UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación y Validación del Algoritmo de Robótica de Enjambre *Ant Colony Optimization* en Sistemas Físicos

Trabajo de graduación presentado por Walter Andres Sierra Azurdia
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

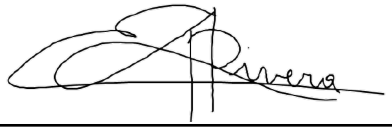
Implementación y Validación del Algoritmo de Robótica de Enjambre *Ant Colony Optimization* en Sistemas Físicos

Trabajo de graduación presentado por Walter Andres Sierra Azurdia
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

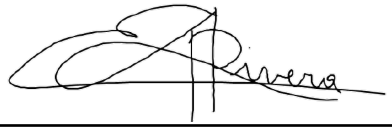
2022

Vo.Bo.:

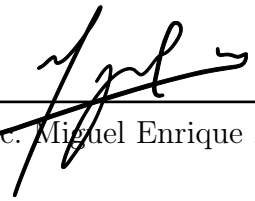
(f) 

Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) 

Dr. Luis Alberto Rivera Estrada

(f) 

MSc. Miguel Enrique Zea Arenales

(f) 

MSc. Pedro Iván Castillo Rivera

Fecha de aprobación: Guatemala, 7 de enero de 2022.

Prefacio

La elaboración de la presente tesis surge de cierto interés en el área de inteligencia computacional, ya que considero la inteligencia artificial, un área muy interesante y con muchos ámbitos de aplicación en la vida cotidiana, haciendo que el esfuerzo del humano sea cada vez menor. Para este trabajo se requirió del conocimientos en distintas áreas de investigación, tales como robótica, sistemas de control, programación, inteligencia computacional y de enjambre. Por esto, este trabajo no hubiera sido posible sin los conocimientos impartidos por mis catedráticos a lo largo de mi trayectoria estudiantil en la Universidad del Valle de Guatemala, los cuales considero personas super preparadas profesionalmente.

Quisiera agradecer al comité de ayuda financiera, a mis padres por apoyarme a cumplir mis metas y por darme la oportunidad de poder estudiar en la Universidad del Valle de Guatemala, y agradezco a todos mis compañeros de la carrera por todos los años que compartimos cumpliendo nuestro sueño de ser ingenieros. Además quiero agradecer especialmente a mi asesor de tesis Dr. Luis Rivera por todo el apoyo que me ha brindado y por compartir su conocimiento y resolver todas las dudas que semana a semana le planteaba. A todas las personas que conocí en la universidad que se convirtieron en parte de mi familia, ya que sin ustedes la universidad no hubiera sido lo mismo.

La situación actual de la pandemia COVID-19 en Guatemala es muy delicada, por esta razón quiero hacer una mención especial y agradecer de corazón a todos los médicos que a diario luchan por la población.

Índice

Lista de figuras

Lista de cuadros

Resumen

El área de inteligencia de enjambre *swarm* busca emular el comportamiento exhibido por la naturaleza en distintas especies de animales que actúan en conjunto, como colonias de hormigas, parvadas de aves o cardumen de peces. La robótica es una de las muchas áreas académicas que han tomado como inspiración este comportamiento.

En la Universidad del Valle de Guatemala se desarrolla el megaproyecto *Robotat* en el cual se utiliza el algoritmo *Modified Particle Swarm Optimization* (MPSO) para el movimiento de robots diferenciales. Como una alternativa de planificación de trayectoria, fue implementado el algoritmo *Ant System* (AS) con el objetivo de comparar ambos algoritmos y tener diversos algoritmos para ser implementados en los robots de UVG.

En el presente trabajo, se toman los avances de la fase anterior que se encuentran a nivel de simulación y se migran a la plataforma Raspberry Pi y lenguaje de programación de C++ para la implementación de este algoritmo en un sistema físico. Se comparan las trayectorias obtenidas por el AS con las trayectorias obtenidas en las simulaciones.

Al no contar con una plataforma móvil funcional, se realizaron diversas pruebas simples en ambientes controlados y de esta forma poder validar el funcionamiento del algoritmo. Se utilizó una plataforma de rastreo con visión por computadora, para lograr obtener la pose del robot. También se utilizó esta plataforma para poder enviar la información del mapa a utilizar para que pueda calcular la ruta con el mapa enviado.

Para lograr la comunicación entre la plataforma de visión por computadora y el robot, se implementó la programación multihilos y transmisión de datos mediante un protocolo UPD. Esta implementación permite la ejecución de varias tareas de forma simultanea. Asimismo fueron implementados los controladores que en trabajos anteriores presentaron la respuesta más suave de velocidad y estos son: el controlador de pose y el controlador de pose de Lyapunov.

Se validó el funcionamiento del algoritmo mediante varias pruebas utilizando la plataforma de rastreo con visión por computadora. Para esta validación se probaron dos escenarios, el primero se utilizó un mapa sin obstáculos y se probaron varias rutas y para el segundo escenario se colocaron uno y varios obstáculos por todo el mapa. Finalmente, se implementó

un sistema dinámico del AS donde el algoritmo es capaz de recalcular la ruta cuando se detecte un nuevo obstáculo en el mapa.

Abstract

Swarm intelligence seeks to emulate the behavior exhibited by nature in different species of animals that act together, such as ants colonies, flocks of birds or shoal of fish. Robotics is one of the many academic areas that have taken this behavior as inspiration.

At the Universidad del Valle de Guatemala the megaproject *Robotat* was developed in which the algorithm *Particle Swarm Optimization* (PSO) is used for the movement of differential robots. As an alternative to trajectory planning, the *Ant System* (AS) algorithm was implemented with the objective to compare both algorithms and to have different algorithms to be implemented in UVG robots.

In the following work, the advances of the previous phase that are at the simulation level are taken and migrated to the Raspberry Pi platform and C ++ programming language for the implementation of this algorithm in a physical system. The trajectories obtained by the AS are compared with the trajectories obtained in the simulations.

As it did not have a functional mobile platform, several simple tests were carried out in controlled environments and in this way be able to validate the operation of the algorithm. A computer vision tracking platform was used to obtain the robot's pose. This platform was also used to be able to send the information of the map that will be used to calculate the route.

To achieve communication between the computer vision platform and the robot, multi-wire programming and data transmission were implemented using a UPD protocol. This implementation allows the execution of several tasks simultaneously. Likewise, the controllers that in previous works presented the smoothest speed response were implemented and these are: the pose controller and the Lyapunov pose controller.

The performance of the algorithm was validated through various tests using the computer vision tracking platform. For this validation, two scenarios were tested, the first was used a map without obstacles and several routes were tested and for the second scenario, one and several obstacles were placed throughout the map. Finally, a dynamic AS system was implemented in which the algorithm is able to recalculate the route when a new obstacle is detected on the map.

CAPÍTULO 1

Introducción

En este trabajo se presenta una propuesta de una implementación del algoritmo *Ant Colony Optimization* para aplicaciones de robótica de enjambre. Principalmente se busca implementar y validar el algoritmo ya realizado en fases anteriores a nivel de simulación a una plataforma y lenguaje adecuados para la implementación en un sistema físico y comparar el desempeño de este con el de la simulación.

Como metodología primero se presenta la validación de la selección más adecuada de la plataforma, lenguaje de programación y entrono de desarrollo mediante un análisis de *Trade Study* donde se evaluaron las opciones de microcontroladores y lenguajes de programación para la implementación en un sistema físico.

También se presenta la migración del algoritmo *Ant System* en la plataforma Raspberry Pi en lenguaje de programación C++. Para validar la migración del algoritmo se replica una de las pruebas realizadas en [1], donde se replica el mapa y todos los parámetros del algoritmo. Se realizó un ajuste en los parámetros logrando un funcionamiento óptimo. Por último se graficarion las rutas obtenidas donde se observa un camino lógico con movimientos suaves. Para un enfoque a robots móviles y aprovechando la plataforma en Webots realizada en [1], se validaron las rutas calculadas con simulaciones utilizando un robot E-puck.

Se implementó la comunicación multihilos para el proceso de recepción de datos de la mesa de pruebas a través de la plataforma de visión por computadora, dicha plataforma está siendo desarrollada en otro trabajo de graduación en paralelo a esta tesis. Se reciben dos tipos de datos por lo que se utilizaron dos tipos de marcadores en la mesa de pruebas, uno para reconocer la pose del robot y otro que representa el mapa (grafo) que se utilizará. Para que esto fuera posible se utiliza un *tag*, representando el robot en la mesa de pruebas y con esta información poder implementar un controlador de posición y velocidad, también se detectar una serie de etiquetas (puntos) en la mesa de pruebas, estas representan nodos de un grafo, de esta forma poder crear un mapa físico y poder aplicar el algoritmo al mapa creado.

Para esta comunicación se usó un protocolo UDP de cliente-servidor conectado a una red local. Esta comunicación fue validada por separado y luego se validó junto con el algoritmo AS. En el cual se recibió la información del mapa y fue posible correr el algoritmo y calcular la ruta con la información recibida para todos los escenarios probados.

CAPÍTULO 2

Antecedentes

2.1. *Programmable Robot Swarms*

Los ingenieros del Instituto *Wyss* de Harvard se inspiran en el comportamiento colectivo que presentan ciertos animales para lograr metas increíbles a través de la distribución de acciones de millones de agentes independientes. Este comportamiento es imitado con el objetivo de construir robots móviles simples que aprovechan y demuestran el potencial de la robótica *swarm* en ejecutar trabajos colectivos. Dentro de las muchas aplicaciones de la robótica de enjambre se puede mencionar: transportar objetos grandes, construir estructuras a grandes escalas, tareas de búsqueda y rescate, aplicaciones médicas, entre otros.

El instituto desarrolló un sistema robótico de bajo costo y fácil de usar llamado Kilobot para el desarrollo avanazdo de robótica *swarm*, estos cuentan con sensores y actuadores para adaptarse a ambientes dinámicos [2].

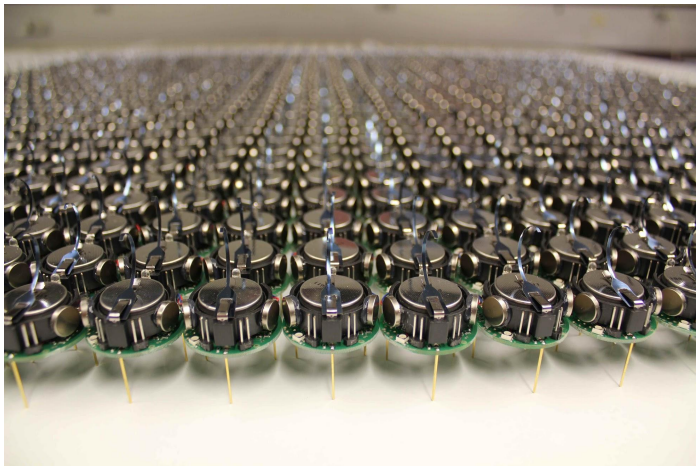


Figura 1: Kilobot del instituto Wyss [3].

2.2. Robotarium de Georgia Tech

El proyecto Robotarium provee acceso remoto a una plataforma de robótica de enjambre de forma gratuita y accesible para cualquier persona en cualquier parte del mundo con el fin que puedan hacer pruebas de sus algoritmos en sus robots reales e ir mas allá de una simulación en sus investigaciones. Para poder experimentar con los robots del Robotarium, se tiene que descargar un simulador ya sea en Matlab o en Python y ejecutar el prototipo en el simulador, subir el código que se ejecutará en el robot real a través de la interfaz web, registrarse en la página y esperar la aprobación para utilizar la plataforma [4].

Figura 2: Robotarium [5].

2.3. Implementación de *Bug Algorithm* en un robot E-Puck

Anteriormente se han implementado algoritmos de seguimiento de trayectoria con obstáculos Por ejemplo, en [6] se describe la implementación en un robot E-Puck de un algoritmo para que este avanzara en una trayectoria mientras evadía obstáculos en el camino, este trabajo fue desarrollado por Departamento de Ingeniería en sistemas y Control del Instituto Universitario de Automática e Informática Industrial, en la Universidad Politécnica de Valencia.

Se modeló y simuló la cinemática del robot para la implementación del algoritmo, también se realizaron pruebas con el robot físico. Tanto en la simulación utilizando WeBots como en el robot real se obtuvo un buen desempeño, a pesar de los buenos resultados analizaron los efectos al ser expuestos a distintos factores ambientales y como estos afectaban el desempeño de los robots. Uno de los factores que más afecto en el desempeño del robot fue el

color de los obstáculos, ya que el robot detectaba mejor los obstáculos que eran de color rojo. También se pudo observar que el módulo de odometría del robot era menos preciso cuando la distancia recorrida aumentaba o cuando el robot realizaba una cantidad considerable de rotaciones. Finalmente se pudo demostrar no solo la implementación del *Bug Algorithm* sino también la implementación de un sistema de control para mantener la trayectoria de una manera muy precisa [6].

2.4. Implementación de PSO en robots diferenciales

En la segunda fase del proyecto Robotat de la Universidad del Valle de Guatemala desarrollado por Aldo Aguilar [7], se utilizó como base el algoritmo clásico *Particle Swarm Optimization*. Este algoritmo se modificó con el fin de poder implementarlo en robots diferenciales reales ya que sin esta modificación, los movimientos con el algoritmo estándar realizado las partículas (robots) son irregulares y como consecuencia se tendrá una saturación en los actuadores del robot.

El algoritmo modificado emplea un planeador el cual antes de seguir la posición exacta dada por el algoritmo, utiliza un sistema de control el cual retraerá la trayectoria determinada y generara velocidades suaves y continuas para los robots diferenciales. Se realizaron distintas pruebas en entorno de simulación de *WebBots* en las cuales se determinó que el controlador con mejor desempeño es el *TUC-LQI*.

2.5. Aprendizaje profundo en aplicaciones de robótica de enjambre

La tercera fase el proyecto Robotat desarrollado por Eduardo Santizo [8] se proponen dos soluciones para realizar mejoras al algoritmo PSO mediante el uso de técnicas de aprendizaje reforzado y profundo. La primera propuesta es una mejora al algoritmo por medio de redes neuronales recurrentes y la segunda propuesta fue una alternativa al algoritmo de navegación alrededor de un ambiente conocido por medio de programación dinámica.

Para poder mejorar el desempeño del PSO se empleo el *PSO Tuner* que consiste de una red neuronal recurrente que toma diferentes métricas propias de las partículas del PSO y a través de su procesamiento por medio de una red LSTM, GRU o BiLSTM las torna en predicciones de los parámetros que deberá utilizar el algoritmo. Luego de obtener los parámetros adecuados, el PSO Tuner fue capaz de reducir el tiempo de convergencia en mínimos locales, la red utilizada fue la BiLSTM ya que esta resulto ser la mejor opción de las tres propuestas.

Para la alternativa al algoritmo se utilizo como base la programación dinámica de *Grid-world*, sin embargo este fue modificado para lograr ajustar problemas con robots diferenciales. En primer lugar, se incremento el numero de acciones que puede tomar el robot, es decir que ahora el robot se podrá mover en una diagonal a 45^o. Luego se divide el espacio de trabajo en celdas y es escaneado para determinar las celdas obstáculo y meta. Por ultimo, mediante *Policy Iteration* se genera una acción óptima por estado, para generar una trayec-

toria a seguir por el controlador, tanto el PSO Tuner como el planificador de trayectorias fueron probados a nivel de simulación utilizando Matlab.

2.6. Implementación del ACO en robots diferenciales

En el trabajo desarrollado por Gabriela Iriarte [1], en el cual se implementó el algoritmo de inteligencia de enjambre *Ant Colony Optimization* como planificador de trayectorias y poder tener otra alternativa al MPSO. Para poder comparar el desempeño de este algoritmo se utilizaron varios controladores y se determinó cuál de todos resulta ser más efectivo. Luego de realizar las pruebas, el controlador con la respuesta más suave son el controlador de pose y el controlador de pose de Lyapunov. También se realizaron pruebas con algoritmos genéticos para explorar otras alternativas al MPSO y el ACO. El planificador de trayectorias fue realizado en Matlab y para realizar pruebas se realizó una simulación en Webots utilizando como robot el E-Puck.

El alcance tanto de este trabajo como el realizado por Eduardo Santizo [8], fue realizar la implementación de los algoritmos y realizar pruebas con dicho algoritmo todo a nivel de simulación. Para obtener los resultados se utilizó Matlab y Webots como herramientas para hacer las pruebas. A pesar que las pruebas realizadas fueron exitosas y se demostró un buen desempeño de los algoritmos, no se llegó a una implementación física en ninguno de los trabajos.

CAPÍTULO 3

Justificación

Dentro de la rama de inteligencia computacional de enjambre, en conjunto con el algoritmo *Particle Swarm Optimization*, el *Ant Colony Optimization* (ACO) es uno de los algoritmos más utilizados. En la fase anterior a este proyecto, se planteó el uso del algoritmo ACO como un método alternativo al PSO en el proyecto Robotat de la Universidad del Valle de Guatemala. Este es utilizado para planificar la trayectoria utilizada por un robot móvil para lograr llegar a la meta de la forma más rápida. Los resultados de este planificador de trayectorias fueron probados en un entorno de simulación en Matlab y Webots. Finalmente se llegó a implementar el algoritmo exitosamente a nivel se simulación.

En este trabajo el enfoque principal está en migrar e implementar el planificador de trayectorias en robots diferenciales físicos, para de esta forma poder someter al sistema a factores reales. También se busca validar el desempeño del algoritmo en los robots diferenciales móviles físicos al ser puestos a prueba en ambientes controlados utilizando el algoritmo ACO. Con esta migración la Universidad del Valle de Guatemala estará un paso más cerca de la implementación de algoritmos de robótica en enjambre enfocado en investigación y/o enseñanza en el laboratorio de robótica.

La migración del algoritmo ACO, permite tener el planificador de trayectorias y poder ser utilizado en un robot móvil autónomo sin la necesidad de tener una computadora conectada a los robots diferenciales que este realizando los cálculos. Con esta migración se podrá en un futuro proyecto adaptar esta migración a una plataforma móvil, para que de forma autónoma reciba el origen, la meta y la pose del robot y que que de forma independiente el robot realice la tarea solicitada, como por ejemplo una aplicación en búsqueda y rescate, y poder enviar las señales correspondientes a los motores para poder desplazarse con movimientos suaves hasta llegar a la meta.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar y validar los algoritmos de robótica en enjambre *Ant Colony Optimization (ACO)* y algunas variantes desarrollados en años anteriores a nivel de simulación, en sistemas físicos.

4.2. Objetivos específicos

- Evaluar distintas opciones de microcontroladores, sistemas embebidos, lenguajes de programación y entornos de desarrollo, y seleccionar los más adecuados para su uso en aplicaciones de robótica de enjambre utilizando el algoritmo ACO.
- Migrar los algoritmos desarrollados anteriormente a los microcontroladores de los sistemas físicos seleccionados.
- Validar la migración de los algoritmos y verificar el desempeño de los sistemas físicos mediante pruebas simples en ambientes controlados.

CAPÍTULO 5

Alcance

El alcance de este trabajo abarcó la migración y validación del algoritmo *Ant Colony Optimization* en un sistema físico para planificar trayectorias, en específico la migración a una Raspberry Pi. Para esto, se implementó la versión de *Ant System* planteada por Marco Dorigo, para la cual se empleó el lenguaje de programación C++. También se implementaron los controladores de posición y velocidad, esto para considerar las restricciones físicas que tienen las ruedas. Todo fue programado en plataforma de Raspberry Pi, para una futura implementación en un robot Pi-puck o similar que posea una plataforma de Raspberry Pi para programación.

La plataforma de rastreo por visión de computadora que se instalará en la mesa de pruebas la cual nos dará la posición y orientación de los robots se encuentra en desarrollo actualmente. A pesar de esta limitante sí fue posible utilizar esta plataforma y se logró realizar algunas pruebas del algoritmo con la información proveniente de esta plataforma, mediante el uso de unas marcadores que simulan ser un robot móvil.

Dadas las limitaciones que se tienen debido a las restricciones a causa de la pandemia Covid-19 que limita la asistencia de forma presencial a la universidad y la falta de una plataforma móvil que posea motores, sensores y módulos de comunicación, se plantean diversas técnicas para demostrar el funcionamiento del algoritmo. En este trabajo no se realizaron pruebas con una plataforma con motores por lo que no es posible hacer una validación de los controladores implementados. La plataforma móvil mencionada se encuentra en desarrollo actualmente en otro trabajo de graduación paralelo a este y se basa en la Raspberry Pi, por lo que todos los programas desarrollados en este trabajo podrán ser implementados en dicha plataforma en una futura fase a este proyecto.

CAPÍTULO 6

Marco teórico

6.1. Computación evolutiva

La computación evolutiva (EC) tiene como objetivo imitar procesos de evolución, donde el concepto principal es la supervivencia del más apto y el débil debe morir. La supervivencia se logra mediante la reproducción, la descendencia que proviene de dos padres (en algunos casos pueden ser más de dos) contiene material genético de los padres, en el mejor de los casos contiene las mejores características (descendencia apta) y en el peor de los casos contiene las peores características, estos son individuos débiles que morirán en el ambiente competitivo donde viven como lo indica Darwin en su teoría de la evolución [9]. Esto está muy bien ilustrado en algunas especies de aves donde una cría logra obtener más comida, obtiene más fuerte, y al final echa a todos sus hermanos del nido para que mueran.

La computación evolutiva es una herramienta poderosa en la resolución de problemas inspirado en la evolución natural. Modela los elementos esenciales de la evolución biológica mediante el uso de una población de individuos y explora el espacio de la solución mediante la herencia genética, la mutación y la selección de individuos más aptos, donde cada individuo se le denomina cromosoma y sus características se denominan genes [10].

Existen distintas clases de algoritmos evolutivos. Estos métodos evolutivos han demostrado su éxito en varios problemas de optimización difíciles y complejos [11]:

- Algoritmos genéticos
- Programación genética
- Programación evolutiva
- Estrategias evolutivas
- Evolución diferencial

- Evolución cultural
- Coevolución
- Inteligencia de enjambre

6.1.1. Inteligencia de enjambre

La inteligencia de enjambre o *swarm inteligence* (SI) forma parte de la rama de la computación evolutiva (CE) cuyo principal enfoque es la investigación del comportamiento colectivo de sistemas descentralizados, auto-organizados ya sea natural o artificial. En términos generales al grupo se pueden referir como *swarm*. Formalmente, *swarm* se define como un grupo de agentes (generalmente móviles) que se comunican entre si (directa o indirectamente) actuando en un ambiente local. Dicha interacción entre los agentes resultan en una estrategia distributiva colectiva de resolución de problemas. El término *swarm intelligence* se refiere a la estrategia de resolución de problemas que surge de la interacción entre los agentes y el término *computational swarm intelligence* (CSI) se refiere al algorimo que modela dicho comportamiento [12].

Los agentes en un sistema SI siguen reglas muy simples. No existe una estructura de control centralizada que dicte cómo deben comportarse los agentes individuales. Los comportamientos reales de los agentes son locales y, hasta cierto punto, aleatorios, sin embargo, las interacciones entre dichos agentes llevan a un comportamiento global "inteligente", que es desconocido para los agentes individuales. La inspiración proviene de sistemas biológicos en la naturaleza. Algunos ejemplos bien conocidos de SI incluyen colonias de hormigas, bandadas de aves, pastoreo de animales, crecimiento de bacterias y cardúmenes de peces [12], [11].

6.2. *Ant Colony Optimization* (ACO)

6.2.1. Inspiración

La inspiración de este algoritmo viene del comportamiento natural en una colonia de hormigas, al rededor de los cuarentas y cincuentas del siglo 20, el entomólogo Pierre-Paul Graseé observo en una colonia de termitas lo que él denominó como "Estímulos Significativos". En sus observaciones pudo ver los efectos de las reacciones tanto en el insecto que produjo como en la colonia y utilizó el término de "estigmergía" para describir este tipo de comunicación en la cual los trabajadores son estimulados por el desempeño que logran. En estas observaciones se determinó que las hormigas poseen la habilidad de encontrar el camino más corto entre la fuente de alimento y su colonia mediante el uso de feromonas.

A partir de estas observaciones, en el cual se demuestra que las hormigas tienen la habilidad de encontrar el camino más corto entre una fuente de alimento y su hormiguero, en 1992 Marco Dorigo desarrolló un algoritmo con base al comportamiento de las hormigas denominado *Ant System (AS)*, a partir de este algoritmo se desarrollaron muchos híbridos

hasta que en 1999 Dorigo junto a Di Caro and Gambardella basándose en un método metaheurístico, en donde se ven las hormigas como base del algoritmo denominado ACO, para resolver problemas discretos de optimización [13].

6.2.2. Funcionamiento

Cuando una hormiga encuentra alimento, esta deja un rastro de feromonas en el camino a la colonia, el resto de compañeras detectan este rastro y lo siguen ya que estas saben que encontraran alimento al final del camino, mientras más hormigas pasan por el camino el rastro se hace cada vez más fuerte. Este es un método eficiente ya que aunque existan dos caminos diferentes, mientras más corto sea el camino, las hormigas que vienen detrás pueden detectar la feromonoas dejadas por sus compañeras más rápido, aumentando las posibilidades que escogan el camino mas corto. En el caso del algorimto en forma computacional, la feromona artificial tiene la misma función, indicar la popularidad de la solución del problema de optimización a realizar, en otras palabras funciona como una memoria del proceso de búsqueda.

Este comportamiento se ve reflejado en el experimento de "doble puente" realizado por Deneuborg, donde se colocan dos caminos para llegar a una fuente de alimento, en un principio los dos caminos tenían la misma distancia, al empezar el experimento el comportamiento de las hormigas fue aleatorio, sin embargo mientras transcurría el tiempo y mas hormigas pasaban comenzaban a llegar al alimento, al final todos los insectos seleccionaron un solo camino, luego uno de los caminos se hizo más corto que el otro, al igual que en el experimento anterior, el comportamiento de la colonia fue aleatorio, pero mientras pasaba el tiempo, los insectos escogieron el camino más corto, así como se ejemplifica en la Figura 3, a partir de este experimento se desarrolló un modelo dado por la Ecuación 1 [13].

