));

  %Asignacion de valores segun avance o retroceso
  if(angu_motor1>0)
   wb_motor_set_position(legs{1},angu_motor1*WALK_AMPL1);
  else
   wb_motor_set_position(legs{1},angu_motor1*WALK_AMPL12);
  end

  if(angu_motor2>0)
   wb_motor_set_position(legs{2},angu_motor2*WALK_AMPL2);
  else
   wb_motor_set_position(legs{2},angu_motor2*WALK_AMPL22);
  end
if(angu_motor3>0)
 wb_motor_set_position(legs{3},angu_motor3*WALK_AMPL3);
else
 wb_motor_set_position(legs{3},angu_motor3*WALK_AMPL32);
end

if(angu_motor4>0)
 wb_motor_set_position(legs{4},angu_motor4*WALK_AMPL4);
else
 wb_motor_set_position(legs{4},angu_motor4*WALK_AMPL42);
end

%Angulos de rodillas (piernas)
resta1=angu_motor1-resta1;
resta2=angu_motor2-resta2;
resta3=angu_motor3-resta3;
resta4=angu_motor4-resta4;
%Asignacion de valores segun sea estancia o levantamiento
if(resta1>0 && resta4<0)
 wb_motor_set_position(legs{5},pi/2);
else
 wb_motor_set_position(legs{5},pi/4);
end

if(resta2>0 && resta1<0)
 wb_motor_set_position(legs{6},pi/2);
else
 wb_motor_set_position(legs{6},pi/4);
end
```

```
    if(resta3>0 && resta2<0)
     wb_motor_set_position(legs{7},pi/2);
    else
     wb_motor_set_position(legs{7},pi/4);
    end

    if(resta4>0 && resta3<0)
     wb_motor_set_position(legs{8},pi/2);
    else
     wb_motor_set_position(legs{8},pi/4);
    end

    resta1=angu_motor1; %Reasignacion de indicadores
    resta2=angu_motor2;
    resta3=angu_motor3;
    resta4=angu_motor4;
 end
```

Anexo D: Locomoción giro cerrado

```
phase = phase + TIME_STEP / 1000.0 * FREQUENCY * pi; %Calculo f
H_phase = H_phase + TIME_STEP / 1000.0 * FREQUENCY *2* pi; %Cal
%Angulos de motores de hombros (piernas)
for i=1:4
 angu_motor(i) = amplitudG*sin(phase + bias(i));
 resta(i)=angu_motor(i)-resta(i);
end
%Asignación de moteres de hombros
for i=1:4
 if (angu_motor(i)>0)
  wb_motor_set_position(legs{i},angu_motor(i)*Walk_Ampl(i,1));
 else
  wb_motor_set_position(legs{i},angu_motor(i)*Walk_Ampl(i,2));
 end
end

if (Giro<0)
 for i=1:2
   if (resta(i)>0)
     if(i==1)
      wb_motor_set_position(legs{5},pi/2);
      wb_motor_set_position(legs{8},pi/2);
     else
      wb_motor_set_position(legs{6},pi/2);
      wb_motor_set_position(legs{7},pi/2);
     end
   else
```

```
    if(i==1)
      wb_motor_set_position(legs{5},pi/4);
      wb_motor_set_position(legs{8},pi/4);
    else
      wb_motor_set_position(legs{6},pi/4);
      wb_motor_set_position(legs{7},pi/4);
    end
   end
 end
else
 for i=1:2
   if (resta(i)>0)
     if(i==1)
      wb_motor_set_position(legs{5},pi/4);
      wb_motor_set_position(legs{8},pi/4);
     else
      wb_motor_set_position(legs{6},pi/4);
      wb_motor_set_position(legs{7},pi/4);
     end
   else
    if(i==1)
      wb_motor_set_position(legs{5},pi/2);
      wb_motor_set_position(legs{8},pi/2);
    else
      wb_motor_set_position(legs{6},pi/2);
      wb_motor_set_position(legs{7},pi/2);
    end
   end
 end
end
resta=angu_motor;
```

Anexo E: Control Semi-difuso

```
if(b-d>=0 && g-tol>0)

 if(b-d>g+tol)
  Girando=1;Avanza=0;Giro=-1;
 elseif(b-d<g-tol)
  Girando=1;Avanza=0;Giro=1;
 else
  Girando=0;
 end

elseif(b-d<0 && g+tol<0)

 if(b-d>g+tol)
  Girando=1;Avanza=0;Giro=-1;
 elseif(b-d<g-tol)
  Girando=1;Avanza=0;Giro=1;
 else
  Girando=0;
 end

elseif(b-d<0 && g-tol>=0)
 if(abs(b-d)-(g+tol)<=pi)
  Girando=1;Avanza=0;Giro=-1;
 elseif(abs(b-d)-(g+tol)>=pi)
  Girando=1;Avanza=0;Giro=1;
 else
  Girando=0;
 end

elseif(b-d>=0 && g+tol<0)

 if(abs(g-tol)+(b-d)<pi)
  Girando=1;Avanza=0;Giro=-1;
 elseif(abs(g-tol)+(b-d)>=pi)
  Girando=1;Avanza=0;Giro=1;
 else
  Girando=0;
 end
end
```

```
if(~Girando)
 if(dist>15)
  Avanza=1;
  Girando=0;
 else
  Avanza=0;
  PuntoTray=PuntoTray+1;
  if(PuntoTray>length(TrayX))
   done=1;
   wb_console_print(toc, WB_STDOUT);
  else
   QX = TrayX(PuntoTray); QY=TrayY(PuntoTray);
  end
 end
end
```