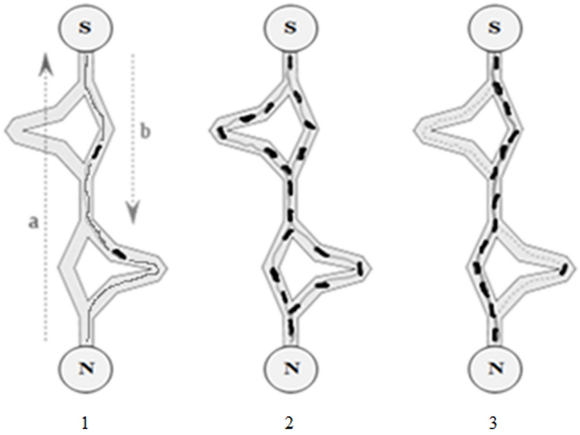


Figura 3: Proceso del algoritmo *Ant Colony Optimization* [14].

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \tag{1}$$

Donde, m_1 y m_2 representan la cantidad de hormigas que han pasado por el camino 1 y

el camino 2 respectivamente, k y h son parámetros que se ajustan mediante experimentación y p_1 representa la probabilidad que la hormiga escoga el camino 1.

6.2.3. Algoritmos desarrollados

Desde el desarrollo de este algoritmo, se han hecho muchas variantes del mismo, algunos de los más conocidos son: Ant System (AS), Ant Colony System (ACS), Max-Min Ant System (MMAS), Ant-Q, Fast Ant System, Antabu, AS-rank y ANTS [11].

Simple Ant Colony Optimization (SACO)

Entre las diversas variantes para el algoritmo ACO, una de las más exitosas y comunes es el *Simple Ant Colony Optimizarion*, cuya principal característica es que en cada iteración los valores de feromonas son actualizados mediante la cantidad de hormigas m que han construido una solución. El modelo que define la cantidad de feromoas τ_{ij} está dado por la Ecuación 2. Generalmente el problema busca el camino más corto entre 2 nodos en un grafo $G(V, E)$ (los grafos se explican detalladamente en el capitulo 6.7).

El algoritmo también considera el largo del camino L^K construido por la hormiga k y este se calcula como el número de saltos en el camino desde el nodo inicial hasta el final [15]. En la Figura 4 se muestra distintas formas de un grafo. En este caso se muestran tres, la primera vemos que tiene dirección, por lo que si se quiere llegar de la arista a a la b, solo se puede llegar por un camino. Con esta restricción la hormiga se deberá desplazar en los nodos disponibles para llegar a la meta. También notamos que cada arista tiene un peso, el cual indica la cantidad de feromona asociada a ese nodo τ_{ij}. Para inicializar el algoritmo, este parámetro toma valores aleatorios y cada hormiga decide a donde dirigirse. Para cada iteración, cada hormiga construye una solución al nodo destino. En cada nodo i, cada hormiga k determina a qué nodo j debe de dirigirse basado en la probabilidad P (Ecuación 4).

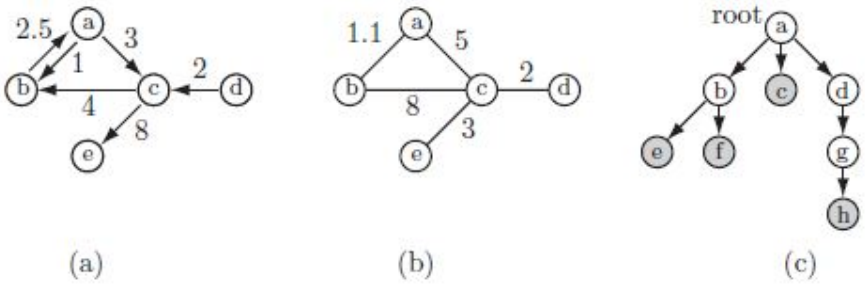


Figura 4: (a) Dígrafo con pesos. (b) Dígrafo con pesos sin dirección. (c) Grafo en forma de árbol [15].

$$\tau_i j = (1 - \rho) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2}$$

Donde, ρ es la taza de evaporación de la feromona, m es el número de hormigas y $\Delta\tau_{ij}^k$ es la cantidad de feromona que hay en las aristas.

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{si } k \in aristas(i,j) \\ 0 & \text{en otro caso} \end{cases} \tag{3}$$

Donde, Q es una constante y L_k es la longitud del recorrido construido por la hormiga k.

$$p_{(i,j)}^k(t) = \begin{cases} \frac{(\tau_y(t))^\alpha}{\sum_{k\in J_k}(\tau_i(t))^\alpha} & \text{si } j \in N_i^k \\ 0 & \text{si } j \notin N_i^k \end{cases} \tag{4}$$

Donde, $N(s^p)$, es el set de componentes factibles, es decir las aristas (i,l) donde l son las aristas que no ha sido visitado por la hormiga k. Los parámetros α y β controlan la importancia de la feromona [13].

Ant System (AS)

Este algoritmo consiste en hacer ciertas mejoras al algoritmo presentado anteriormente. El algoritmo anterior tenía un objetivo más instructivo, por lo que era más simple. Algunas de las mejoras son la incorporación de información heurística en la ecuación de probabilidad [13], esta ecuación queda como:

$$p_{(i,j)}^k(t) = \begin{cases} \frac{(\tau_1(t))^\alpha \cdot (\eta_i(t))^\beta}{\sum_{k\in J_k}(\tau_{ij}(t))^\alpha \cdot (\eta_{ij}(t))^\beta} \end{cases} \tag{5}$$

Este nuevo parámetro, influye en la importancia de la feromona y representa el inverso del costo de la arista dada por la información heurística η_{ij} definida como:

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{6}$$

Donde, d_{ij} es la distancia entre la arista i y j [13].

6.3. Raspberry Pi

La Raspberry Pi (RPi) es un ordenador pequeño y de bajo coste, al cual se puede conectar un monitor y un teclado para interactuar con ella como cualquier otra computadora. Fue desarrollado como una organización caritativa de la Fundación Raspberry Pi en 2009 cuyo objetivo era animar a los niños a aprender informática en las escuelas.

Con Raspberry Pi esto es mucho más sencillo y abre las puertas de la experimentación y el aprendizaje en distinetos ambitos. Puede usarlo para aprender a programar, crear proyectos

de electrónica, y para muchas de las cosas que hace cualquier PC de escritorio, como hojas de cálculo, procesamiento de texto, navegar por Internet y jugar ciertos videojuegos. También reproduce videos de alta definición. La Raspberry Pi está siendo utilizada para aprender de programación y creación digital [16].

La Raspberry Pi es la placa de un ordenador simple compuesto por un SoC, CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, reloj, una toma para la alimentación, conexiones para periféricos de bajo nivel, reloj. Se tiene que conectar conectar periféricos de entrada y salida para poder interactuar con la RPi, instrumentos como un monitor, un ratón y un teclado y grabar un sistema operativo para Raspberry Pi en la tarjeta SD.

6.3.1. Modelos de RPi

Conforme la tecnología avanzaba la compañía Raspberry Pi Trading fue desarrollando distintos modelos. Cada versión nueva fue mejorando el software de sus antecesores haciendo que las nuevas versiones tuvieran mejores características tecnológicas las cuales hacían estos ordenadores más potentes. La primer mejora de esta compañía fue en el 2014, donde se lanza la Raspberry Pi Model B+, que es la versión mejorada de la Raspberry Pi original. Dentro de estos modelos se tienen los siguientes:

- Raspberry Pi 1
- Raspberry Pi 1 Model B+
- Raspbery Pi 2 Model B
- Raspberry Pi 3 Model B+
- Raspberry Pi 3 Model A+
- Raspberry Pi 4 Model B+

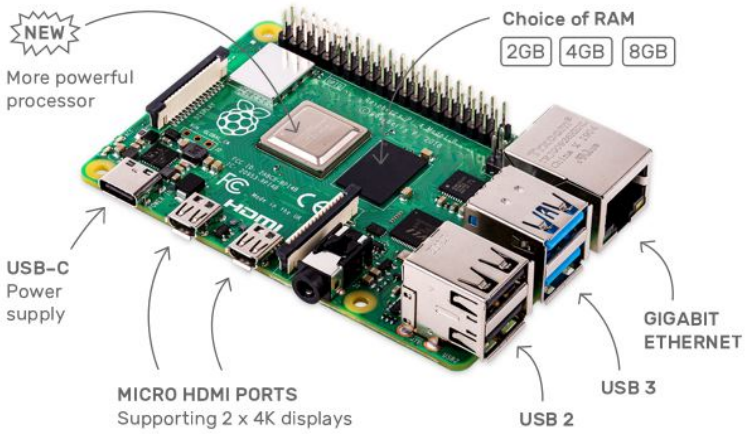


Figura 5: Raspberry Pi [16].

6.4. Robots móviles

Los robots móviles son una clase de robots con la capacidad de moverse en el entorno. Existe una variedad de robots móviles que pueden moverse en el suelo, sobre el agua, a través del aire o bajo del agua. Con estos ejemplos se destaca el diversidad de lo que se conoce como plataforma robótica.

A pesar de los distintos tipos, estos robots móviles son muy similares en términos de lo que hacen y cómo lo hacen. Una de las funciones más importantes de un robot móvil es moverse a de un punto inicial hasta un punto final. La meta podría especificarse en términos de alguna característica del entorno, por ejemplo se mueven hacia la luz, o en términos de alguna coordenada geométrica o referencia de mapa. En cualquier caso, el robot tomará algún camino para llegar a su destino y se enfrentará desafíos como obstáculos que pueden bloquear su camino [17].

Los robots móviles, a diferencia con los seriales que poseen juntas y se obtiene un vector de configuración del robot que está dado por los ángulos de estas juntas y que es mapeado a la posición, representado mediante cinemática directa, los robots móviles poseen ruedas por lo que para saber la distancia recorrida se utiliza:

$$s = r\phi \tag{7}$$

Donde s representa la velocidad recorrida, r es el radio de la llanta y ϕ la velocidad angular.

El modelo más simple para del robot móvil es el modelo uniciclo, que consiste en una rueda y vector de configuración, a partir de este se puede obtener el modelo diferencial que toma en cuenta 2 ruedas, en el cual se describe la velocidad de las llantas en función de la velocidad lineal y angular con las siguientes ecuaciones [18]:

$$v_R = \frac{v + wl}{r} \tag{8}$$

$$v_L = \frac{v - wl}{r} \tag{9}$$

Donde r es el radio de las ruedas y l es el radio del robot, si se asume el modelo de un robot Pi-puck.

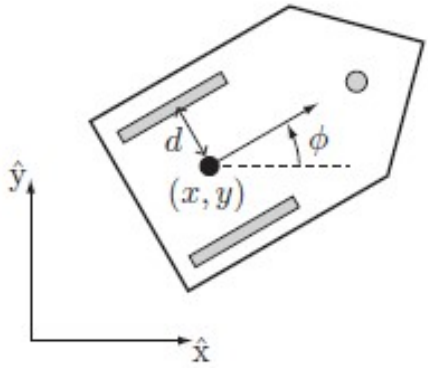


Figura 6: Robot diferencial móvil con dos llantas [18].

6.5. E-Puck

El E-Puck es un robot diferencial móvil con llantas desarrollado por Dr. Fancesco Mondada y Michel Bonani en el 2006 en *EPFL, (Federal Swiss Institute of Technology in Lausanne.* El robot consiste de dos llantas con actuadores que permiten girar en ambas direcciones para cambiar la dirección del robot. Además tiene diversos sensores que permiten hacer mediciones muy buenas de distancia con objetos a su alrededor. El robot utiliza un procesador dsPIC que funciona como un microcontrolador [6].

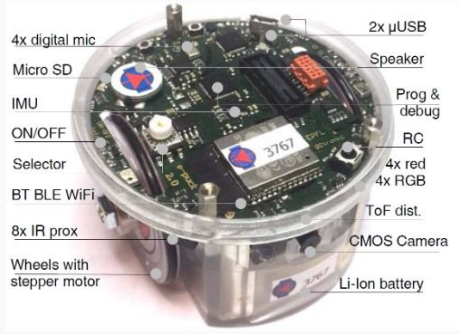


Figura 7: Sensores del robot E-Puck [19].

6.6. Pi-puck

El Pi-puck es la plataforma del robot E-puck con la adaptación de la interfaz de la Raspberry desarrollado en la Universidad de York por (YRL) *(York Robotics Laboratory)* en conjunto con GCtronic. Es una extensión del robot E-puck y el E-puck 2 que permite montar una computadora de placa única Raspberry Pi Zero en el robot, agregando soporte para Linux, periféricos adicionales y mayores posibilidades de expansión [20].

Figura 8: Robot Pi-puck [20].

6.7. Grafos

El algoritmo ACO a diferencia del PSO que utiliza funciones costo para la búsqueda, este algoritmo se basa en grafos. Un grafo G puede definirse como un par de conjuntos (V, E) donde V es el conjunto de vértices y E es el conjunto de ramas [21].

6.7.1. Complejidad

Los algoritmos de búsqueda en grafos tienen una cierta complejidad y es que un criterio importante para evaluar la eficiencia de un algoritmo A es su tiempo de ejecución, es decir, el tiempo necesario para ejecutar el algoritmo en un modelo computacional que captura las características más relevantes de un sistema de elaboración real. En la práctica, uno está interesado en estimar el tiempo de ejecución como una función de un solo parámetro n que caracteriza el tamaño de la entrada dentro de una clase específica de un problema. En la planificación de movimiento, este parámetro puede ser la dimensión del espacio de configuración, o el número de vértices de el espacio de configuración libre (si es un subconjunto poligonal).

El peor tiempo de ejecución $t(n)$ representa el tiempo de ejecución máximo del algoritmo A en función a la cantidad de entradas n. La expresión funcional exacta del tiempo de ejecución $t(n)$ depende de la implementación de el algoritmo, y tiene poco interés práctico porque el tiempo de ejecución en el modelo computacional es solo una aproximación del real. Algo más significativo es el comportamiento asintótico de $t(n)$, es decir, la tasa de crecimiento del tiempo con respecto de las entradas. Si el peor tiempo de ejecución del algoritmo pertenece al grupo de funciones aceptables, la complejidad del tiempo del algoritmo es este tipo de funciones.

Una clase muy importante está representada por algoritmos cuya ejecución en el peor de los casos el tiempo es asintóticamente polinomial en el tamaño de la entrada. En particular, si $t(n)$ pertenece a este grupo de funciones, para algún punto mayor a cero, se dice que el algoritmo tiene tiempo de complejidad polinómica. Si el comportamiento asintótico del peor tiempo de ejecución no es polinomio, la complejidad temporal del algoritmo es exponencial, por lo que se limita este tipo de algoritmos a problemas de tamaño pequeño [22].

6.7.2. Representación

Existen grafos cuya conexión entre vértices que tienen una dirección, la cual se le denomina grafo dirigido o digrafo, esto implica el ir de un nodo a a un nodo b, tendrá una ruta diferente si se quiere ir del nodo b al nodo a. Eswaran y Tarjan resolvieron el problema de una conexión fuerte entre los nodos en el teorema del mínimo-máximo, el cual explica que debido a la dirección de las ramas ciertos nodos tendrán mayor importancia que otros ya que la longitud del camino será mas corta si se escoge dicho nodo [23].

Un grafo puede tiene varias representaciones, una de ellas es la representación mediante una matriz de adyacencia. Esta representación esta vinculada con el desarrollo teórico de

los grafos en la resolución de ecuaciones algebraicas lineales. La matriz cuadrada representa cada uno de los vértices del grafo y mediante el uso de 1 y 0 se identifica cuales son los vértices que están y los que no están conectados respectivamente. Si se considera un grafo $G = (V, E)$ y $V = (v_1, ..., v_n)$, la matriz de adyacencia $M = |m_{ij}|$ del grafo G es una matriz de nxn definida por:

$$m_{ij} = \begin{cases} 1, & \text{si } (v_i, v_j) \in E \\ 0, & \text{de lo contrario} \end{cases} \tag{10}$$

El grafo representado por una matriz se vería de la siguiente forma:

$$M = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Figura 9: Matriz de adyacencia [23].

6.7.3. Algoritmos

En muchas aplicaciones, las ramas de los grafos G están etiquetadas con un número positivo denominado peso. Como consecuencia, se puede definir el costo de una trayectoria de G como la suma de todos los pesos de las ramas tomadas. Considerando el problema de conectar el nodo N_s al nodo N_f en G pro la ruta mínima, en planificación de movimiento, por ejemplo, el peso de cada rama generalmente representa la longitud del camino que representa. Es evidente que este camino mínimo es el de interés ya que es el camino mas corto para unir el nodo inicial con el final.

Una estrategia ampliamente utilizada para determinar la ruta mínima en un grafo es el algoritmo A*, este visita los nodos de G de forma iterativa a partir de N_s, almacenando solo las rutas mínimas actuales de N_s a los nodos visitados en un árbol T. El algoritmo emplea una función de costo $f(N_I)$ para cada nodo N_I visitado durante la búsqueda. Esta función, que es una estimación del costo del mínimo camino que conecta N_s con N_f.

6.8. Planificación de movimientos

La planificación del movimiento es el problema de encontrar el movimiento de un robot desde un estado inicial. a un estado de meta que evita obstáculos en el medio ambiente y satisface otros restricciones, como límites de unión o límites de par [18].

Al igual que con los manipuladores, el problema de planificar la trayectoria de un robot móvil puede desglosarse en encontrar un camino y definir una ley de tiempo en el camino. Sin embargo, si el robot móvil está sujeto a restricciones no holonómicas, el encontrar un camino se vuelve más difícil que en el caso de los manipuladores. De hecho, además de cumplir las condiciones de de frontera (interpolación de los puntos asignados y continuidad del grado deseado) el camino debe también satisfacen las restricciones no holonómicas en todos los puntos [22].

6.8.1. Espacio de configuración

Un concepto clave en la planificación del movimiento es el espacio de configuración, o espacio C. Cada punto en el espacio C, corresponde a una configuración única q del robot, y cada configuración del robot se puede representar como un punto en espacio C Por ejemplo, la configuración de un brazo robótico con n articulaciones se puede representado como una lista de n posiciones conjuntas, $q = (\theta_1; ...; \theta_n)$. El espacio C libre C_{free} consta de las configuraciones en las que el robot no atraviesa ningún obstáculo ni viola un límite conjunto [18].

6.8.2. Planificación de trayectorias

El problema de la planificación del trayectoria es un subproblema del problema general de planificación de movimiento. La planificación del trayectoria es el problema puramente geométrico de encontrar un camino $q(s)$ libre de colisiones desde una configuración de inicio $q(0) = q_{start}$ hasta una configuración de meta $q(1) = q_{goal}$, sin preocuparse por la dinámica, la duración del movimiento o las limitaciones en el movimiento o en las entradas de control y cumpliendo con las restricciones no holonómicas y, posiblemente, los límites de las entradas de velocidad. Se supone que el camino devuelto por el planificador de ruta se puede escalar en el tiempo para crear una trayectoria factible. En general, estos se integran en el procedimiento de diseño como la optimización de un criterio de coste adecuado a lo largo de la trayectoria. También se asume que hay disponible un controlador de retroalimentación para garantizar que el movimiento planificado se sigue de cerca, también se asume que un modelo geométrico preciso del robot y el entorno está disponible para evalúe es espacio espacio de configuración libre (sin obstáculos y sin violar limites de las juntas) durante la planificación del movimiento [22].

6.9. Controladores de posición y velocidad

Para poder garantizar que un robot móvil llegué a la posición deseada, es necesaria la implementación la implementación de un controlador. Este se encarga de controlar tanto la posición como la velocidad y además se busca que las trayectorias sean suaves y controladas en todo momento.

En control de robots diferenciales es muy común observar aplicaciones de control en donde se utiliza el error de posición, orientación y velocidad para determinar las velocidades

de referencia que el robot debe recibir para describir una trayectoria definida [24]. Existen diversos tipos de controladores que presentan comportamientos diferentes, para profundizar cada uno de los controladores se recomienda ver [1], donde se detalla el funcionamiento de cada controlador, a continuación se mencionan algunos de los controladores de este tipo mas comunes:

- Control proporcional de velocidades con saturación limitada: El cual utiliza velocidades de entrada u_1 y u_2 para poder describir la velocidad lineal y angular del robot.

- Control PID de velocidad angular: Este controlador se implementa para ajustes de posición y velocidad, este permite realizar estos ajustes mediante 3 parámetros: k_p, k_i y k_d, los cuales modifican el comportamiento del sistema.

- Control proporcional de velocidad lineal v: Este controlador utiliza los parámetros k_x y k_y en función del error de posición respecto a la meta, con el objetivo que la velocidad de convergencia hacia la meta decrezca a medida que el robot se acerca a esta [25].

Los controladores presentados anteriormente aunque facilitan el manejo de los robots diferenciales a llegar a la meta, estos no toman en cuenta la orientación o pose final que el robot tendrá cuando llegue al destino. La pose final del robot depende de la pose inicial, y teniendo en cuenta estos datos a la hora de controlar el robot se pueden lograr trayectorias con una Convergencia más suave hacia la meta [26].

Como se puede observar en [1], para aplicaciones del algoritmo ACO, se utiliza controladores como los descritos anteriormente.

6.9.1. Controlador de pose

Este controlador ya toma en cuenta los parámetros mencionados anteriormente.

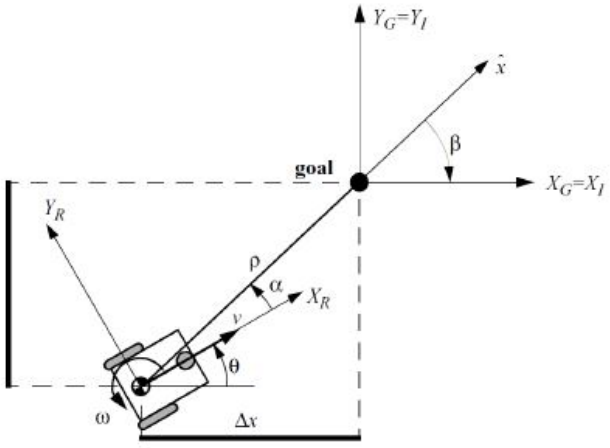


Figura 10: Cinemática de robot y referencias [26].

El objetivo de este controlador es encontrar una matriz K:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = K \cdot e = K \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}^R \tag{11}$$

Tal que el control cause que $\lim_{t\to\infty} e(t) = 0$. La relación entre las velocidades cartesianas y las velocidades angular w y v del robot esta dada por la Ecuación 12 y el error de posición entre el robot y el punto de meta e se expresa como la Ecuación 13.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{12}$$

$$e = R \begin{bmatrix} x, & y, & z \end{bmatrix}^T \tag{13}$$

Y ahora, esta relación se debe trasladar a un espacio de coordenadas polares sabiendo que α es el ángulo entre el eje X_R del marco referencial del robot y el vector entre el centro del robot y el punto de meta como se observa en la Figura 10, β es el ángulo de orientación entre el eje horizontal del marco inercial y el vector entre el robot y la meta, y ρ es la distancia euclideana del vector entre el centro del robot y el punto de meta.

Ahora en coordenadas polares, la relación de velocidades cuando el robot está orientado en dirección a la meta está dada por la Ecuación 14 y cuando el robot está orientado de espaldas a la meta los signos se invierten [26].

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos(\alpha) & 0 \\ \frac{\sin(\alpha)}{\rho} & -1 \\ \frac{-\sin(\alpha)}{\rho} & 0 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{14}$$

Por ultimo las ecuaciones que definen este controlador son:

$$\alpha = -\theta + \theta_g \tag{15}$$

$$\beta = -\theta + \alpha \tag{16}$$

$$v = k_p * \rho \tag{17}$$

$$w = k_\alpha * \alpha + k_\beta * \beta \tag{18}$$

6.9.2. Controlador de pose de Lyapunov

Este es un controlador de pose similar al de la sección anterior, pero utiliza leyes de control para las velocidades lineal y angular diferentes, las ecuaciones de conversión a coordenadas polares son las mismas y la relación de velocidades también utiliza las mismas

ecuaciones dependiendo del ángulo α. El objetivo de este controlador es que el estado del sistema converja a $(\rho, \alpha, \beta) = [0, 0, \beta]$ en un tiempo finito.

Para asegurar la convergencia del robot hacia la meta, se utiliza el Criterio de Estabilidad de Lyapunov, el cual afirma que, si existe una solución en las cercanías de un punto de equilibrio X_o de una ecuación diferencial homogénea $\dot{X} = f(X)$, esta se mantiene cerca del punto de equilibrio para todo $t > t_o$, además que este punto es asintóticamente estable y que es un atractor de soluciones que convergen a X_o cuando $t \to \infty$. Las ecuaciones que definen este controlador son las mismas en el caso de los parámetros α y ρ pero las velocidades están definidas por:

$$v = k_p \ \rho \ \cos\alpha \tag{19}$$

$$w = k_p \ \sin\alpha + k_\alpha \ \alpha \tag{20}$$

6.10. Lenguaje C++

El lenguaje de programación C++ proporciona muchos mecanismos para expresar abstracciones y relaciones entre ellos, algunos de estos, como algunas rutinas y estructuras de datos, se proporcionan en un lenguaje ampliamente utilizado en la programación científica y de ingeniería [27].

6.10.1. Programación orientada a objetos

El mecanismo en C++ que permite expresar estas abstracciones es la programación orientada a objetos. Las tres ideas principales que caracterizan la programación orientada a objetos son: objetos, jerarquía clase y polimorfismo.

Un objeto tiene datos de estado y funciones de comportamiento. Los objetos combinan datos que representan el estado con funciones (subrutinas) que acceden o modifican los datos. La jerarquia de clase permite organizar clases para su reutilización con clases en la parte inferior heredando operaciones de las clases de arriba. El polimorfismo permite que diferentes tipos de objetos que comparten un comportamiento común se utilicen en un código que solo requiere ese comportamiento.

6.10.2. Clases

Una clase es un nuevo tipo de dato que puede ser usado para crear objetos. Específicamente, una clase crea una consistencia lógica que define una relación entre sus miembros. Cuando se declara una variable de una clase, se está creando un objeto. Estas nos permite asociar datos y funcionalidad relacionada en un solo objeto. La idea principal es separar la data de la funcionalidad y tratarlas cada una por separado.

Una clase es en general un modelo, receta o plantilla que define el estado y comportamiento de cierto tipo de objetos. Una clase puede pensarse como una colección de variables

(atributos o propiedades) y funciones (métodos) que permiten representar un conjunto de datos y especificar las operaciones o procedimientos que permiten manipular tales datos. Se puede inclusive entender una clase como un tipo de dato personalizado, similar a las estructuras, donde cada programador define los miembros que va a tener su tipo de dato. De hecho, los tipos de dato nativos de C++ son en realidad clases.

Especificadores de acceso

Hay tres especificadores de acceso en C++: *public*, *private* y *protected*. Cuando se declara público (*public*) un miembro de una clase, usted permite el acceso a tal miembro desde dentro y fuera de la clase. Los miembros de datos que son declarados protegidos (*protected*) son únicamente accesibles por funciones miembro de la clase, pero no se pueden acceder a ellos desde otras clases. Cuando un miembro de una clase es declarado privado (*private*) es inaccesible otras clases y otras partes del programa [27].

6.10.3. Programación multihilos

Desde hace bastantes años, los sistemas operativos son multitarea, lo que les permite ejecutar diversos procesos de forma simultánea. Esta capacidad ha permitido el hecho de optimizar la programación hacia modelos multiproceso o multihilo. En programación, se refiere a los lenguajes de programación que permiten la ejecución de varias tareas en forma simultánea.

Un hilo o *thread* es básicamente una parte o procesos pequeños independientes de un proceso grande. También podemos decir que un hilo es un flujo de ejecución único dentro de programa que se ejecuta en su propio espacio de direcciones (proceso). Los subprocesos no pueden ejecutarse solos, se ejecutan dentro del programa, porque necesitan la supervisión del proceso principal para ejecutarse. Se pueden organizar varios subprocesos de ejecución para que se ejecuten simultáneamente en el mismo programa. Se puede llegar a pensar que los procesos son análogos a las aplicaciones o a programas aislados, pero realmente tiene asignado espacio propio de ejecución dentro del sistema.

Una de las ventajas de la programación multihilo es cuando dentro de una aplicación queremos priorizar alguno de los procesos que comporta sobre el resto. Es muy habitual usar este tipo de programación en videojuegos priorizando la parte de renderización de la imagen sobre otros procesos "paralelos" que se ejecutan a la vez. En el artículo [28], se propone una implementación multihilos del algoritmo SHO donde se interrumpe periódicamente la ejecución en todos los hilos para comparar los resultados y aplicar técnicas de cruce entre ellos, incorporando elementos de recocido simulado y algoritmos genéticos, para mejorar el rendimiento del algoritmo ya planteado.

CAPÍTULO 7

Validación de plataforma y lenguaje de programación

En este capítulo se presenta la validación de la selección del microcontrolador a utilizar así como del lenguaje de programación utilizado. Para poder realizar una correcta selección para realizar la migración del algoritmo se realizó un *trade study*, en el cual se comparan los parámetros y especificaciones de todas las opciones planteadas con el objetivo de elegir la mejor opción posible.

7.1. Selección del microcontrolador

Las opciones de plataforma de desarrollo evaluadas fueron: Tiva-C, Raspberry Pi, Arduino Uno y PIC. Estas fueron seleccionadas debido a que se tiene experiencia con las cuatro y son de las más utilizadas en distintos tipos de proyectos. Las cuatro plataformas son muy diferentes entre sí y no son muy comparables, ya que han sido diseñados para distintos propósitos. Sin embargo se seleccionaron distintos criterios que se consideraron fundamentales para el desarrollo de este trabajo con los cuales se podrán comparar estas plataformas.

Al tener las opciones definidas de microcontroladores se procedió a evaluar distintas opciones de robots móviles disponibles, dentro de las cuales se encuentra uno de los robots móviles más comunes, el E-puck, descrito en la sección 6.5, el cual es uno de los más utilizados en temas de investigación dada su versatilidad. Una variante de este robot es el Pi-puck que como se menciona en la sección 6.6 tiene adaptada la interfaz RPi.

Para la migración del algoritmo se plantearon las distintas opciones de plataforma y se seleccionaron los criterios de comparación con los cuales se seleccionará la mejor plataforma. En el Cuadro 1 se presentan los criterios utilizados en el *trade study* así como el peso que se

le asignó a cada uno. El peso de cada uno de los criterios fue seleccionado de forma que la suma de estos forme el 100 % y se le asignó el valor con lo que se consideró más importante, se tiene un total de 10 criterios a considerar para una mejor toma de decisión.

Cuadro 1: Criterios de comparación

Criterios	Peso
Costo	7.000
Disponibilidad	12.000
Memoria	7.000
Adaptabilidad a robots móviles	17.000
Unidad de procesamiento	9.000
Periféricos incluidos	8.000
Entorno de desarrollo	10.000
Conectividad	15.000
Potencia de cálculo	8.000
Velocidad de procesamiento	9.000

7.1.1. Criterios para el *Trade Study* de las plataformas

El *trade study* es un método de toma de decisiones que se utilizan para identificar la solución técnica más aceptable entre un conjunto de soluciones propuestas. Los *trade studies* proporciona un medio eficaz para abordar el proceso de toma de decisiones, ya que este método clasificará las soluciones propuestas asignando un valor numérico a cada una. Esta clasificación se basa en factores de peso de cada uno de los criterios a evaluar [29], [30].

En los trabajos [31] y [32] se realizaron *trade studies* el primero para un análisis realizado la NASA en dirección de misiones y el segundo para el análisis de una aeronave de resistencia a una altitud elevada. Con estos trabajos se respalda la selección de los pesos en un rango de uno a diez para cada uno de los criterios presentados a continuación.

Costo

De no contar con los dispositivos necesarios se deben realizar la compra de los dispositivos faltantes. A este criterio se le asignó el peso del 7 %. En este criterio se le asigna la ponderación más alta al PIC, ya que este microcontrolador es el que tiene el precio mas bajo, este cuesta alrededor de $9.00, comparado con las otras plataformas, este sale ganador. Si se compara el precio de este con la RPi 4 es casi quince veces más barato, por esto la ponderación más baja fue de la RPi en el Cuadro 12 se puede ver el precio de las otras plataformas, así como características importantes que serán mencionadas en los criterios de evaluación.

Cuadro 2: Ponderación de costo

Alternativas	Costo
Raspberry	**1**
PIC	**10**
Tiva-C	**7**
Arduino	**8**

Disponibilidad

Este criterio es considerado de los más importantes, se deben tener varios dispositivos para poder validar la comunicación entre agentes. El peso asignado a este criterio es del 12 %.

En esta sección las ponderaciones más altas las tiene la Raspberry Pi y la Tiva-c, ya que en la Universidad del Valle de Guatemala se cuenta con suficientes dispositivos para poder realizar las pruebas establecidas para la validación. En el caso del PIC y del Arduino Uno se le asigna una podrición media ya que estos son fácil de conseguir.

Cuadro 3: Ponderación de disponibilidad

Alternativas	Disponibilidad
Raspberry	**10**
PIC	**5**
Tiva-C	**10**
Arduino	**7**

Memoria

A este criterio se le asigna un peso del 7 %. Como se ve en los Cuadros 12, 13 y 14, donde se observa la capacidad de memoria de las plataformas, la Raspberry Pi es muy superior en este ámbito por lo que la ponderación más alta la tiene dicha plataforma.

Cuadro 4: Ponderación de memoria

Alternativas	Memoria
Raspberry	**10**
PIC	**1**
Tiva-C	**3**
Arduino	**4**

Adaptabilidad a robots móviles

Este es el criterio que se considera como el más importante, considerando una futura implementación a una plataforma móvil. Por esta razón se le da un peso del 17 %.

La ponderación más alta en este criterio se le asigna a la Raspberry Pi, el criterio principal es tener un antecedente de la implementación de un robot móvil utilizando esta plataforma, como se menciona en el Capítulo 6.6. Aparte de esta razón, este ordenador cuenta con salidas de video HDMI y varios puertos USB. En el caso del Arduino y Tiva recibe una ponderación media dado la cantidad de pines que se tiene y los distintos módulos que se pueden implementar para estas plataformas, como en el caso de la Tiva-c el *BoosterPack CC3100* para una conexión a una red WiFi o utilizar un módulo *ESP8266* el cual permite conectarse a una red o crear una red propia. En el caso del PIC, la implementación de los módulos necesarios para una implementación será necesario el uso de una PCB para la conexión de los pines, por esta razón el PIC recibe la ponderación más baja.

Cuadro 5: Ponderación de adaptabilidad

Alternativas	Adaptabilidad a robots móviles
Raspberry	**10**
PIC	**3**
Tiva-C	**7**
Arduino	**7**

Unidad de procesamiento

El peso asignado a este criterio fue de 9 %. Con base en las características observadas en los Cuadros 12, 13 y 14, la ponderación más alta fue asignada a la RPi, la cual demuestra una unidad de procesamiento mucho más avanzada que las otras plataformas.

Cuadro 6: Ponderación de unidad de procesamiento

Alternativas	Unidad de procesamiento
Raspberry	**10**
PIC	**3**
Tiva-C	**7**
Arduino	**5**

Periféricos incluidos

El peso asignado a este criterio fue de 7 %. Para este criterio la RPi tiene superioridad en comparación con las otras plataformas. Esta posee más pines de entrada y salida los cuales pueden ser configurados para distintos tipos de comunicación. Asimismo cuenta con entradas USB a los cuales puede ser conectado teclados, monitores y ratón. Además cuenta con puertos

HDMI, conexión a WiFi o Ethernet y dispositivo de Bluetooth. Aparte la RPi cuenta con muchos periféricos adicionales que pueden ser de utilidad para futuras implementaciones como lo puede ser la *PiCam*, que es una cámara pequeña para captura de video, también se puede implementar alguna pantalla táctil para el ingreso y visualización de datos.

Cuadro 7: Ponderación de periféricos incluidos

Alternativas	Periféricos incluidos
Raspberry	**10**
PIC	**3**
Tiva-C	**6**
Arduino	**5**

Entorno de desarrollo

Este criterio también es considerado importante ya que se debe tener un buen entorno para facilitar la implementación de este trabajo, el peso asignado a este criterio fue de 10 %. Las ponderaciones más altas fueron de la Tiva-c y del Arduino UNO, ya que estas dos plataformas poseen su propio entorno de desarrollo integrado los cuales son Arduino IDE y Energia. Ambos poseen funciones propias que facilitan el desarrollo de los programas.

En cuanto a la RPi se le asigna una ponderación alta ya que al ser un ordenador se le puede instalar muchas herramientas para el desarrollo del programas que resultan muy útiles para el desarrollo de un proyecto.

Cuadro 8: Ponderación de entrono de desarrollo

Alternativas	Entorno de desarrollo
Raspberry	**8**
PIC	**3**
Tiva-C	**10**
Arduino	**10**

Conectividad

La conectividad es el segundo criterio con más peso, la razón de esto es que para este trabajo se necesita una comunicación distintos dispositivos para poder validar el funcionamiento, el peso asignado a este criterio fue de 15 %.

Se le asigna a la RPi la ponderación más alta debido a las distintas herramientas que posee. Como se puede observar en el Cuadro 12, este ordenador posee conexión wireless, bluetooth y varios puertos USB.

Cuadro 9: Ponderación de conectividad

Alternativas	Conectividad
Raspberry	**10**
PIC	**5**
Tiva-C	**6**
Arduino	**6**

Potencia de cálculo y velocidad de procesamiento

Estos criterios también son considerados importantes, ya que depende de ellos obtener un buen desempeño a la hora de correr el algoritmo. En el trabajo [33], se analiza la complejidad de los algoritmos ACO. El tiempo de computación de orden $O(m\ log(m))$ es suficiente para obtener un resultado óptimo del algoritmo. Dada la complejidad del algoritmo y también basándose en el periodo de muestreo utilizado para las pruebas realizadas en [1], se puede decir que todos los dispositivos son capaces de correr el algoritmo. A pesar de esto, se le dio mayor ponderación a la RPi debido a que posee un CPU considerablemente más rápida como se ve en las características del Cuadro 12.

Cuadro 10: Ponderación de potencia de calculo

Alternativas	Potencia de cálculo
Raspberry	**10**
PIC	**5**
Tiva-C	**8**
Arduino	**7**

Cuadro 11: Ponderación de velocidad de procesamiento

Alternativas	Velocidad de procesamiento
Raspberry	**10**
PIC	**3**
Tiva-C	**8**
Arduino	**7**

Cuadro 12: Características de la Raspberry Pi

	Raspberry Pi 3B	Raspberry Pi 4
SoC	BCM2837	BCM2711
CPU	Quad Cortex A54 @ 1.2 GHz	Quad Cortex a72 @ 1.5 GHz
Entorno de desarrollo	ARMv8-A	ARMv8
GPU	VideoCore IV 400 MHz	VideoCore VI
RAM	1 GB SDRAM	1 GB /2 GB /4GB SDRAM
Almacenamiento	MicroSD	MicroSD
Ethernet	10/100	10/100/1000
Wireless	802.11n/Bluetooth 4.0	802.11ac/Bluetooth 5.0 BLE
Salidas de video	HDMI/Compuesto	2 x micro -HDMI
Precio	$ 100.00	$ 150.00

Cuadro 13: Características de Arduino Uno y Tiva-c

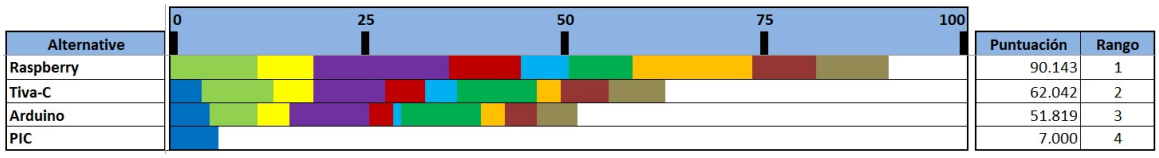
	Arduino Uno	Tiva C
SoC	ATmega328	TM4C123GH6PM
CPU	ATmega328p @ 16 MHz	ARM Cortex -M4F @80 MHz
Entorno de desarrollo	Arduino IDE	Energia
GPU	ATmega328	ARM Cortex
RAM	2 KB	32 KB SRAM
Almacenamiento	EEPROM 1 KB	EEPROM 2 KB
Ethernet	No	No
Wireless	No	No
Salidas de video	No	No
Precio	$ 15.00	$ 23.00

Cuadro 14: Características del PIC

	PIC 16F
SoC	PIC 16F
CPU	20MHz
Entorno de desarrollo	RISC
GPU	NA
RAM	68 Bits
Almacenamiento	EEPROM 64 bits
Ethernet	No
1 Wireless	No
Salidas de video	No
Precio	$ 9.00

7.1.2. Resultados

Con los pesos y ponderaciones de los criterios presentados en la sección anterior se procede a obtener los resultados del *trade study*. Como se puede observar en la Figura 12, los colores que se observan corresponden a los criterios dados por el Cuadro 1. Según el puntuación final, se puede ver que la RPi es la mejor opción como plataforma. Por lo tanto, el ordenador Raspberry Pi es elegido como plataforma para poder realizar la migración del algoritmo ACO. El modelo a utilizar será la RPi 4 ya que es el modelo que se tiene disponible.



Alternative	Puntuación	Rango
Raspberry	90.143	1
Tiva-C	62.042	2
Arduino	51.819	3
PIC	7.000	4

Figura 11: Resultados del *Trade Study*.

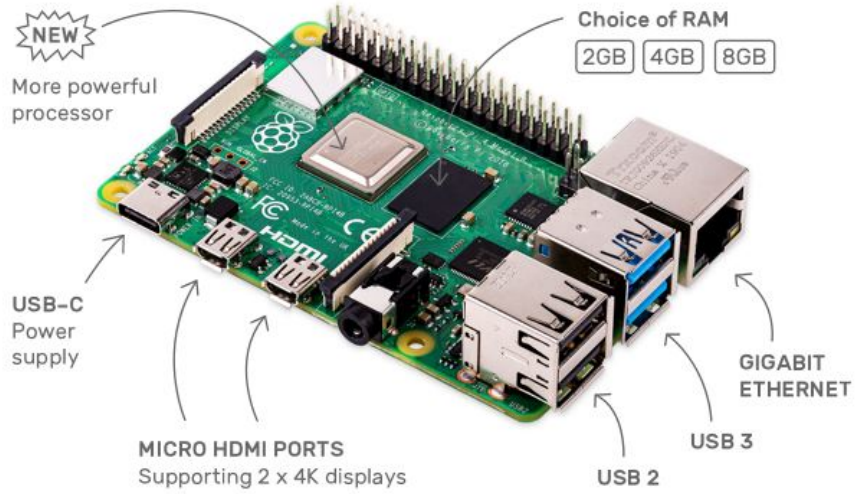


Figura 12: Plataforma seleccionada.

7.2. Selección del entorno y lenguaje de programación

Una vez seleccionada la plataforma con la que se trabajará (RPi), se procede a seleccionar el lenguaje de programación a utilizar,dado que la programación orientada a objetos es fundamental para el desarrollo del algoritmo, se plantean dos opciones que posean este tipo de programación: *lenguaje C++* y *python*.

En la Figura 14 se puede ver la percepción de dificultad que tienen los programadores al utilizar distintos aspectos en ambos lenguajes de programación. Para este estudio se utilizó

una escala de uno a cinco donde: el valor de uno significa que es fácil y cinco que presenta mayor dificultad. Al observar los resultados no se puede observar una amplia diferencia entre ambos lenguajes, sino que en todos los parámetros la dificultad es muy similar.

7.2.1. Comparación de los dos lenguajes

Para poder comparar ambos lenguajes y seleccionar el más apto se tomaron las consideraciones dadas en los artículos [34] y [35].

La principal diferencia entre los dos lenguajes es el tipo de compilación. Cuando se compila en C++ se utiliza un compilador que convierte el código fuente y produce un ejecutable, el cual puede ser corrido como un programa. Por otro lado python compila en código fuente como en C++, pero con la diferencia que compila para generar un *bytecode*, el cual no corre por el procesador sino que interpreta el código utilizando *python virtual machine*, estos procesos se ilustran en la Figura 13. Interpretar el código es más lento que correr un código nativo directamente.

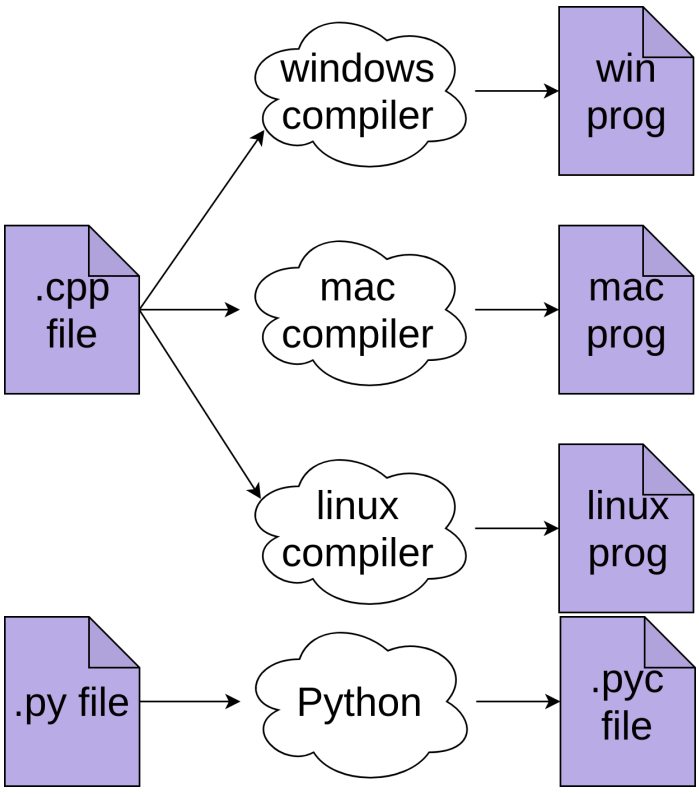


Figura 13: Tipo de compilación [34].

	Python		C++	
Construct	**Mean**	**Std Dev**	**Mean**	**Std Dev**
Algorithms	2.64444	0.88306	2.5556	1.0347
Variables	2	0.87905	2.2222	0.9975
List/Arrays	2.75556	1.19003	2.6889	1.0406
Boolean Expressions	2.47619	1.08736	2.5714	1.1293
For Loop	2.04444	0.76739	2.7273	1.0424
While Loop	2.15556	0.9524	2.6818	1.0292
If Statement	1.84091	0.68005	2.1429	0.9258
Functions	2.95556	1.0435	2.7619	0.983
Files	3.75	0.98058	---	---

	Python		C++	
Feature / Aspect	**Mean**	**Std Dev**	**Mean**	**Std Dev**
Ease to Program	2.47727	0.731	2.6818	0.9092
Expressiveness	2.76923	0.70567	2.8	0.791
Simplicity	2.44444	1.05649	2.6279	1.1552
Richness of Modules	2.4419	0.90219	2.7404	0.9386
Flexibility	2.57778	0.72265	2.8605	0.8614
Satisfaction	2.57778	0.86573	2.9556	1.0435
Debugging Facility	2.42222	1.01105	2.6744	0.9933
Compactness	2.51111	0.89499	2.8667	1.0135

Figura 14: Dificultad del lenguaje [35].

Otros aspectos importantes que se debe mencionar es el tipo de datos que se utilizan. En C++ se utiliza de tipo estático, lo que significa que cada variable debe tener un tipo como: *int*, *char*, entre otros. La ventaja de esto es que se puede saber de que tipo es una variable en particular, por lo que al mandarle la variable a una función debe coincidir con el tipo de dato que se este manejando.

Por otro lado python utiliza tipo de datos dinámico, este nos permite mayor flexibilidad ya que nos permite utilizar cualquier tipo de variable cuando se necesite. Sin embargo puede ser un problema ya que se le puede estar mandando un tipo de variable no permitido a cierto objeto o función.

Una de las mayores diferencias cuando se compara Python con C ++, es cómo manejan la memoria. Python no tiene punteros y tampoco permite manipular la memoria directamente. En C++ se debe tener en cuenta si la data que se busca acceder se encuentra en la memoria *heap* o en la memoria *stack*, en python no se debe preocupar por esto. Sin embargo hay ocasiones que se busca liberar memoria asignada por lo que se python utiliza *garbage collector* que encuentra memoria no utilizada y la libera, nuevamente este proceso es mucho más lento.

7.2.2. Resultados

Considerando las diferencias presentadas anteriormente y sabiendo que los dos lenguajes pueden ser utilizados ya que ambos poseen las librerías y herramientas necesarias (como programación multihilos) para la migración del algoritmo, y tomando en cuenta el criterio de la experiencia. Se decide utilizar el lenguaje de programación C++, ya que se tiene más conocimiento de este lenguaje en comparación con python.

Figura 15: Lenguaje de programación.

7.2.3. Entorno de desarrollo

Dada la versatilidad del ordenador RPi, se pueden considerar diversas opciones de entorno de desarrollo. Para este trabajo se decide utilizar el editor de texto *K-write* utilizando IDE *Geany* y *Visual Studio Code*. La razón de utilizar estas dos opciones es que ambas son fáciles de instalar y utilizar.

Aparte de estos editores de texto, para poder compilar y ejecutar los programas, se decide utilizar la terminal propia de la Raspberry Pi y también la consola instalada en *Visual Studio Code*. Ambas realizan el mismo trabajo de compilar y correr los ejecutables que se generen.

Figura 16: Entornos de desarrollo.

CAPÍTULO 8

Implementación del *Ant Colony Optimization*

En este capítulo se muestra la metodología utilizada para la implementación del algoritmo ACO en un sistema físico. Ya que no se cuenta con una plataforma del robot móvil funcional, el sistema físico comprende el uso de un RPi conectada a una red y la plataforma de rastreo por visión de computadora que posee una cámara que enviará las coordenadas de la RPi. La plataforma móvil que se está desarrollando está basada en una plataforma de Raspberry Pi, por lo que todo lo realizado a continuación podrá ser implementado en dicha plataforma y realizar las mismas pruebas pero con el robot en movimiento.

Este trabajo toma como base la fase anterior a esta tesis definida en la sec. 2.6. Para las primeras pruebas se toman en cuenta los valores de los parámetros ya calculados para un funcionamiento óptimo del algoritmo, los cuales fueron calculados mediante un barrido de parámetros por Gabriela Iriarte en su trabajo de graduación [1]. Estos parámetros fueron modificados debido al comportamiento que presentó el sistema al correr el algoritmo en las primeras pruebas.

8.1. Migración del algoritmo

Para la implementación del ACO, se implementa el uso de clases en C++. Para inicializar es necesario definir las funciones que se utilizarán para la implementación del algoritmo, asimismo se definen todos los espacios de memoria y los tipos para las variables a utilizar dentro del algoritmo.

8.1.1. Creación de la clase ACO

Se procede a definir todas las funciones ya creadas. Para poder crear la clase ACO fue necesario el uso de punteros, doble punteros y funciones void. Estos fueron agrupados en publico, si se accederán en el *main file* y como privado si estas serán utilizadas en los cálculos internos del algoritmo.

En el diagrama mostrado en la Figura 17 se detallan todos los miembros que posee la clase ACO, en el cual se puede observar las funciones y variables utilizadas para cada actividad principal del algoritmo. Por motivos de visualización se decidió dividir en esas cuatro secciones para poder ver de mejor manera el funcionamiento del programa, dividiendo las funciones que se utilizan en conjunto para una tarea especifica del algoritmo. Además se indica si cada miembro es publico o privado con los símbolos + y -, respectivamente.

En la primer sección se colocaron las funciones utilizadas para calcular la ruta, en la segunda casilla las funciones utilizadas para imprimir resultados, en la tercer casilla la función para inicializar el algoritmo y reservar la memoria. Por último se colocaron las funciones que servirán para inicializar las variables y matrices con sus respectivos valores.

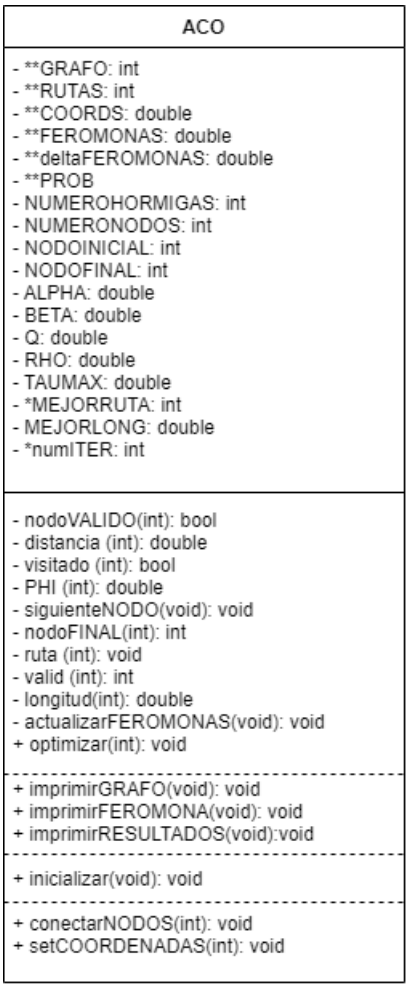


Figura 17: Diagrama de la clase.

8.1.2. Configuración de parámetros e inicialización

Para poder inicializar el algoritmo, este necesita ciertos parámetros que serán utilizados por los miembros de la clase, por lo que al principio el programa se le asignan estos parámetros y se ejecuta la inicialización, al hacer esto a cada puntero o dobles punteros según sea el caso se le asignarán los espacios de memoria correspondientes ya sea a la cantidad de hormigas o a la cantidad de nodos del grafo. El espacio de memoria para las variables fue asignado al *heap memory*, esto con el objetivo que en todo momento tengamos ese espacio reservado y accesibilidad a la información en ese espacio y no se tenga problemas al acceder a dichos espacios.

Una vez asignados los espacios de memoria se asignan un valor inicial que se irá actualizando a medida que el algoritmo vaya calculando la ruta y servirá para saber si el nodo donde se encuentra la hormiga ya fue visitado y por cuántos agentes este ha sido visitado a través de la cantidad de feromona.

8.1.3. Determinación de la trayectoria

Para iniciar el proceso de planificar la trayectoria, para lograr la mejor ruta se inicializa dando a cada hormiga un nodo inicial en el cual partirán todas y un nodo final donde deberán terminar. Las variables presentadas anteriormente y con el espacio de memoria reservados, representan la matriz de adyacencia. La matriz de pesos asignados para este algoritmo están descritos en la variable *FEROMONA()*, esta asigna un valor de feromona aleatoria a cada uno de los nodos validos de la matriz. Una vez inicialice el proceso del algoritmo los nodos con mayor valor de feromona tendrán mayor probabilidad de ser escogidos en la función por la hormiga, este proceso se realiza en la función *ruta(k)*.

El proceso de planificación de trayectoria sigue la lógica presentada en el siguiente pseudo código y diagrama de flujo.

```
1  Inicializar parametros
2  hasta se cumpla el numero de iteraciones:
3     for cada hormiga :
4          hasta encontrar el nodo destino:
5              caminar al siguiente nodo segun la ecuacion de probabilidad
6          end
7     end
8     for cada arista:
9          evaporar feromona
10         actualizarr feromona segun largo de cada arista
11    end
12 return el camino encontrado
```

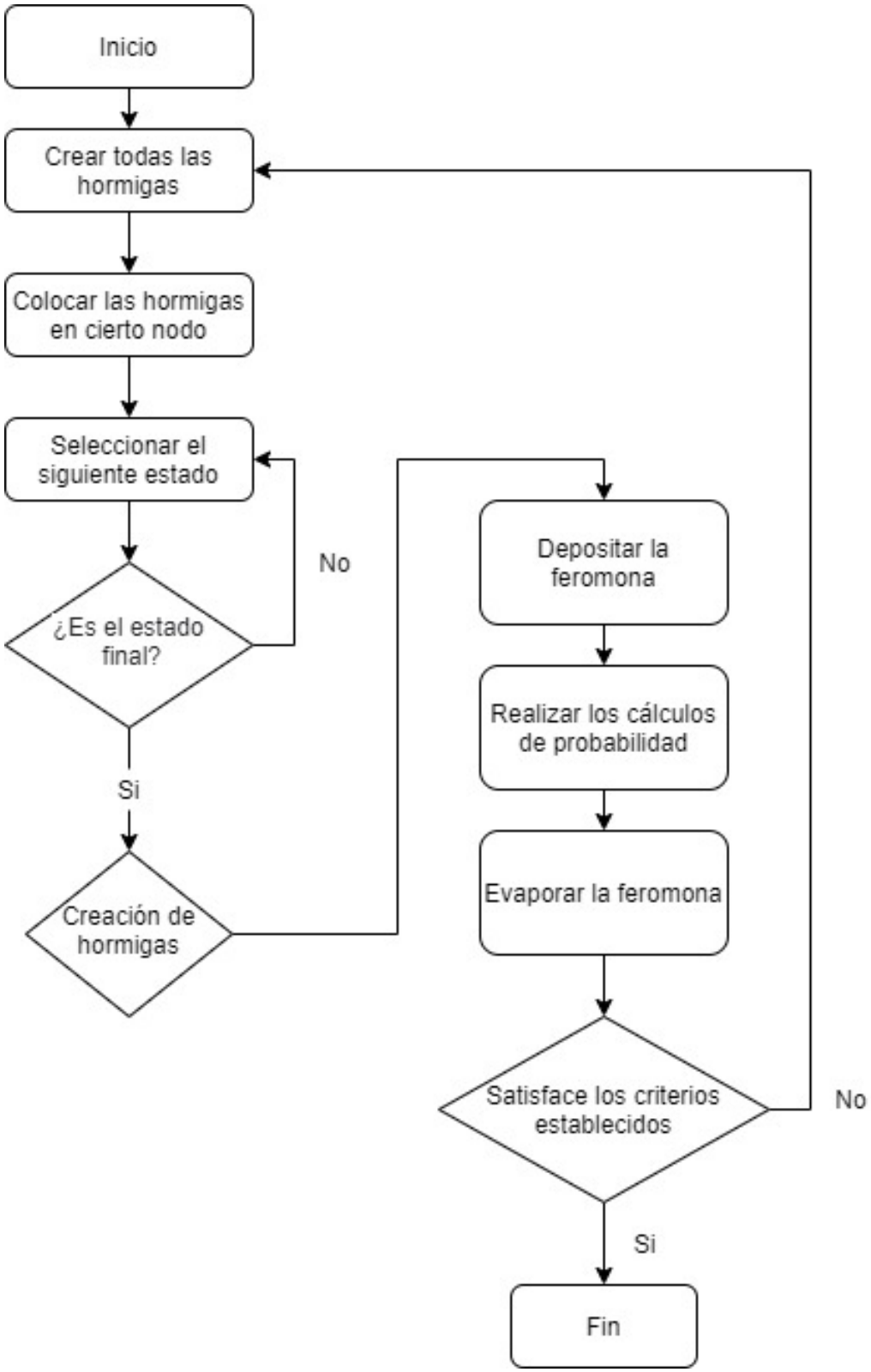


Figura 18: Diagrama de flujo del ACO.

Para los primeros experimentos de planificación de trayectoria se utilizaron los siguientes parámetros para el algoritmo.

Cuadro 15: Parámetros del algoritmo

Parámetros	Valor
alfa	1.3
beta	1
rho	0.5
Q	2.1
Iteraciones	25
Tau máximo	2

8.2. Variables del ACO

En esta sección se presenta la inicialización de las variables de almacenamiento del algoritmo ACO implementado en la RPi y la forma en que estas variables fueron estructuradas para un correcto funcionamiento del planificador de trayectorias.

8.2.1. Inicialización

Como se explica en secciones anteriores al realizar la inicialización del algoritmo, los espacios reservados de la memoria representan las matrices de adyacencia. A continuación se presenta un ejemplo de un grafo utilizado para la implementación del algoritmo así como la matriz de pesos que representa la feromona en cada arista. Estas matrices que se irán actualizando en cada iteración durante el proceso.

En la Figura 19 se puede observar un ejemplo de una matriz creada. Los nodos identificados con uno representan los nodos a los cuales se puede desplazar una hormiga estando en el nodo de la columna. Es decir, una hormiga que se encuentra en el nodo cero, se puede mover al nodo uno, dos, tres o cuatro. En cambio se puede observar que los nodos siete, ocho y nueve no tienen ninguna conexión por el momento por lo que si se encuentran en esos nodos, la hormiga no se podrán desplazar a otro lado. También se puede observar en la matriz de la feromona, los valores se muestran únicamente en los nodos que son válidos, estos valores son asignados de manera aleatoria.

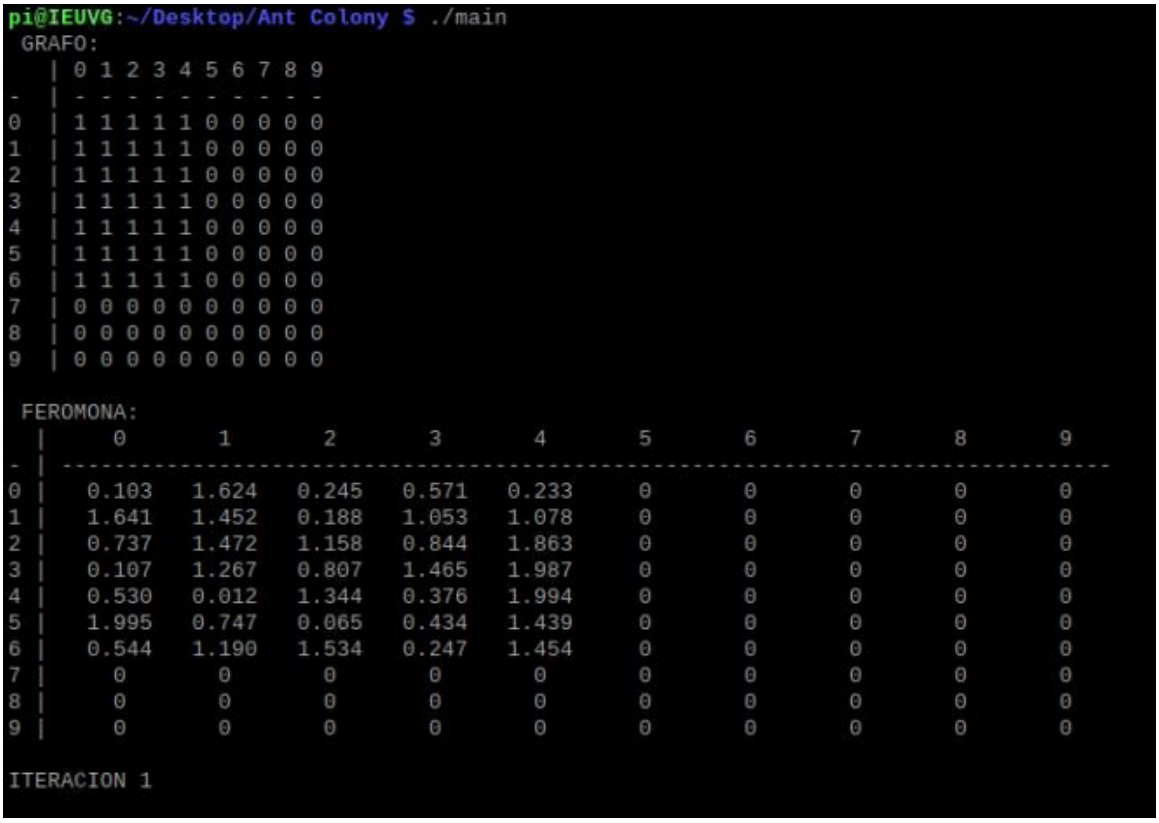


Figura 19: Variables de almacenamiento.

8.3. Validación del ACO

Para validar el funcionamiento del algoritmo se decidió realizar varias pruebas corriendo el algoritmo ACO en C++ desde la RPi con distintos puntos finales. Estas rutas calculadas fueron comparadas con la ruta que se calcula en Matlab. En esta sección se presentan tablas y gráficas con los resultados obtenidos.

8.3.1. Resultados del *Ant System*

Como primera validación se implemento el algoritmo ACO, específicamente la variante del algoritmo *Ant System* en el ordenador *Raspberry Pi* para poder comparar el funcionamiento con el algoritmo desarrollado en fases pasadas descritas en la sección 2.6. Para poder tener una métrica de comparación valida, en ambas plataformas se le asignaron los mismos parámetros. Para poder llegar a unos parámetros óptimos, se empezó con los valores del Cuadro 15, con estos parámetros se corrió el algoritmo, se observó que los resultados no eran los esperados ya que la ruta generada no presentaba un comportamiento lógico ni tampoco una ruta con movimientos suaves como se ve en la Figura 20. A partir de estos resultados se empezó a barrido de parámetros de forma manual y observar el impacto que tenían cada uno en el comportamiento del sistema. Durante estas pruebas se puedo observar

que el parámetro más sensible era el valor *beta*, seguido por Q. Al final se llegó a unos parámetros óptimos, estos se pueden ver en el Cuadro 16, los cuales fueron utilizados más adelante para la validación.

Se evaluó los nodos escogidos por el algoritmo en ambas plataformas. Los resultados de esta prueba se pueden ver en los Cuadros 17, 18, 19, 20, 21, 22. También se graficaron los resultados obtenidos de la RPi para poder observar el resultado de la ruta seleccionada. Estas gráficas se observan el las Figuras 21, 22, 23, 24, 25, 26, 27 y 28.

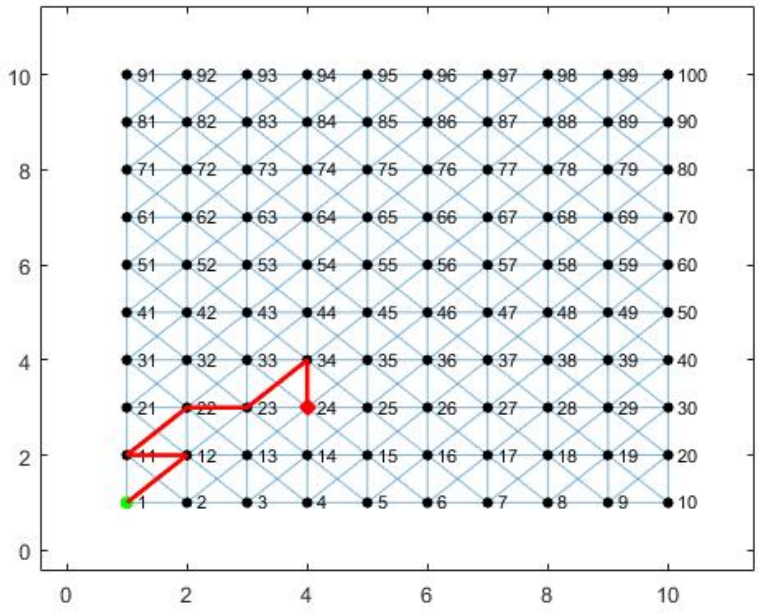


Figura 20: Ruta calculada en las primeras pruebas.

Cuadro 16: Parámetros óptimos del algoritmo

Parámetros	Valor
alfa	1
beta	0.1
rho	0.5
Q	10
Tau máximo	2
Iteraciones	75
Hormigas	50

Cuadro 17: Compraración de la ruta entre Matlab y C++

RUTA AL NODO 23	
Matlab	C++
1	1
12	12
23	23

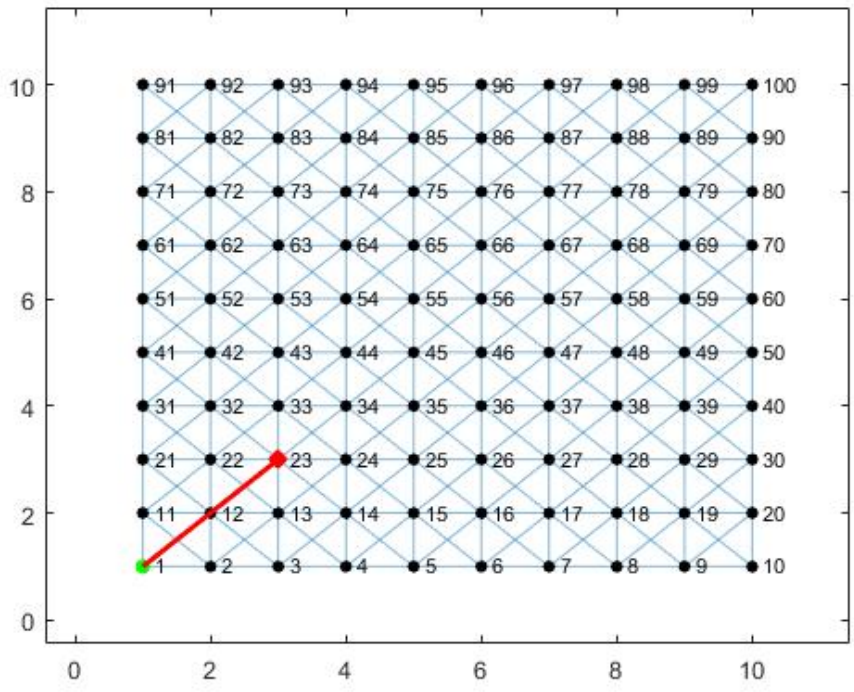


Figura 21: Ruta seleccionada al nodo 23

Cuadro 18: Compraración de la ruta entre Matlab y C++

RUTA AL NODO 24	
Matlab	C++
1	1
12	2
23	13
24	24

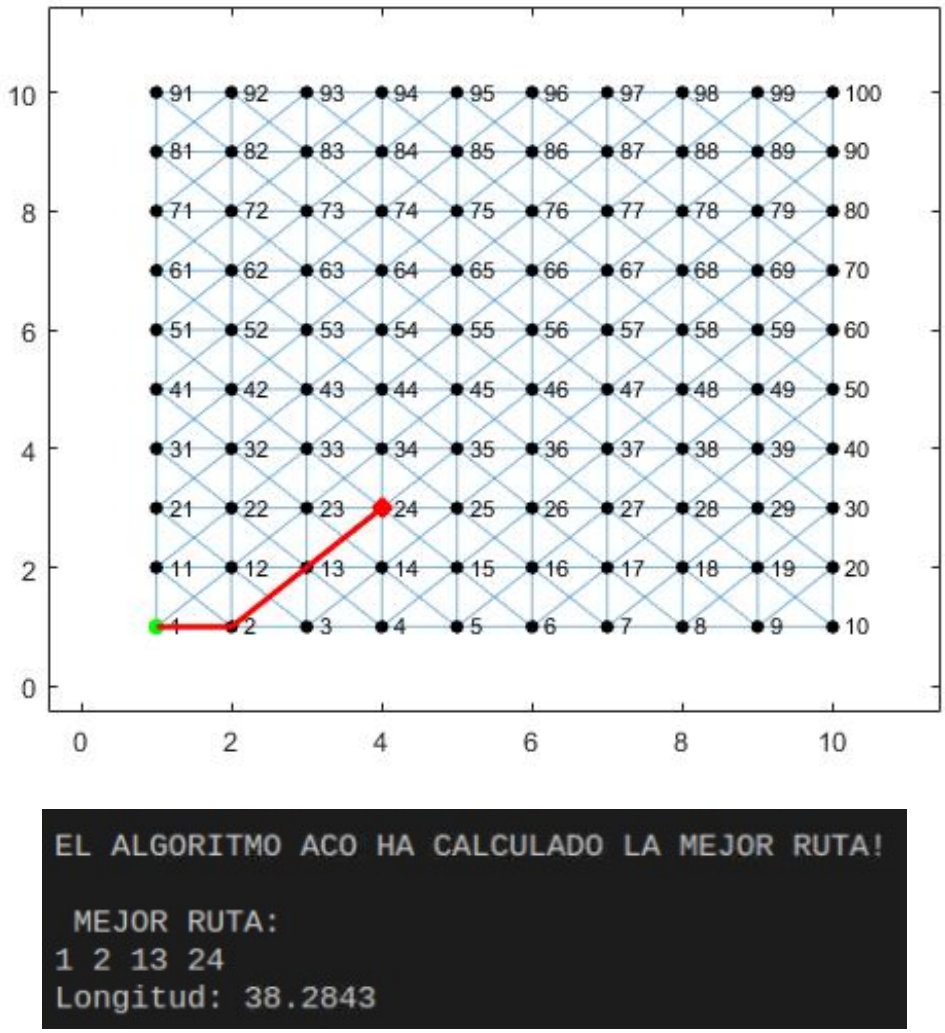


Figura 22: Ruta seleccionada al nodo 24.

Cuadro 19: Compraración de la ruta entre Matlab y C++

RUTA AL NODO 25	
Matlab	C++
1	1
12	12
3	13
14	24
25	25

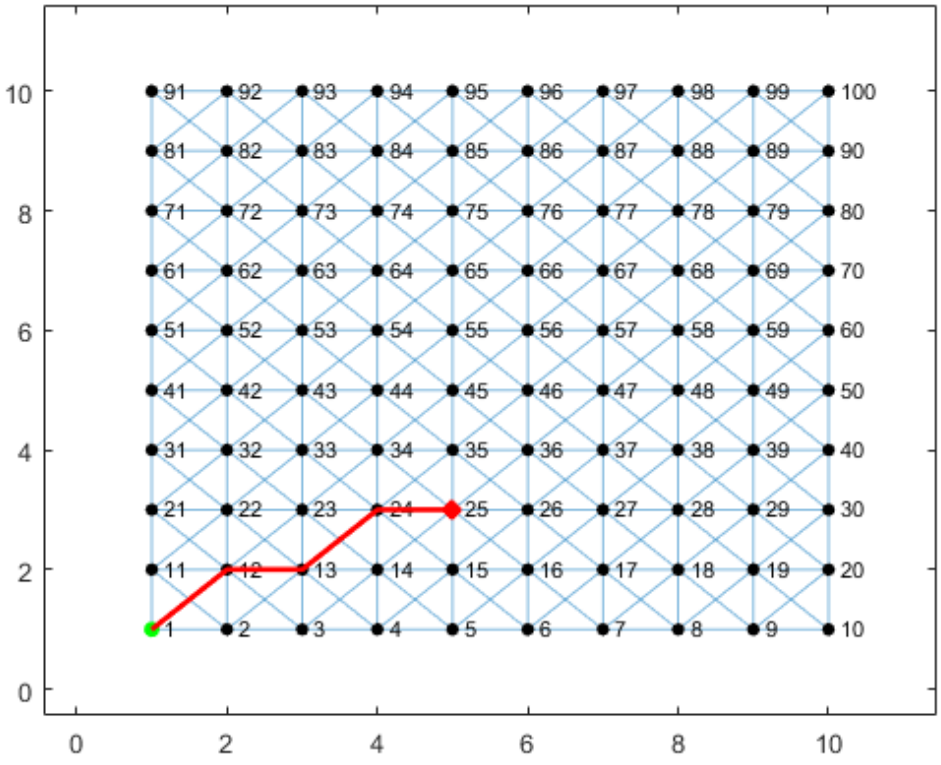


Figura 23: Ruta seleccionada al nodo 25.

Cuadro 20: Comparación de la ruta entre Matlab y C++

RUTA AL NODO 36		RUTA AL NODO 46	
Matlab	C++	Matlab	C++
1	1	1	1
12	12	12	12
23	13	23	23
34	24	34	34
25	25	45	35
36	36	46	46

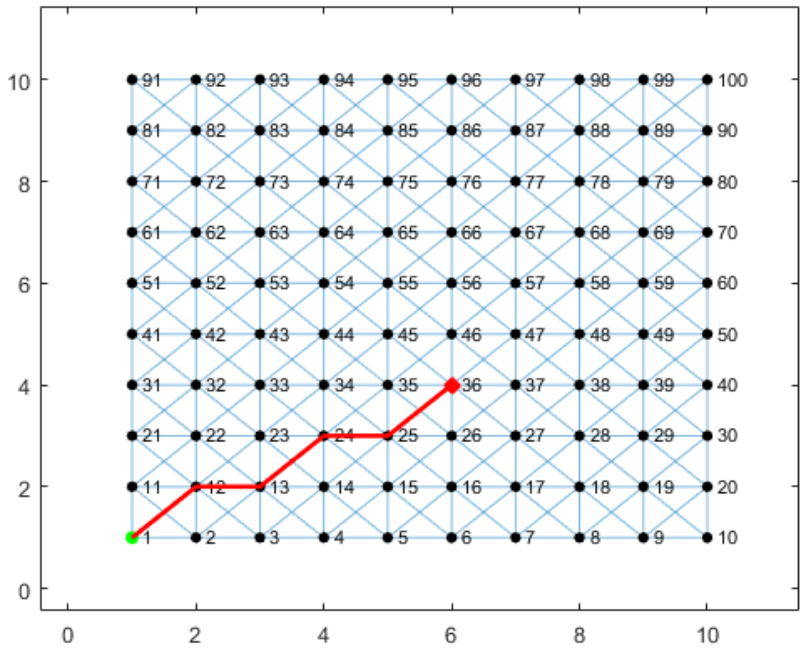


Figura 24: Ruta seleccionada al nodo 36.

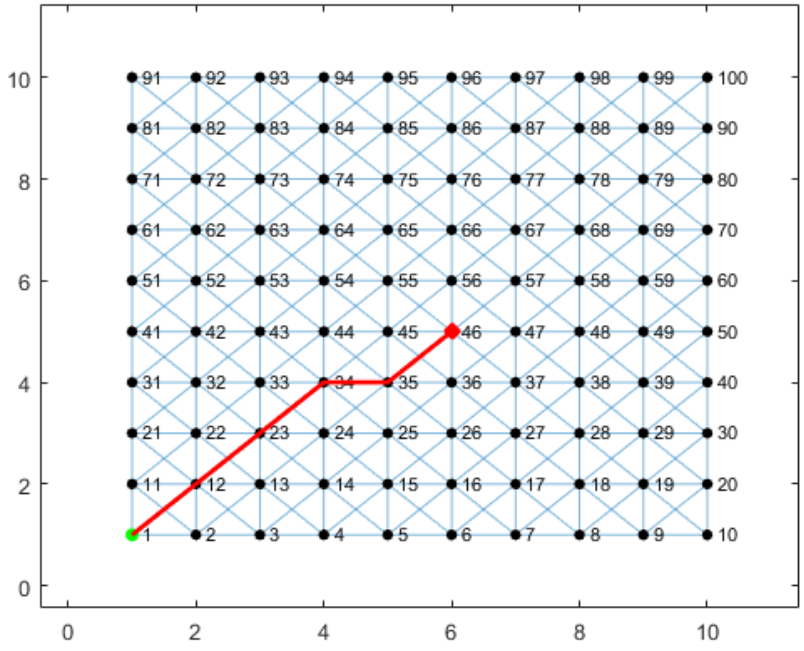


Figura 25: Ruta seleccionada al nodo 46.

Cuadro 21: Compraración de la ruta entre Matlab y C++

RUTA AL NODO 56	
Matlab	C++
1	1
12	12
23	23
34	34
45	45
56	56

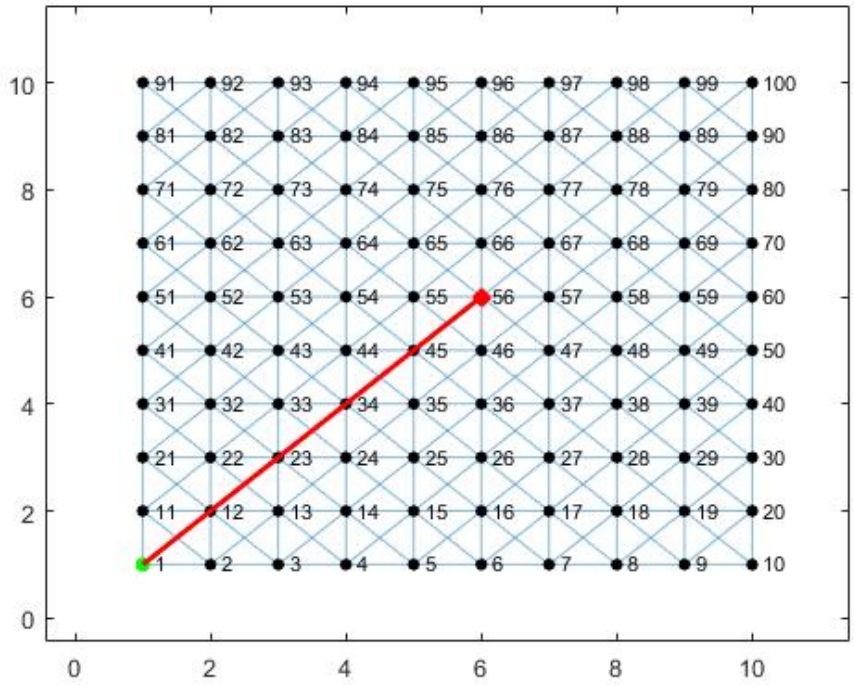


Figura 26: Ruta seleccionada al nodo 56.

8.3.2. Análisis de los resultados

Según los resultados generados se puede observar que el comportamiento de las rutas generadas en C++ es similar a las generadas por Matlab. Un cambio importante que se debe mencionar es que al generar la ruta en Matlab las hormigas tienden a tomar una ruta en diagonal por esta razón en algunas ocasiones no es posible encontrar una ruta ya que no encuentra convergencia en la cantidad de iteraciones definidas. Este comportamiento fue diferente en C++ donde las hormigas se comportan de mejor manera.

Aparte de las rutas ya calculadas se hicieron pruebas para ver si en C++ es posible calcular una ruta que en Matlab no fue posible. Los resultados de estas pruebas se muestran

en el Cuadro 22 y en las Figuras 27 y 28 donde se puede ver que sí fue posible calcular un camino. Además la ruta calculada presenta movimientos suaves desde el inicio hasta llegar a la meta.

Cuadro 22: Compraración de la ruta entre Matlab y C++

Rutas seleccionadas	
C++ Nodo 48	C++ Nodo 73
1	1
12	11
13	21
14	31
25	42
36	53
37	63
48	73

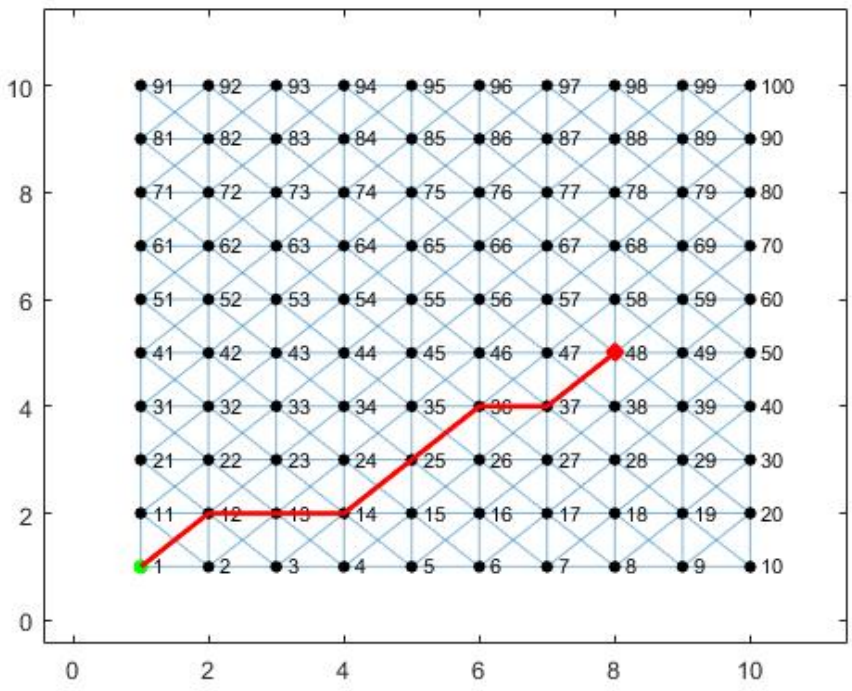


Figura 27: Ruta seleccionada al nodo 48.

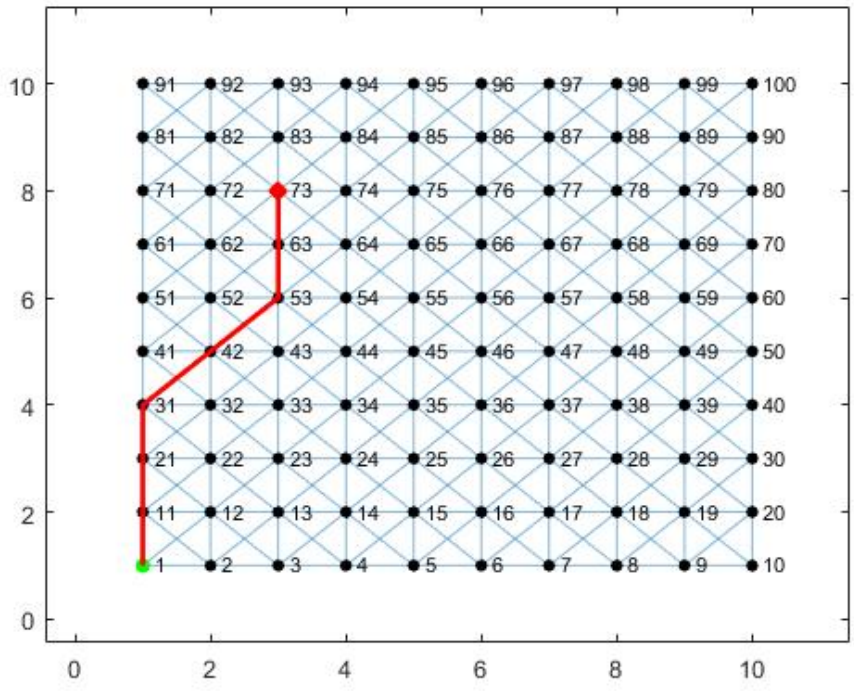


Figura 28: Ruta seleccionada al nodo 73.

CAPÍTULO 9

Ant Colony Optimization enfocado a robots móviles

En este capítulo se detallan las pruebas y validaciones realizadas para poder implementar el planificador de trayectorias en un robot móvil.

9.1. Simulaciones en software

Luego de realizar la validación del algoritmo y comprobar que funciona de manera correcta se procedió a realizar simulaciones más realistas aprovechando la plataforma ya desarrollada en fases anteriores donde se utilizó Webots R2020a. Para las simulaciones que se realizaron se utilizó el software Webots R2021a. Se realizaron varias pruebas con distintas metas, estas rutas planificadas en C++ fue extraída en un archivo txt y fue importada desde Matlab para convertir el archivo en una matriz numérica y que Webtos sea capaz de leer la información. Para esta simulación no consideraban las restricciones de los actuadores del robot. Además, tampoco se consideraba un modelo en específico de robot diferencial.

Al igual que en las fases anteriores se colocó una de 2x2 metros. Esta mesa de pruebas tiene su propio marco de referencia, que se tomó como el inercial. Como ya se mencionó anteriormente no se cuenta con una plataforma móvil disponible, por lo que se decidió trabajar con el robot E-puck y tomar las medidas de este robot para realizar todas las pruebas. Este último también tiene su propio marco de referencia, que no coincide con el inercial. Para una explicación más detallada de los marcos de referencia que se utilizan tanto en la mesa de pruebas como en el robot se recomienda leer el Capítulo 9 de [1].

9.1.1. Controladores de posición y velocidad

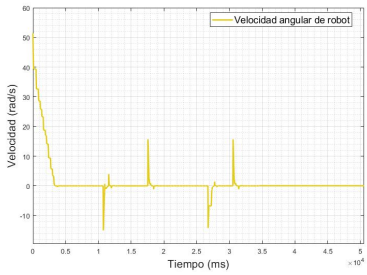
Se evaluaron los controladores descritos en la sección 6.9, los cuales en la fase pasada fueron los que presentaron mejores resultados a nivel de simulación.

Los controladores utilizados fueron:

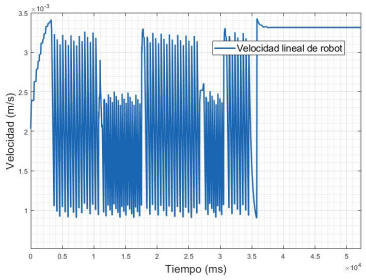
- Controlador de Pose
- Controlador de Pose de Lyapunov

Se decidió analizar el comportamiento de estos controladores observando las gráficas de la velocidad lineal, velocidad angular, velocidad de los motores y la trayectoria generada probando la ruta que se observa en la Figura 27 debido a que esta ruta presenta varios cambios de dirección y el objetivo era verificar el comportamiento en un ruta que se consideró complicada. Además se coloco el robot observando hacia la pared para tener un escenario más complicado. Los resultados obtenidos se pueden observar en las Figuras 29 y 30.

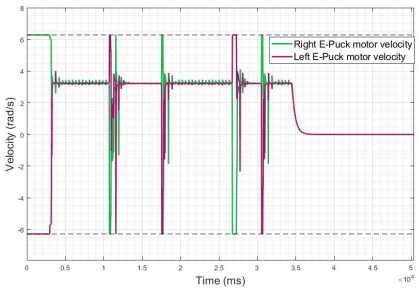
El comportamiento de ambos controladores es muy parecido. Podemos observar que ambos controladores al inicio presentan una alta velocidad angular (Figura 29a), esto es porque el robot debe girar ya que se encuentra viendo a la pared. Tanto los cambios bruscos en la velocidad angular como los picos en las velocidades de los motores (Figura 29c) coincide justo con los cambios de dirección observados en la Figura 29d.



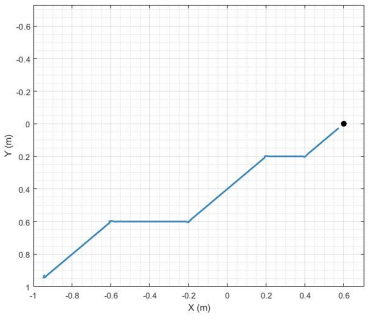
(a) Velocidad angular.



(b) Velocidad lineal.

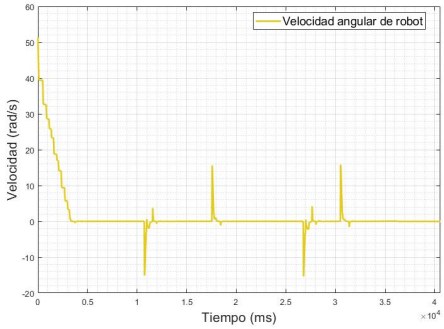


(c) Velocidad de los motores.

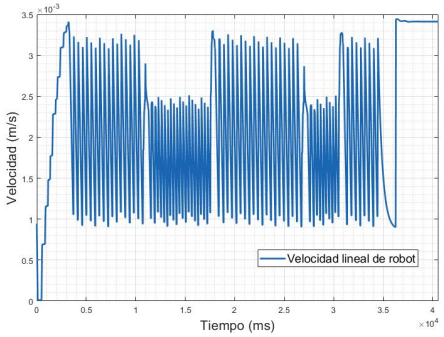


(d) Velocidad de los motores.

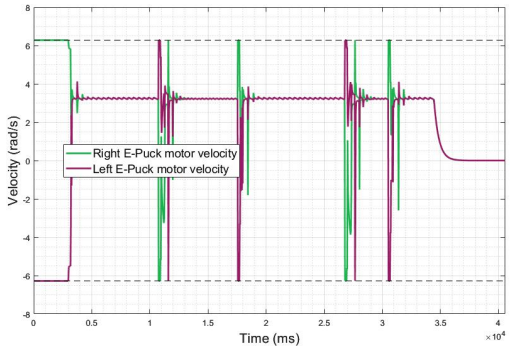
Figura 29: Gráficas del controlador de Pose.



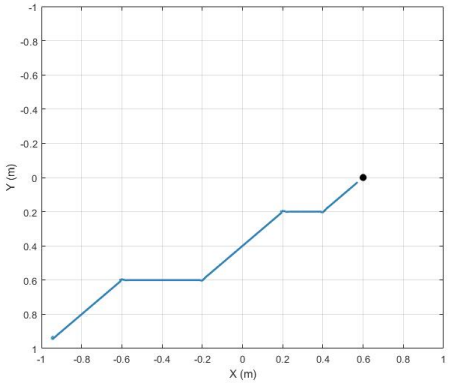
(a) Velocidad angular.



(b) Velocidad lineal.



(c) Velocidad de los motores.



(d) Velocidad de los motores.

Figura 30: Gráficas del controlador de Pose de Lyapunov.

9.1.2. Trayectorias obtenidas

Las simulaciones utilizando el software de Webots resultaron exitosas, con estas pruebas se valida el correcto funcionamiento del planificador de trayectoria descrito en el capítulo anterior. Las trayectorias generadas se pueden ver en las Figuras 31, 32, 33 y 35. Donde observamos que en 3 de los casos se pudo llegar exactamente al punto final (punto marcado con una estrella). En el caso de la ruta mostrada en la Figura 35, se ve que el robot logra seguir la ruta calculada, sin embargo no pudo llegar al final y se detiene antes de lo esperado, a pesar de esto se considera como una buena validación ya que se detiene cerca de la meta y la mayor parte del camino si logra seguir la ruta.

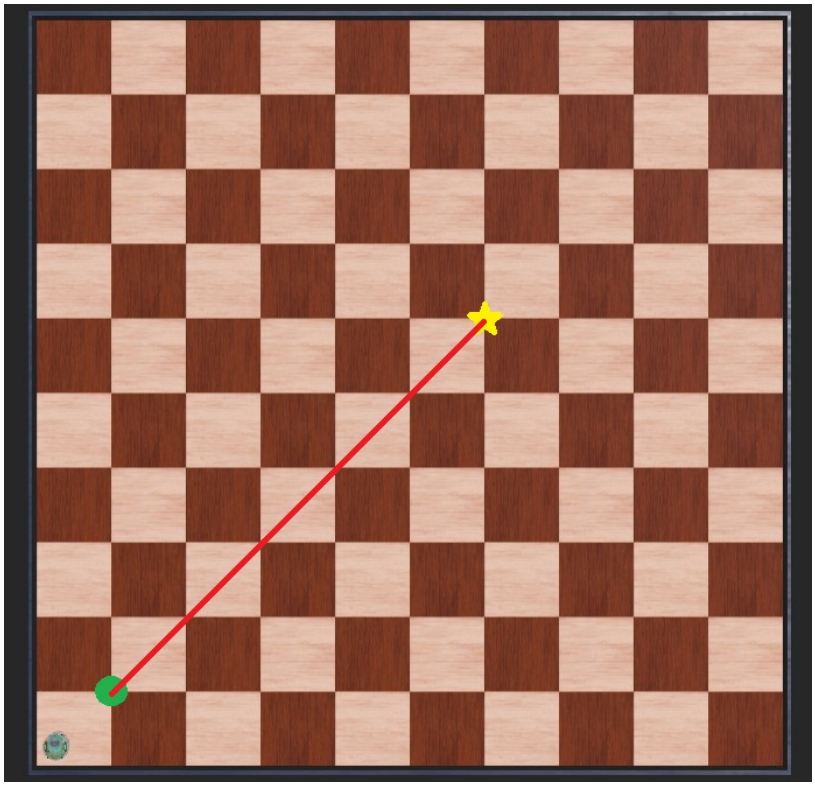


Figura 31: Simulación de la primera ruta en Webots.

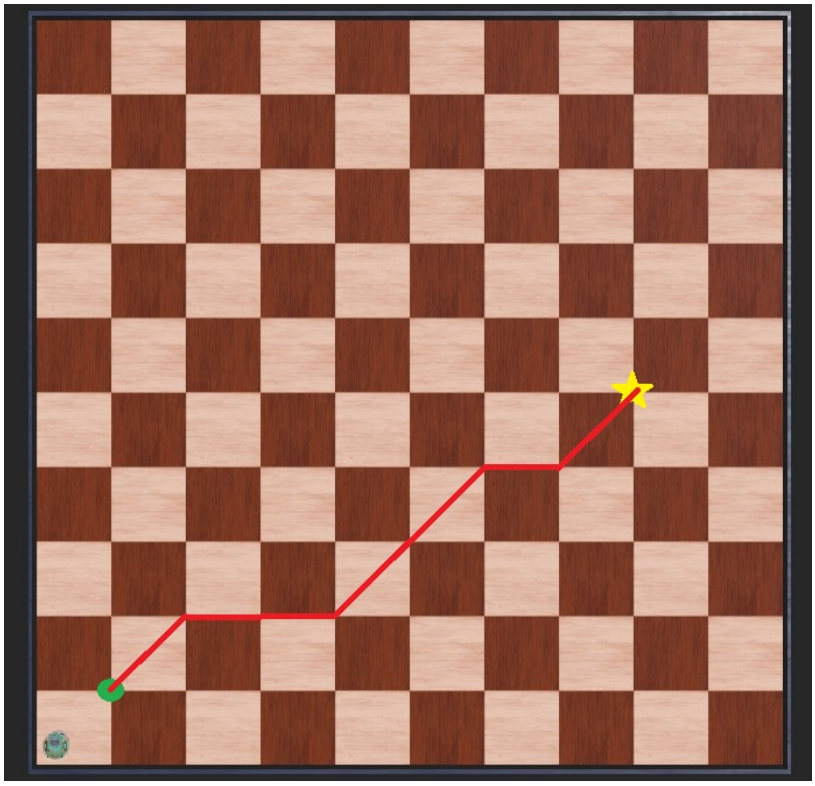


Figura 32: Simulación de la segunda ruta en Webots.

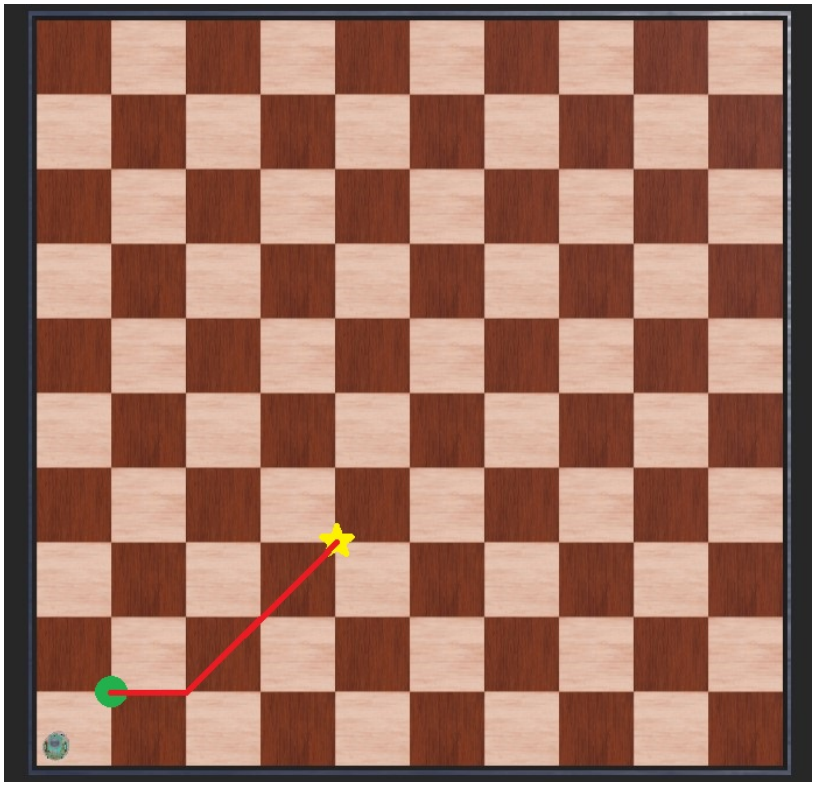


Figura 33: Simulación de la tercera ruta en Webots.

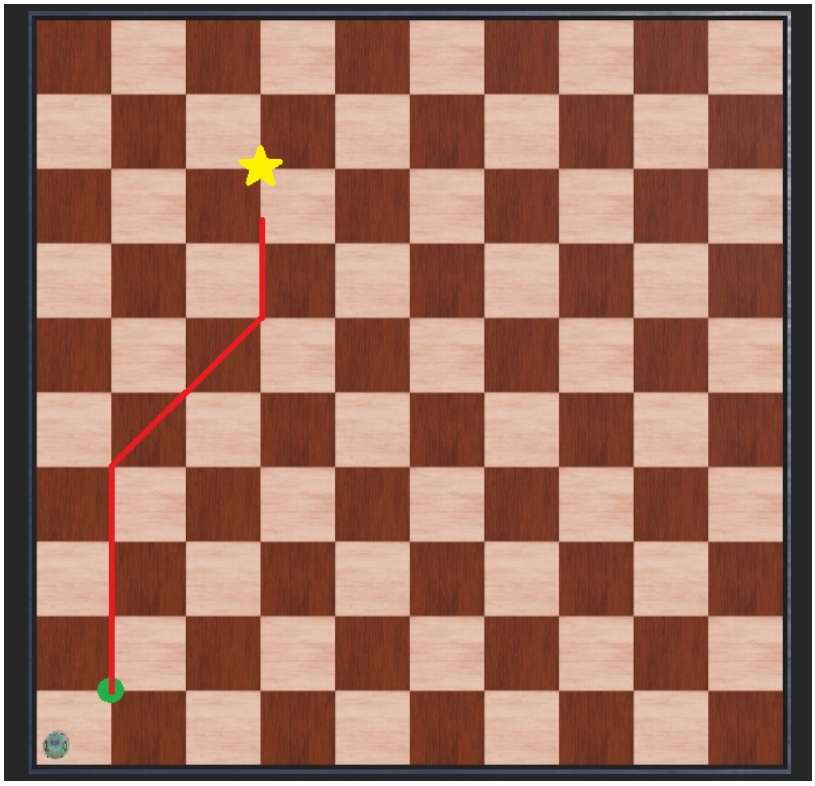


Figura 34: Simulación de la cuarta ruta en Webots.

9.2. Envío y recepción de datos

Para poder implementar el controlador de posición y velocidad y poder seguir la ruta calculada por el ACO, es necesario que el agente sea capaz de saber su posición y orientación para que este se pueda desplazar hasta llegar a la meta. Con el objetivo de poder recibir información constantemente se implementa el uso de la comunicación multihilos. Con esta implementación se puede estar recibiendo datos mediante algún protocolo de comunicación o algún módulo, mientras algún otro proceso esté corriendo de manera simultanea.

Se implementó el uso de una plataforma de rastreo con visión por computadora, dicha plataforma estará enviando la información a la o las RPi conectadas a la red. Esta comunicación se hace mediante un protocolo UDP cliente-servidor, para el cual todos los dispositivos que se comunicarán deben estar conectados a una misma red.

La plataforma que se utilizó para las pruebas está programada en una interfaz en Matlab, donde se define una transmisión a través de una IP especifica de la red. Estas serán las IP de todas las RPi que estén funcionando, para este caso únicamente funcionará una RPi en la cual se inicializa el hilo de recepción aparte, para esto se se procede a crear la función para recibir los datos de un *buffer*, por lo que se define una función *receiving* que servirá para obtener y separar la data del *buffer* proveniente.

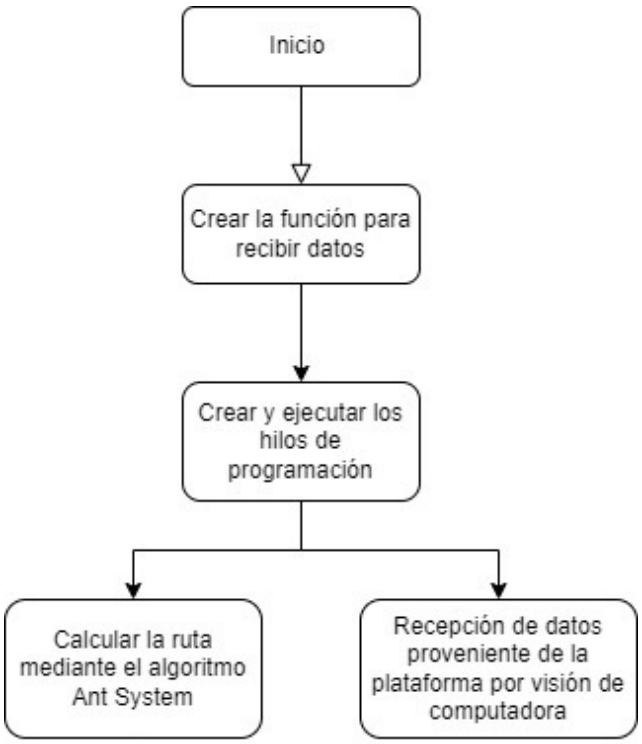


Figura 35: Proceso del algoritmo con la implementación de multihilos.

9.2.1. Comunicación entre agentes

A diferencia del MPSO, para este algoritmo no es necesario comunicar varios agentes para validar el funcionamiento, debido a que es un planificador de trayectorias que funciona individualmente en cada robot.

En alguna futura modificación a este algoritmo en la cual sea necesario comunicar varios agentes se realizaron algunas pruebas con comunicación entre RPi. Al igual que la comunicación con la plataforma de rastreo, esta implementación se hace mediante un protocolo UDP pero esta vez a través de broadcast, siempre conectado a una misma red. Esta comunicación entre agentes se realiza por broadcast y no por una IP especifica debido a que al transmitir vía broadcast, es posible conectar varios agentes y todos recibirán la información que se envíe. De esta forma no se tendrá que enviar a cada agente por separado sino que se envía a todos. Cada agente seleccionará la información que necesita.

Para verificar el funcionamiento de dicho proceso se realiza una prueba por medio de una transmisión a través de un broadcast, donde se envía y recibe un buffer con la información deseada. Para validar el proceso se envían desde dos RPi valores distintos de coordenadas, estos datos son recibidos por ambos agentes y separados. Cada agente recibe la información y es capaz de diferenciar de que RPi proviene la información. Las coordenadas recibidas son desplegadas en la consola diferenciando las coordenadas de cada uno. Esta prueba se observa en la Figura 36.

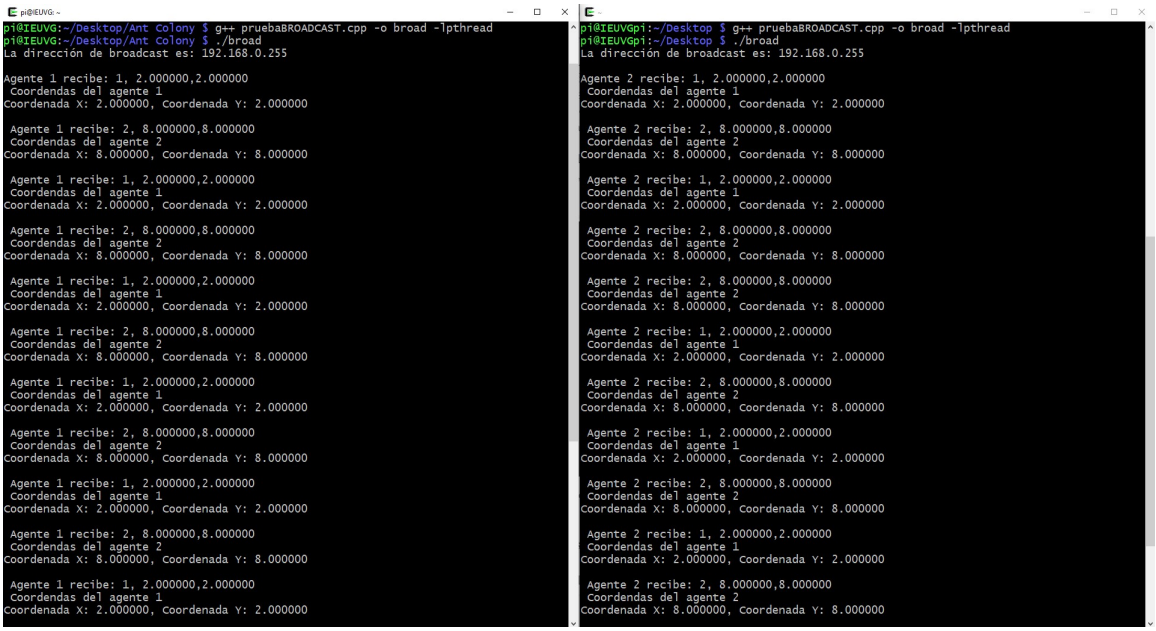


Figura 36: Envío y recepción de datos entre agentes a través de broadcast.

9.2.2. Recepción de coordenadas

Para poder recibir las coordenadas del robot en la mesa de pruebas se implementó la plataforma de rastreo por visión de computadora. También se hicieron algunas pruebas con

un módulo GPS para recibir las coordenadas.

Módulo GPS

Se realizaron algunas pruebas con el módulo Adafruit Ultimate GPS Breakout. El uso de este módulo no fue implementado en este trabajo ya que la precisión de este es poco certera con un cambio muy pequeño en la posición del robot. Este módulo podrá ser implementado en un proyecto de robótica de enjambre a gran escala donde el terreno a explorar sea grande.

Este módulo se basa en el chipset MTK3339, un módulo GPS de alta calidad con la capacidad de rastrear hasta 22 satélites en 66 canales, tiene un receptor de alta sensibilidad (seguimiento de -165 dBm) y una antena incorporada. Puede realizar hasta 10 actualizaciones de ubicación por segundo para registro o seguimiento de alta velocidad y alta sensibilidad. El uso de energía es bajo consumiendo únicamente 20 mA durante la navegación [36].

Para la recepción de los datos enviados por el módulo se ejecutan los comando directos en la consola linux de la RPi descritos en el artículo [37].

Para que este módulo funcione de manera correcta, es necesario hacer una configuración a la RPi. Se debe deshabilitar la consola serial y dejar únicamente los puertos seriales habilitados. Luego se debe instalar el software necesario. Por último se debe redirigir la data al puerto serial correcto, en este caso se utiliza el puerto serial 0 (ttyS0). Con dos instrucciones se puede desplegar la data recibida en la consola como se muestra en las Figuras 37 y 38 y con esto se valida la recepción de datos por medio del puerto serial 0 de la RPi utilizando este módulo GPS.

```
Time:       2021-09-30T23:51:45.000Z
Latitude:    14.64546999 N
Longitude:   90.46434166 W
Altitude:   1480.000 m
Speed:      0.68 kph
Heading:    14.4 deg (true)
Climb:      0.00 m/min
Status:     3D DIFF FIX (1243 secs)
Longitude Err:   +/- 11 m
Latitude Err:    +/- 3 m
Altitude Err:    +/- 11 m
Course Err:      n/a
Speed Err:       +/- 83 kph
Time offset:     0.751
Grid Square:     EK44sp
```

PRN:	Elev:	Azim:	SNR:	Used:
10	12	150	33	Y
16	58	183	25	Y
26	73	067	16	Y
27	12	174	39	Y
31	32	023	14	Y
1	18	257	16	N
3	34	319	15	N
4	07	312	00	N
21	17	238	18	N
22	53	301	13	N
32	28	085	17	N
138	64	230	28	N

Figura 37: Recepción de datos del módulo GPS.

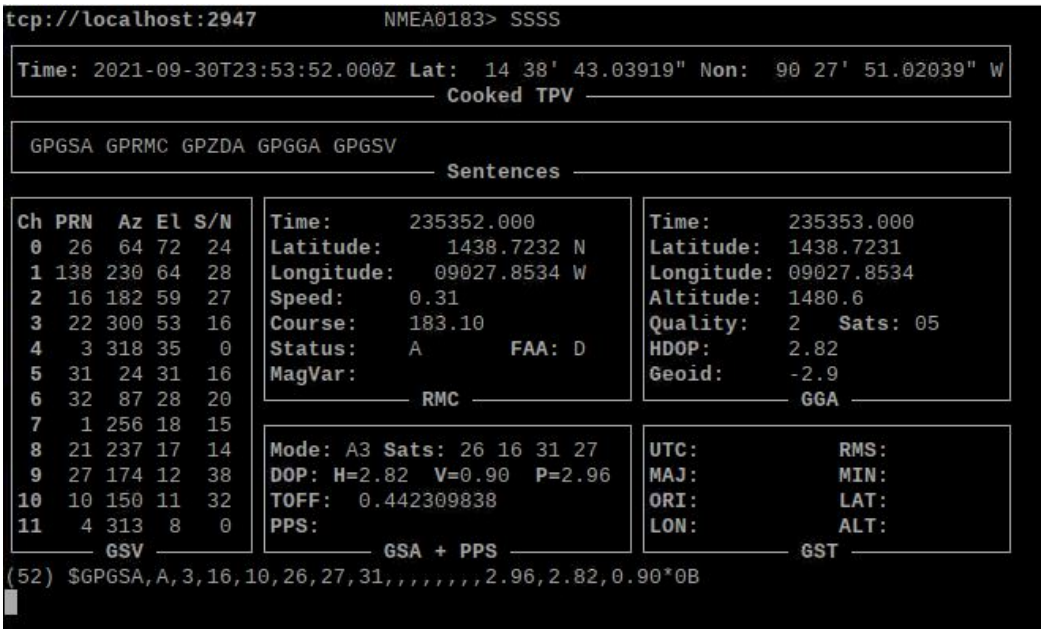


Figura 38: Recepción de datos del módulo GPS.

9.3. Plataforma de rastreo con visión por computadora

Esta plataforma está siendo desarrollada por José Ignacio Ramirez [38] como continuación al trabajo realizado por José Pablo Guerra en la tesis [39]. El algoritmo de visión por computadora empleado es capaz de reconocer la pose de unos marcadores, que representan los robots. Además, es capaz de reconocer otro tipo de marcadores (puntos), que representan los nodos del grafo con el que se trabajará. La plataforma cuenta con tres versiones en tres lenguajes de programación, Matlab, Python y C.

Como primera prueba se simularon los datos que se recibirán de la plataforma utilizando Matlab e implementando el protocolo de comunicación UDP cliente-servidor mencionado anteriormente. Se le enviaron distintos mensajes a la RPi para poder validar los distintos programas que se integraron.

9.3.1. Estructura de la data

Como ya se mencionó, se reciben dos tipos de información, información sobre la pose del robot o información sobre el grafo. Para poder identificar que se está recibiendo, se implementó un protocolo con indicadores que identifican el *buffer* recibido. Para la implementación de este protocolo se decidió concatenar un indicador al inicio de la cadena enviada. Cuando se envía información sobre la pose del robot se concatena la letra P al inicio de la cadena, para la información sobre conexión de nodos en el grafo se utilizó la letra G, por último para las coordenadas de los nodos se utilizó la letra C.

Cuando la RPi recibe información a través del *buffer*, lo primero que hace es comparar

el primer caracter de la cadena con las letras ya establecidas y determinar que tipo de información es. Una vez identificado qué se está recibiendo, se procede a separar la información y guardarla en las variables establecidas que serán utilizadas más adelante. El proceso de esta implementación se ilustra en el diagrama de flujo de la Figura 39.

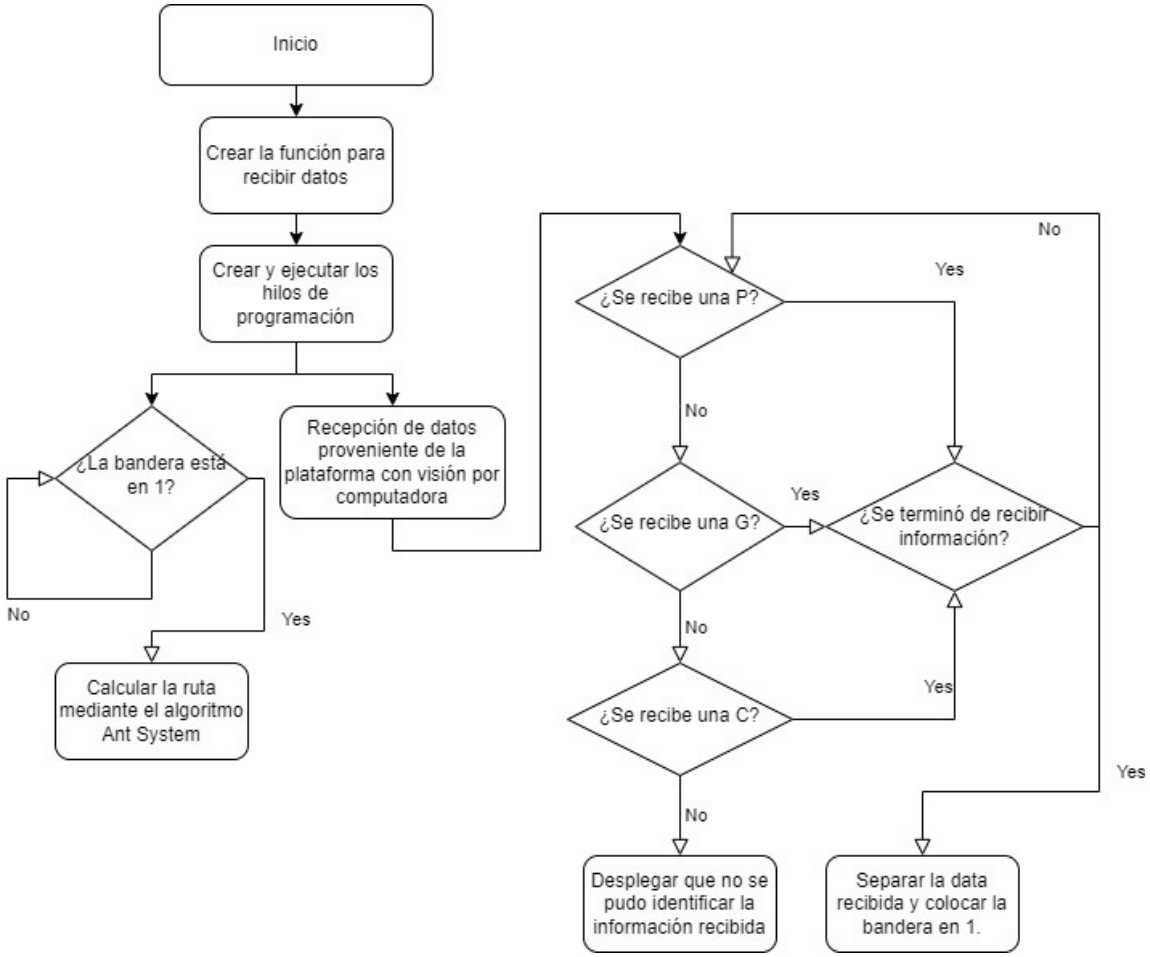


Figura 39: Protocolo de recepción de datos e hilos de de programación.

9.3.2. Validación de la recepción de datos

Para poder validar el correcto funcionamiento del proceso de comunicación es necesario realizar pruebas y validar tanto del protocolo UDP cliente-servidor como del protocolo que se implementó para estructurar la data. La primer prueba realizada consistió en implementar únicamente el protocolo de comunicación UDP a través de una IP específica. Se crearon distintas cadenas de caracteres las cuales fueron enviadas a la dirección IP de la RPi y se desplegó en la consola la información recibida. En la Figura 40 se puede ver que la RPi es capaz de recibir la información, con esta prueba se puede validar el funcionamiento del protocolo de comunicación UPD utilizado.

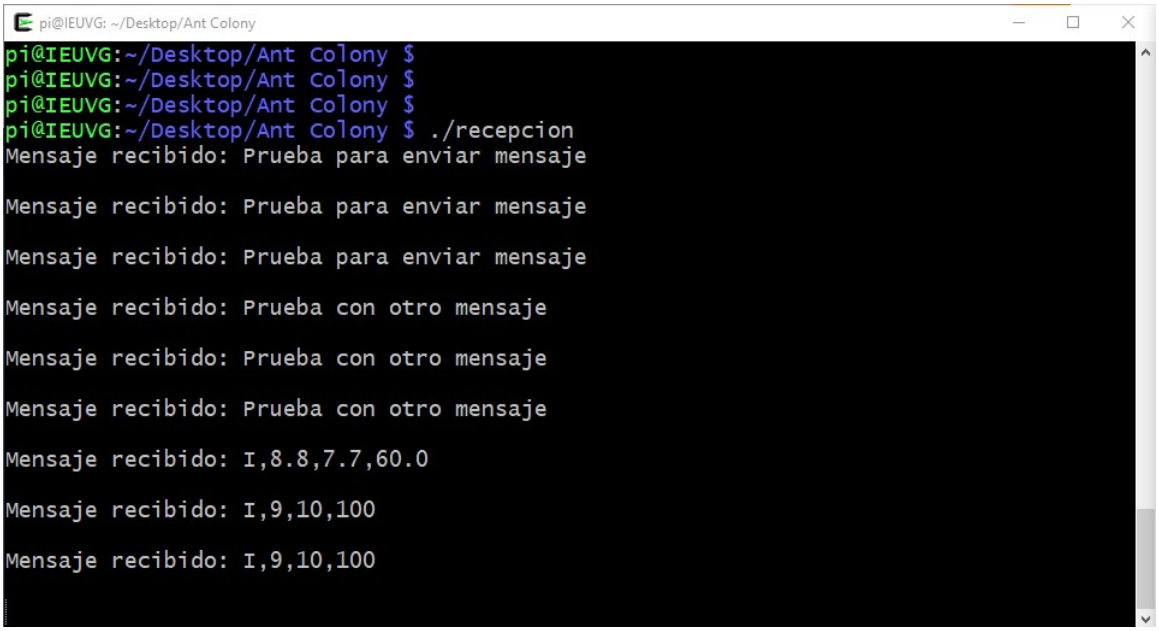


Figura 40: Prueba de la comunicación UDP con Matlab.

La segunda prueba se realizó con el objetivo de validar el protocolo para identificar la información recibida. Para comprobar que la RPi sea capaz de diferenciar la información recibida se procedió a enviar distintas cadenas de caracteres con los indicadores establecidos.

En la Figura 41 se puede observar como le llega la información y la RPi es capaz de reconocer el tipo de información y separarla. Se enviaron diferentes cadenas para poder probar los cuatro escenarios posibles. El primero contenía un identificador con la letra P, lo que significa que se recibirán coordenadas del robot, el mensaje que se desplegó indica que sí se logró identificar la cadena y separar la información. La segunda cadena se envió un indicador con la letra G, lo cual indica una conexión de nodos. Se puede observar en la consola los dos nodos que se conectarán. La tercer cadena recibida contenía el indicador con la letra C, que significa coordenadas del nodo, al igual que los escenarios anteriores, si fue posible identificar y separar la información. Por último, se envía un mensaje sin ningún identificador y como se puede ver, en la consola se despliega el mensaje donde indica que no se pudo identificar el mensaje.

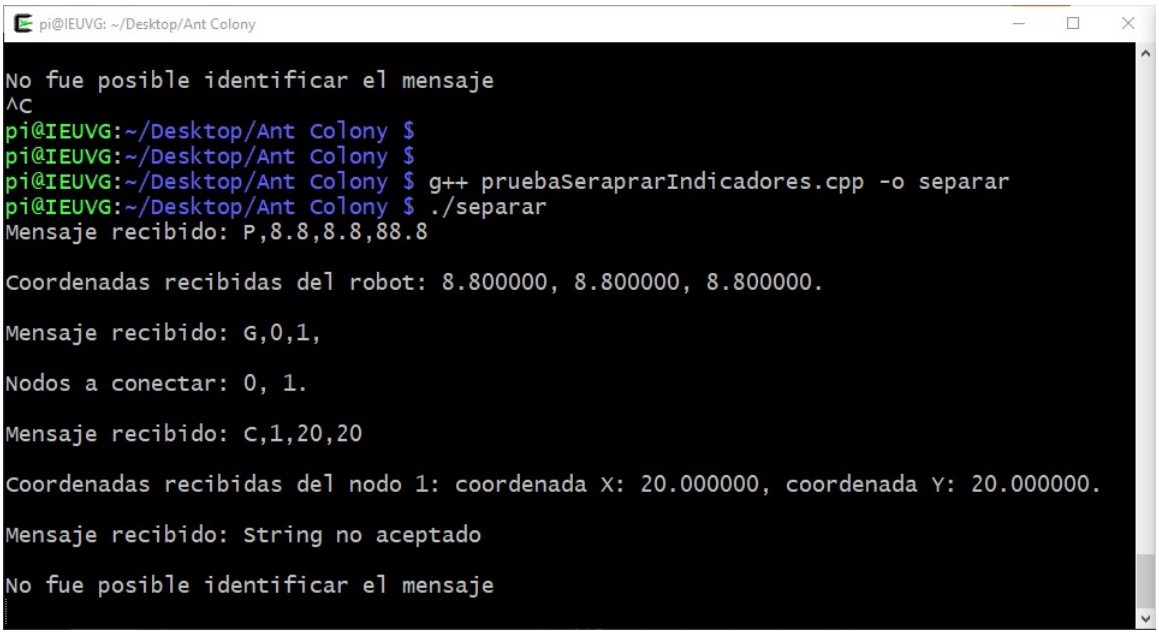


Figura 41: Prueba del protocolo de identificación del mensaje.

9.3.3. Implementación de multihilos

Luego de validar el funcionamiento de todos los procesos por separado se procedió a integrar todos los programas utilizando programación multihilos. Como ya fue explicado anteriormente, fue creado un hilo adicional donde se ejecutó el proceso de comunicación por UDP y separación del mensaje recibido. Asimismo se implementó el uso de una bandera al recibir información, de esta forma poder indicar en el hilo principal que se deben ejecutar las tareas, este proceso se ilustra en la Figura 39.

Para validar este nuevo proceso se realizaron pruebas enviando información desde Matlab y poder verificar que se este ejecutando el código correcto. Se enviaron varias cadenas con los indicadores establecidos para poder visualizar la ejecución de los programas correctos. Como primer prueba se enviaron tres conexiones y una coordenada del grafo, en la Figura 42 se puede ver que el programa es capaz de recibir el mensaje, identificarlo, separarlo y luego ejecutar las funciones del algoritmo AS correspondientes para definir el mapa.

Un requerimiento importante que se debe mencionar es que es necesario tener toda la información del mapa antes de empezar a recibir la pose del robot ya que se necesita tener la información completa del mapa para calcular la ruta, de lo contrario se calculará una ruta no óptima ya que el mapa estaría incompleto. En la Figura 43 se pude observar que se empieza a recibir la pose del robot, luego de esto se muestra en la consola el grafo recibido y por último el algoritmo comienza a calcular la mejor ruta. También se puede ver que las conexiones de los nodos mostradas en la Figura 42 fueron realizadas con éxito ya que se puede ver un número 1 en las conexiones recibidas (el nodo 0 con el nodo 1, el nodo 1 con el nodo 2 y el nodo 3 con el nodo 4).

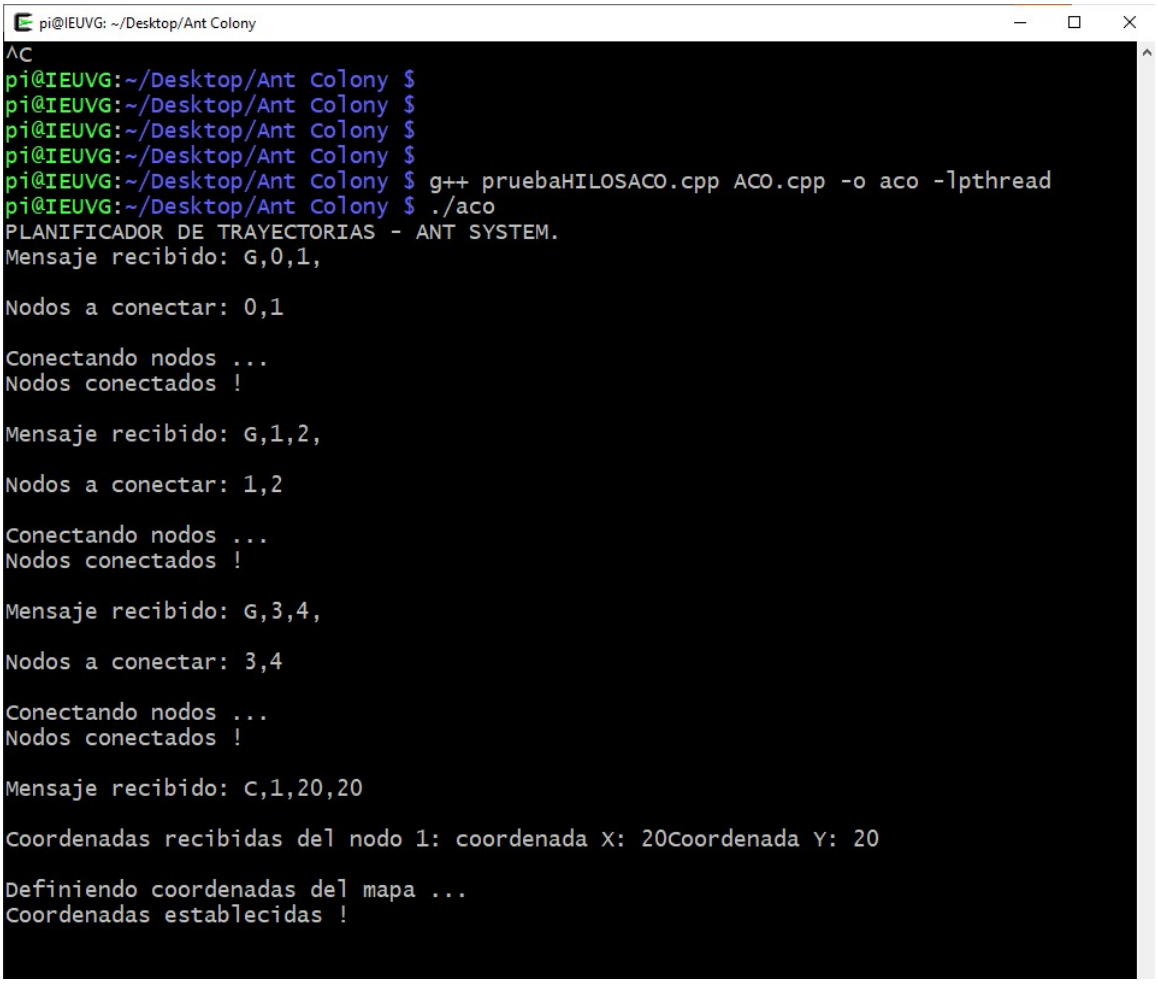


Figura 42: Recepción de datos por UDP.

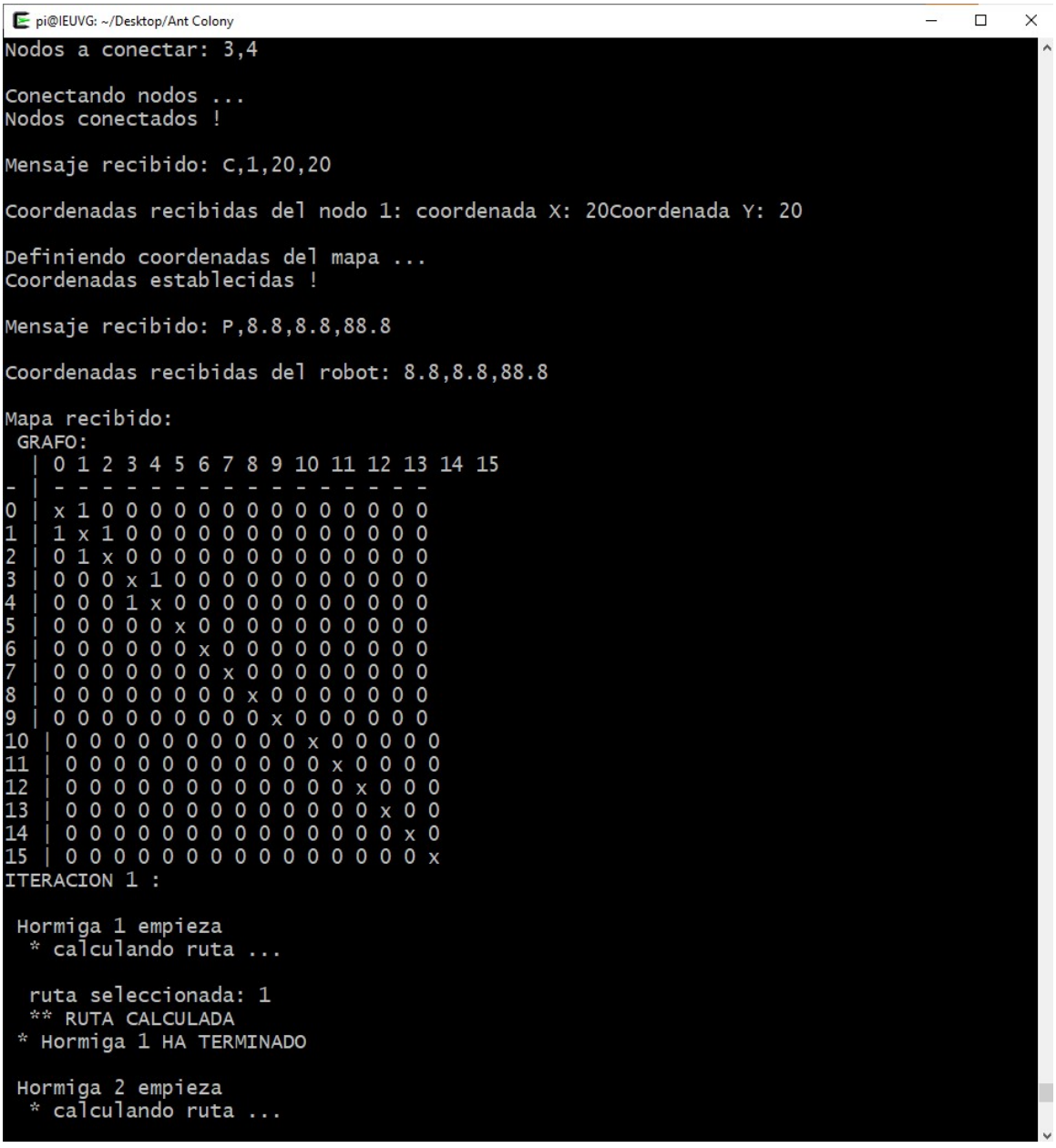


Figura 43: Grafo recibido por UDP.

CAPÍTULO 10

Validación utilizando la plataforma de rastreo con visión por computadora

En este capítulo se detallan las pruebas realizadas en la mesa de pruebas de la UVG utilizando la plataforma de rastreo con visión por computadora y poder validar los programas descritos en el capítulo anterior con los datos recibidos por la plataforma.

Para el envío de la pose del robot y la información del mapa en las pruebas realizadas se utilizó una interfaz desarrollada en Matlab, la cual tiene la función de cliente. Se define una IP especifica a la que se le enviará la información (la IP de la RPi) que hace la función de servidor en el protocolo. Para las pruebas realizadas con esta plataforma se construyó un grafo de 4x4 en la mesa de pruebas como se observa en la Figura 45 y se realizaron las pruebas de la planificación de trayectorias con el mapa recibido.

Se realizaron dos tipos de pruebas en dos escenarios distintos para la validación del algoritmo, la primera consistió en recibir el grafo de la plataforma con todas las conexiones entre los nodos. Para la segunda prueba se le agregaron obstáculos al mapa, lo que impedía la conexión entre ciertos nodos del grafo. Con el primer escenario se realizaron cuatro rutas distintas y con el segundo escenario se probaron tres casos con obstáculos en distintos puntos.

Como se puede observar en el diagrama de flujo de la Figura 46 fue necesaria la implementación de nuevas banderas con el objetivo de identificar el mensaje que se está recibiendo y poder indicar en el hilo principal que tipo de información se esta recibiendo y poder ejecutar las funciones correctas de la clase descrita en la sección 8.1.1. Una vez se recibe un mensaje, este es separado, la información es almacenada un variables globales y se activa la bandera, inmediatamente después, en el hilo principal se utiliza la información de las variables como argumentos de entrada a las funciones especificas y por último una vez todo ha sido ejecutado se desactiva la bandera.

Al momento de tener toda la información del mapa, se procede a ejecutar el planificador de trayectorias utilizando el algoritmo *Ant System*. Para las primeras pruebas se utilizaron

los parámetros descritos en la sección 8.3.1, sin embargo se decidió aumentar el número de iteraciones y hormigas para obtener mejores resultados, se utilizaron los parámetros mostrados en el Cuadro 23. También se implementaron los controladores de posición y velocidad diseñados por Aldo Aguilar en [7].

Cuadro 23: Parámetros óptimos del algoritmo

Parámetros	Valor
Iteraciones	100
Hormigas	90

Figura 44: Setup de la mesa de pruebas UVG.

Figura 45: Grafo construido en la mesa de pruebas.

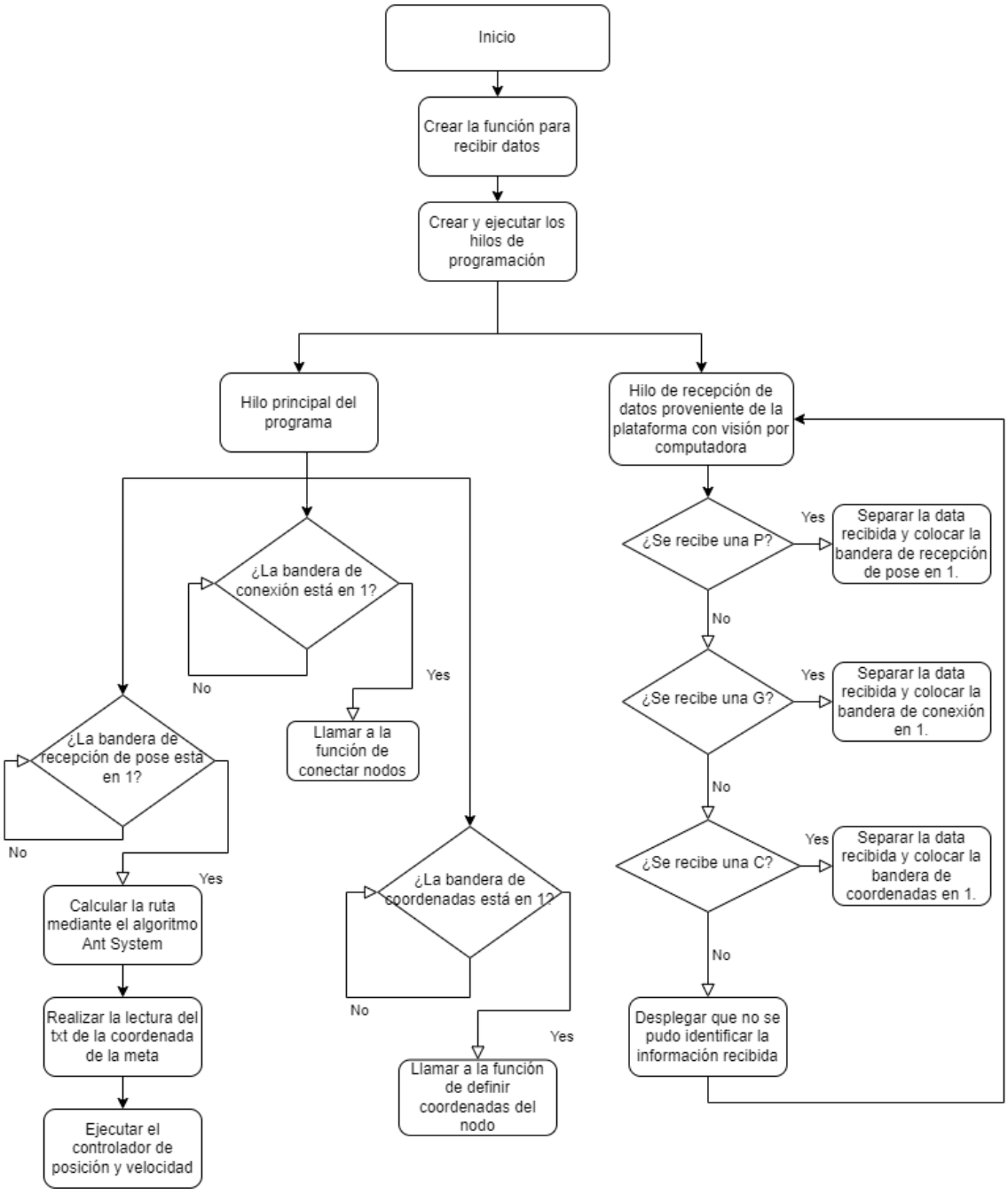


Figura 46: Diagrama de flujo del proceso del algoritmo.

10.1. Recepción de datos de la plataforma

Una vez se tiene el mapa en la mesa de pruebas se hace una fotografía y se procede a identificar los marcadores (puntos) que formarán el grafo. Se utiliza un filtro Canny para detectar bordes y se rellenar dichos bordes. El resultado después de aplicar el filtro se puede ver en las Figuras 47 y 48. Luego se identifican los círculos encontrados en la imagen para poder encontrar los centroides de cada uno. En el caso que se tengan obstáculos en el mapa, es necesario tomar una fotografía de los obstáculos para saber la posición como se observa en la Figura 48. Finalmente cuando ya se este detectando el marcador del robot, estas mascaras son eliminadas para detectar únicamente el robot.

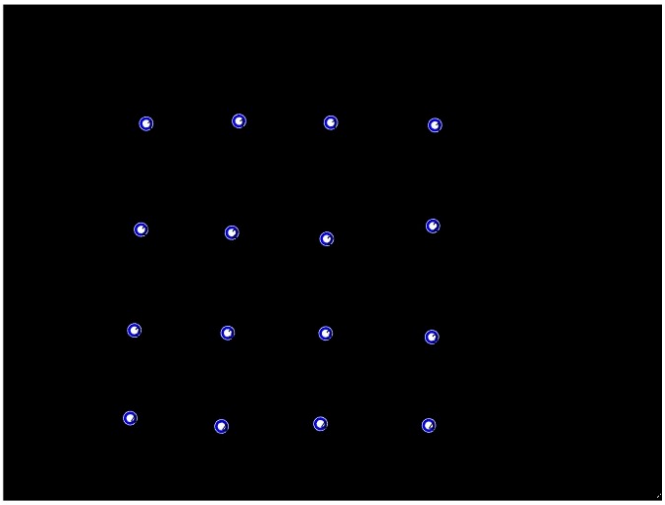


Figura 47: Marcadores detectados por la plataforma de rastreo.

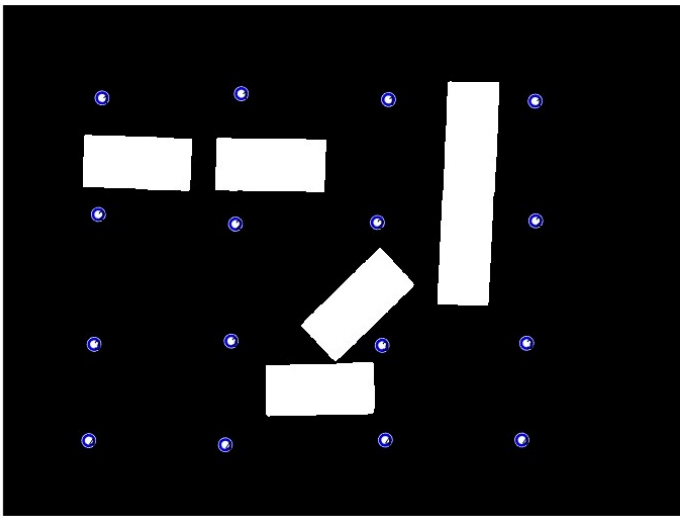


Figura 48: Marcadores y obstáculos detectados por la plataforma de rastreo.

10.1.1. Pruebas realizadas

Para poder validar el funcionamiento del algoritmo en la mesa de pruebas se utilizó un marcador que simula ser el robot debido a que no se cuenta con una plataforma móvil, el marcador utilizado se puede observar en la Figura 49. Este marcador se fue moviendo manualmente simulando el movimiento que tendría el robot. En las Figuras 50, 51 y 52 se puede observar los mensajes recibidos por la plataforma y la ejecución de las funciones de la clase de la primer prueba realizada.

Para esta prueba se probó el grafo con todas las conexiones. Como se puede observar en la Figura 50, se reciben las coordenadas de todos los nodos, esta información sobre las coordenadas están dadas en centímetros. Desde la plataforma en Matlab, luego de encontrar el centroide de cada marcador se guardan y se envían todas juntas. Una de las limitaciones que se tiene es la velocidad de procesamiento de la RPi, al recibir varios mensajes de manera consecutiva es necesario tener un *delay* para poder ejecutar las funciones y recibir el siguiente mensaje sin tener pérdidas en la información que se esta recibiendo. Este *delay* fue de 1 milisegundo para asegurar un tiempo bastante grande y que la RPi pueda procesar la información sin ningún problema.

De igual forma se implemento el mismo *delay* al recibir las conexiones de los nodos, las cuales se pueden ver en la Figura51. Una vez se tiene toda la información del mapa se coloca el marcador del robot en la mesa de pruebas y se comienza a enviar la pose del marcador para que se calcule la mejor ruta y se comience a ejecutar los controladores. En la Figura 52 se puede observar la ruta calculada por el algoritmo y dos recepciones de coordenadas del robot, para una mejor visualización de esta ruta se puede observar la Figura 53. También se puede observar el funcionamiento del controlador implementado indicando la velocidad de cada motor.

Una limitante a la hora de la validación de los controladores y por ende resultaron ser complicados de evaluar fue el no disponer de una plataforma móvil con motores. Esto causó que el movimiento del marcador no fuera continuo sino que se movía manualmente luego de cada fotografía. El envío de la pose era aproximadamente cada 40 segundos ya que se tiene que mover el marcador y tomar una nueva fotografía, este proceso depende mucho de la capacidad de la computadora que se esté utilizando para procesar la imagen de la mesa de pruebas.

Aparte de esto la velocidad de los motores puede variar en fracciones de segundo para ajustar la trayectoria que se está siguiendo por lo que se puede tener velocidades negativas o positivas que no necesariamente impliquen el movimiento hacia adelante o hacia atrás sino que pueden ser velocidades únicamente para ajustar la orientación del robot. Dadas estas limitantes resulta imposible de validar el funcionamiento de los controladores al ver únicamente las velocidades cada cierto tiempo.

Para recibir la pose del robot se fue moviendo manualmente el marcador, simulando ser la plataforma móvil desplazándose por la mesa de pruebas. Al inicio de la trayectoria se le asigna al robot las coordenadas del primer nodo como meta. El marcador se mueve cierta distancia y luego se vuelve a tomar la pose, a medida que el robot era colocado cerca de la meta, el error de posición empieza a disminuir. A nivel de simulación el error de posición utilizado fue de 0.05 cm para poder decir que el robot llegó a la posición deseada y el

desempeño obtenido fue muy bueno. Sin embargo al hacer las pruebas con los marcadores, se pudo ver que este error era muy pequeño y era casi imposible colocar el marcador justo en la posición de la meta por lo que se decidió realizar varias pruebas para encontrar un valor adecuado con el que se pudiera colocar el marcador en una posición bastante cercana a la meta y poder decir que se llegó a las coordenadas deseadas. Para las pruebas realizadas se utilizó un error de posición de 1 cm. El proceso de esta prueba se puede observar en las Figuras 54, 55, 56 y 57, donde se muestra la posición inicial, dos puntos medios y cuando se llega a la meta.

En la Figura 54 como se observa en el el rectángulo rojo se ve la ruta calculada, las coordenadas del robot, las coordenadas del nodo al que tiene que llegar (nodo marcado con un círculo azul) y la velocidad de los motores. En la Figura 55 se observa que el robot ya se encuentra en el primer nodo, en el rectángulo rojo se observa que al llegar a las coordenadas del primer nodo, se actualiza la siguiente meta, que son las coordenadas del nodo marcado en azul. En la Figura 56 únicamente es un punto medio donde se ve que el robot sigue avanzando. Por último en la Figura 57 se puede observar que el robot llegó a la meta (círculo azul) tal y como lo indica el mensaje mostrado en la consola el cual se encuentra en el rectángulo rojo, también se detiene el movimiento de los motores y finaliza el proceso.

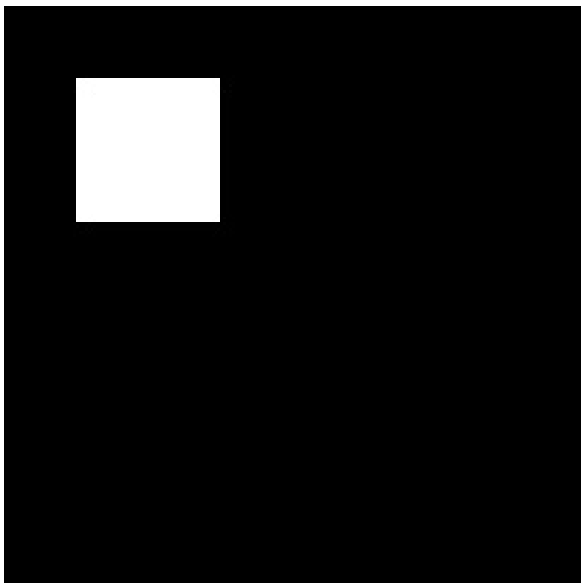


Figura 49: Marcador utilizado.

```
pi@IEUVG: ~/Desktop/Ant Colony
pi@IEUVG:~/Desktop/Ant Colony $ g++ ACO_hilos.cpp ACO.cpp  -o aco -lpthread
pi@IEUVG:~/Desktop/Ant Colony $ ./aco
PLANIFICADOR DE TRAYECTORIAS - ANT SYSTEM.
Mensaje recibido: C,0,23.2,19.4
Coordenadas recibidas del nodo 0: coordenada X: 23.2Coordenada Y: 19.4

Definiendo coordenadas del mapa ...
0 23.2 19.4
Coordenadas establecidas !

Mensaje recibido: C,1,22.3,36.6
Coordenadas recibidas del nodo 1: coordenada X: 22.3Coordenada Y: 36.6

Definiendo coordenadas del mapa ...
1 22.3 36.6
Coordenadas establecidas !

Mensaje recibido: C,2,21.3,53
Coordenadas recibidas del nodo 2: coordenada X: 21.3Coordenada Y: 53

Definiendo coordenadas del mapa ...
2 21.3 53
Coordenadas establecidas !

Mensaje recibido: C,3,20.6,67.3
Coordenadas recibidas del nodo 3: coordenada X: 20.6Coordenada Y: 67.3

Definiendo coordenadas del mapa ...
3 20.6 67.3
Coordenadas establecidas !
```

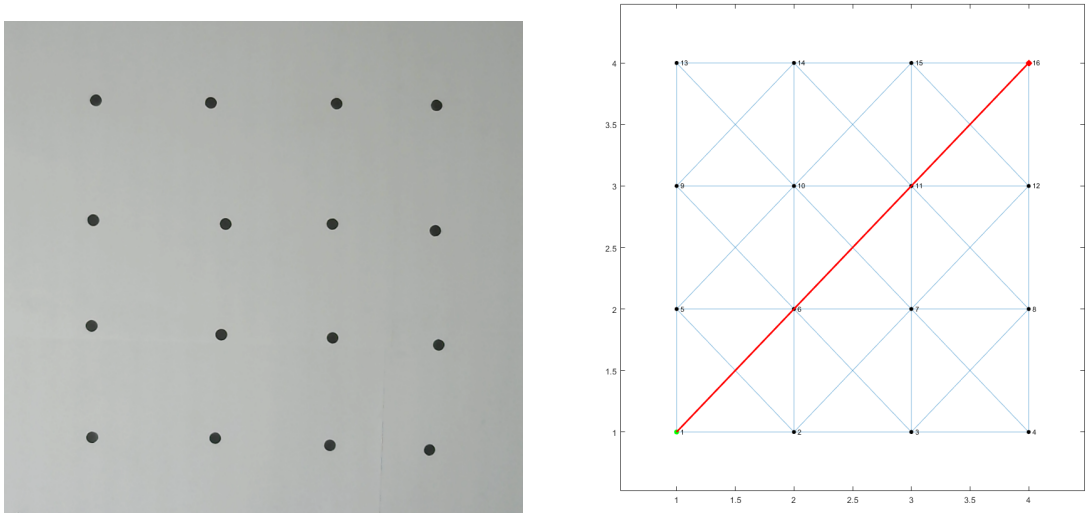
Figura 50: Recepción de coordenadas de los nodos.

```
Mensaje recibido: G,0,1,
Nodos a conectar: 0,1

Conectando nodos ...
Nodos conectados !

Mensaje recibido: G,0,4,
Nodos a conectar: 0,4

Conectando nodos ...
Nodos conectados !

Mensaje recibido: G,0,5,
Nodos a conectar: 0,5

Conectando nodos ...
Nodos conectados !

Mensaje recibido: G,1,2,
Nodos a conectar: 1,2

Conectando nodos ...
Nodos conectados !
```

Figura 51: Recepción de las conexiones de los nodos.

```
EL ALGORITMO ACO HA CALCULADO LA MEJOR RUTA!

 MEJOR RUTA:
1 6 11 16
Longitud: 67.1238

Posicion Actual:
4.9, 6.3
Posicion Siguiente:
23.2, 19.5
Velocidad del motor izquierdo: 2
Velocidad del motor derecho: 2

Mensaje recibido: P,10,8.2,320.4
Coordenadas recibidas del robot: 10,8.2,320.4

Posicion Actual:
10, 8.2
Posicion Siguiente:
23.2, 19.5
Velocidad del motor izquierdo: 2
Velocidad del motor derecho: 2
```

Figura 52: Recepción de pose del robot.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

Figura 53: Primera prueba sin obstáculos.

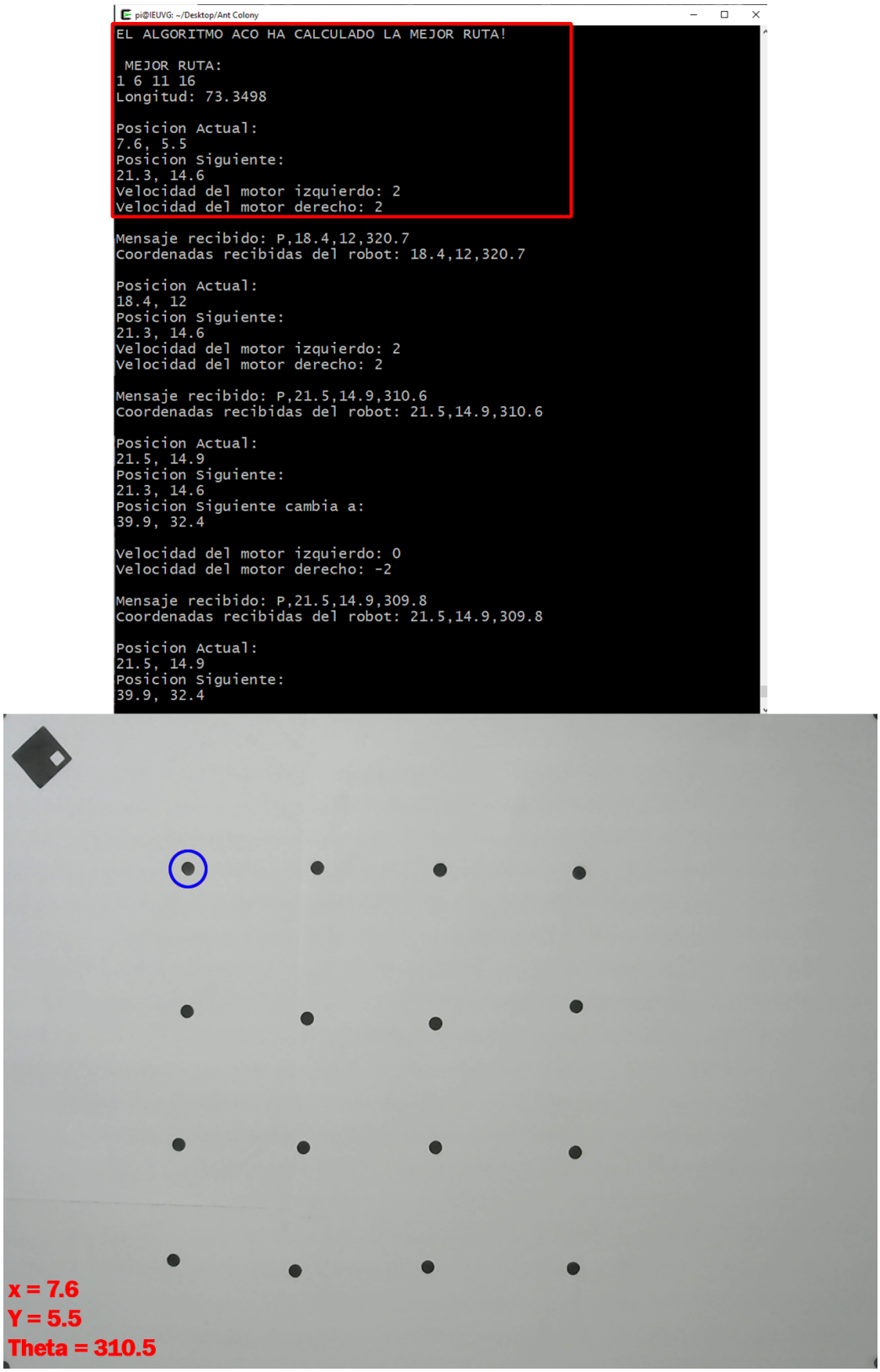


Figura 54: Proceso de validación. Punto inicial.

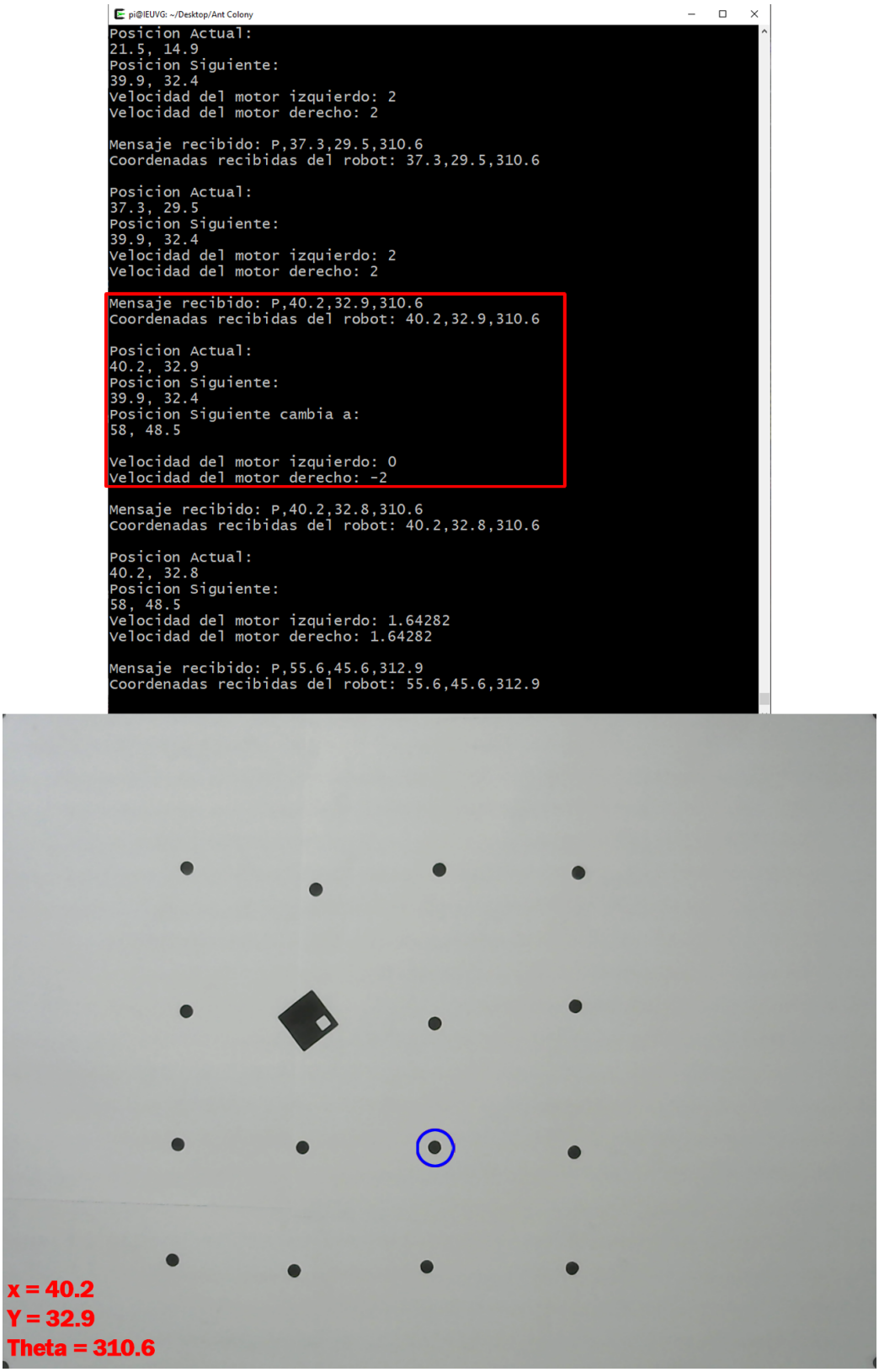


Figura 55: Proceso de validación. Punto medio.

Figura 56: Proceso de validación. Punto medio.

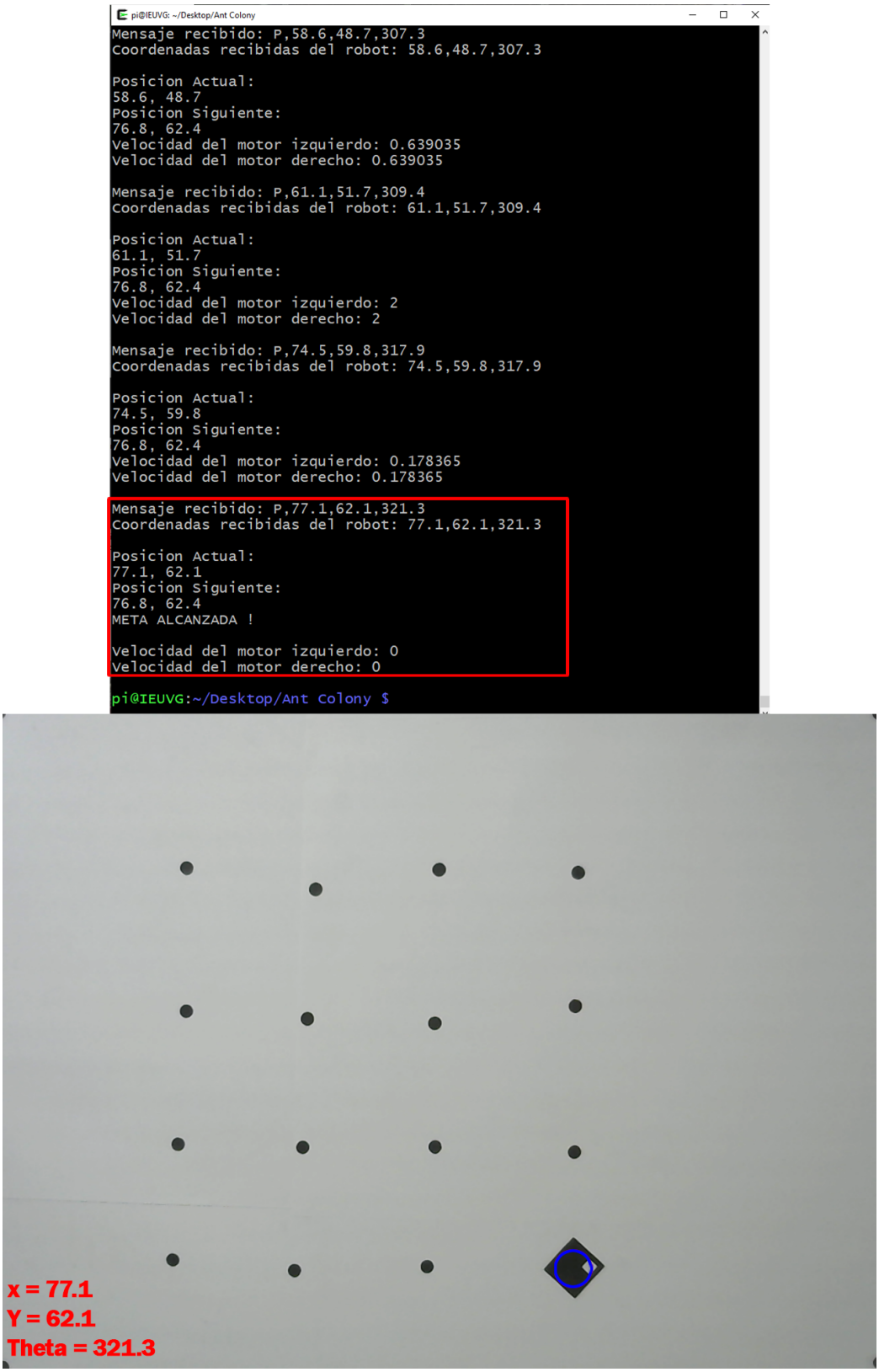
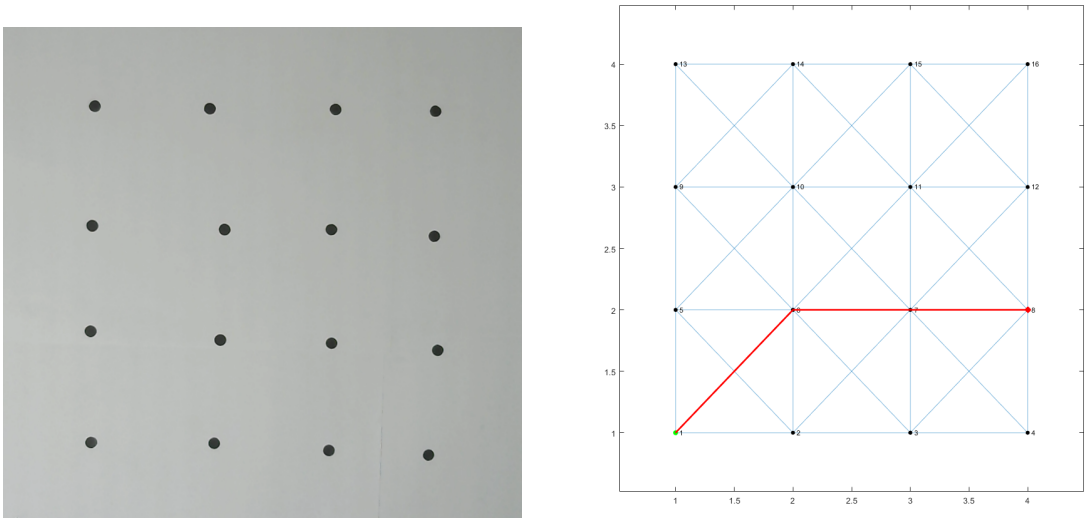


Figura 57: Proceso de validación. Punto final.

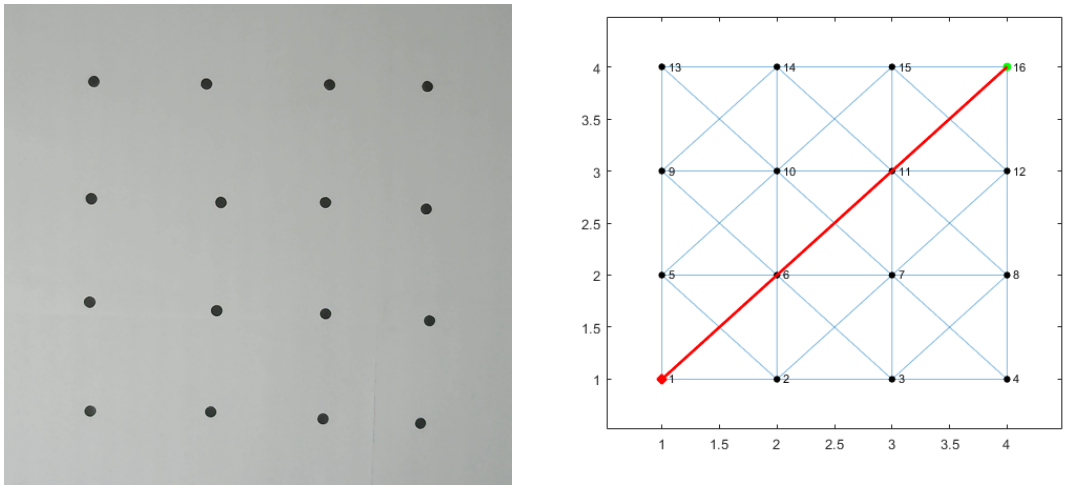
Para las siguientes pruebas se siguió el mismo procedimiento como se explicó para la primera prueba, los resultados de las trayectorias seleccionadas se muestran a continuación.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

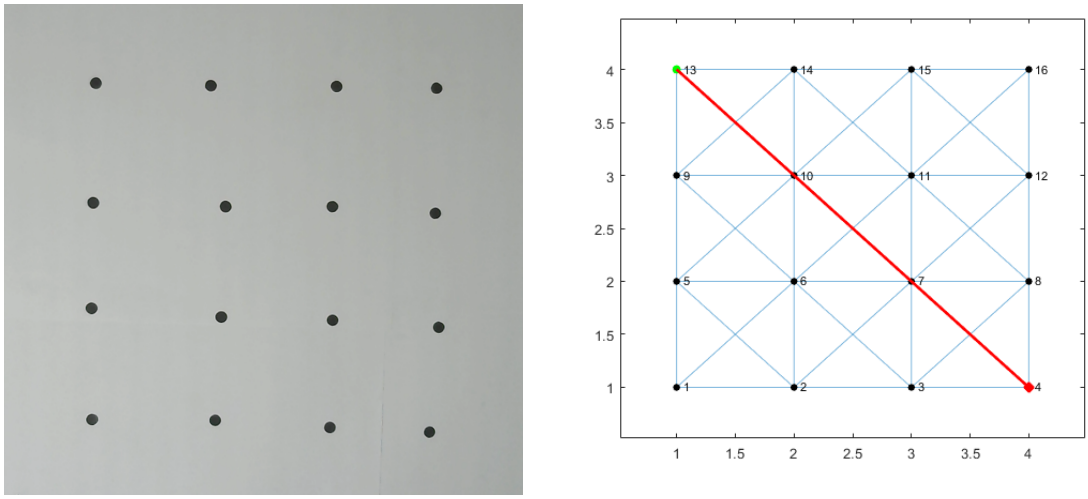
Figura 58: Segunda prueba sin obstáculos.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

Figura 59: Tercera prueba sin obstáculos.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

Figura 60: Cuarta prueba sin obstáculos.

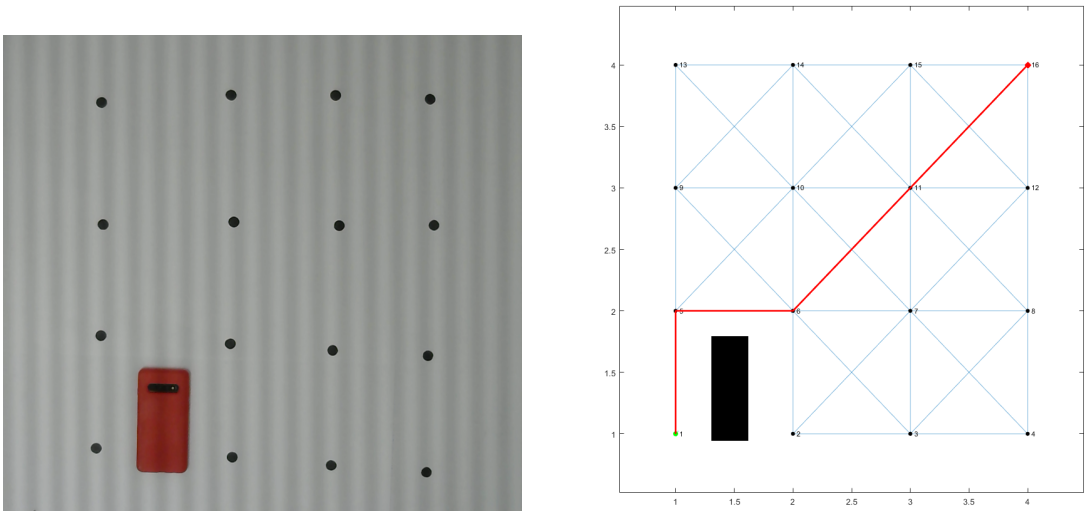
Luego de validar el funcionamiento del algoritmo sin ningún obstáculo en el mapa se procedió a colocar uno y varios obstáculos con el objetivo de complicar el cálculo de la ruta y simular una situación más realista ya que el robot no podrá pasar por cualquier lugar debido a que los nodos no estarán conectados.

En la Figura 61 se puede observar los resultados de la primer prueba que se realizo con obstáculos. Se puede ver que entre los nodos 1, 2, 5 y 6 no existe una conexión, lo que causará que el algoritmo deba buscar otra ruta a pesar que la óptima sería ir del nodo 1 al nodo 6. En las Figuras 65 y 67 se puede observar las otras dos pruebas realizadas donde fueron colocados más obstáculos. Tanto en la Figura 65 como en 67 se puede observar tanto la trayectoria seleccionada como el grafo en la mesa de pruebas, a pesar que los obstáculos en la fotografía (a) y en la (b) no es igual, para el algoritmo significa lo mismo ya que lo que se busca demostrar es que no existe una conexión entre los nodos.

Para la segunda prueba, los obstáculos fueron colocados para forzar al algoritmo a seguir una única ruta, como se puede ver en la Figura 65, el algoritmo sí fue capaz de encontrar este camino y pudo llegar a la meta. Para la tercer prueba se colocaron los obstáculos a manera que el robot tuviera varios caminos posibles. Como se puede observar en la Figura 67, el algoritmo eligió el camino más corto esquivando exitosamente los obstáculos colocados. Con estas pruebas se logra validar el funcionamiento del algoritmo AS en ambientes controlados utilizando un marcador (robot) en la mesa de pruebas e implementando la plataforma de rastreo con visión por computadora.

De la misma manera que se presentaron los resultados para la prueba anterior, en las Figura 62, 63 y 64 se muestra una secuencia de fotografías que muestran el proceso de validación de la ruta mostrada en la Figura 61. Para los otros dos escenarios se siguió el mismo procedimiento y los resultados se muestran en las Figuras 66 y 68 donde el nodo inicial esta marcado con un círculo rojo y el nodo final con un círculo azul. En la Figura 66

se identificaron con círculos verdes los nodos por los cuales pasó el robot. En la Figura 68 se marcaron con círculos de distintos colores las rutas posibles que se podía tomar para llegar a la meta finalmente el algoritmo tomó la ruta marcada con círculos verdes.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

Figura 61: Primera prueba con obstáculos.

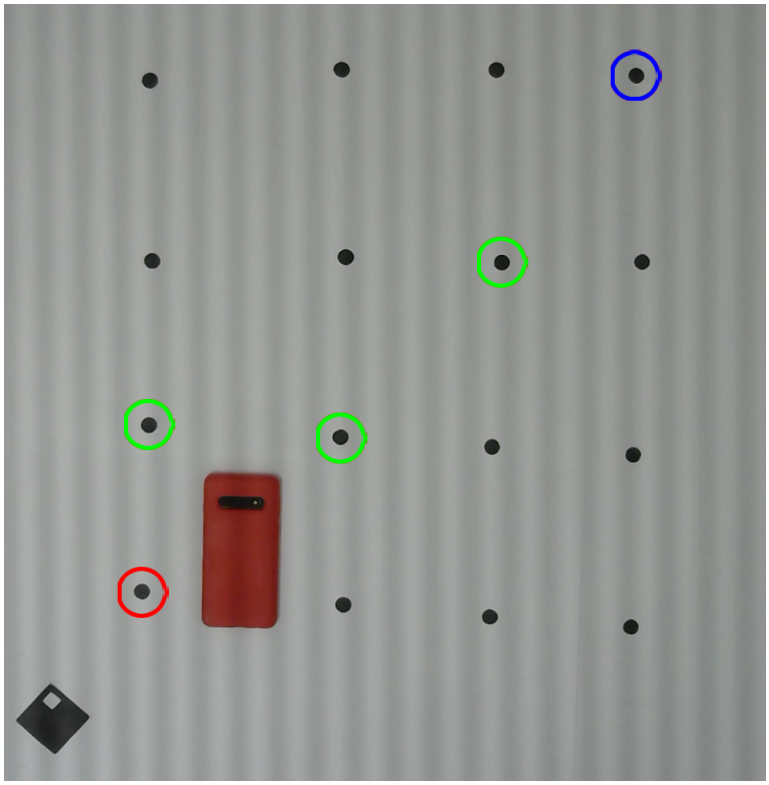


Figura 62: Proceso de validación. Punto inicial.

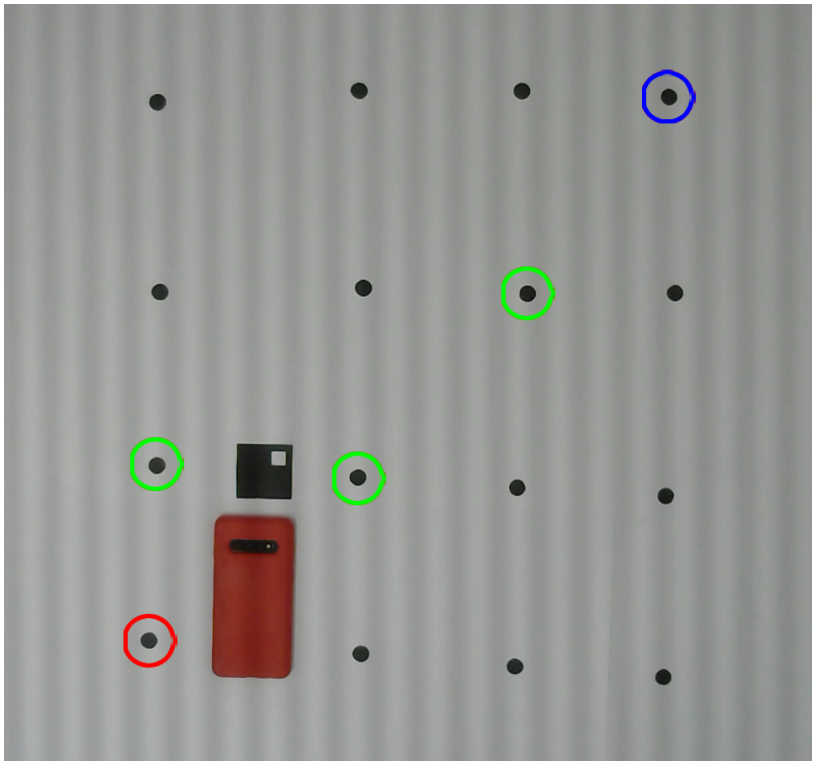


Figura 63: Proceso de validación. Punto medio.

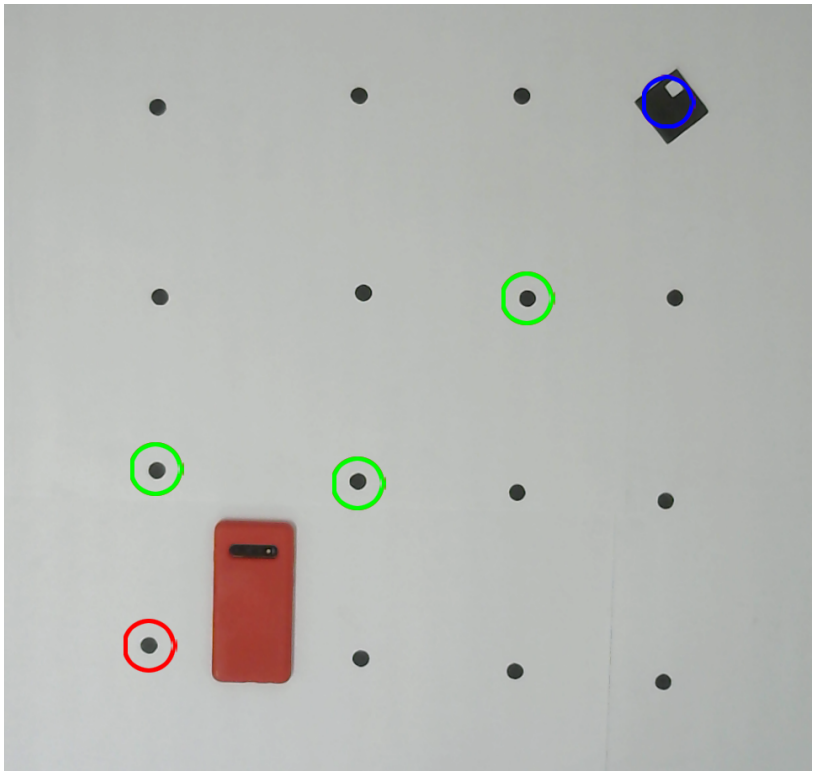
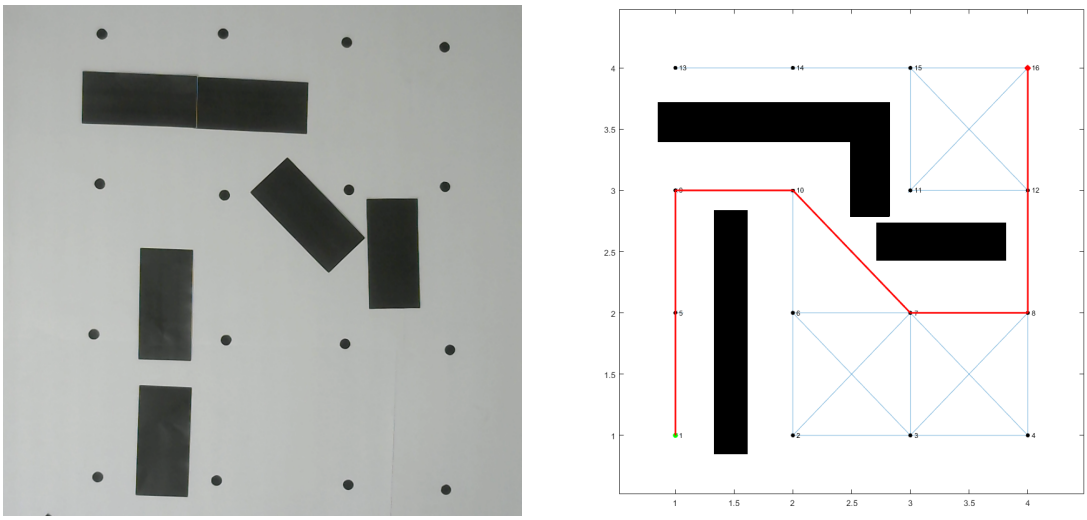


Figura 64: Proceso de validación. Punto final.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

Figura 65: Segunda prueba con obstáculos.

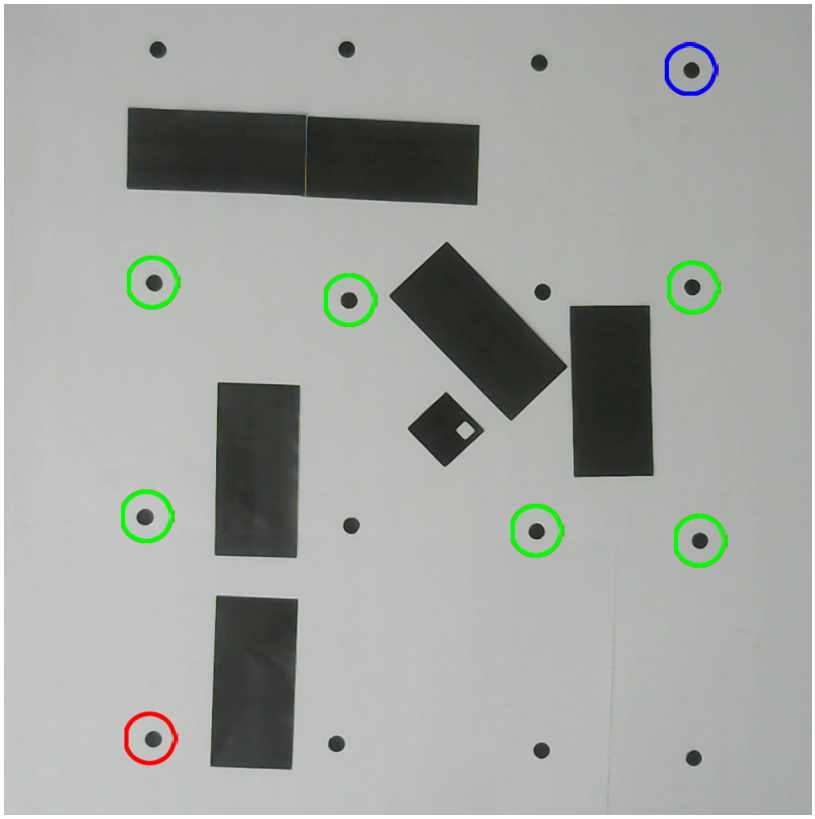
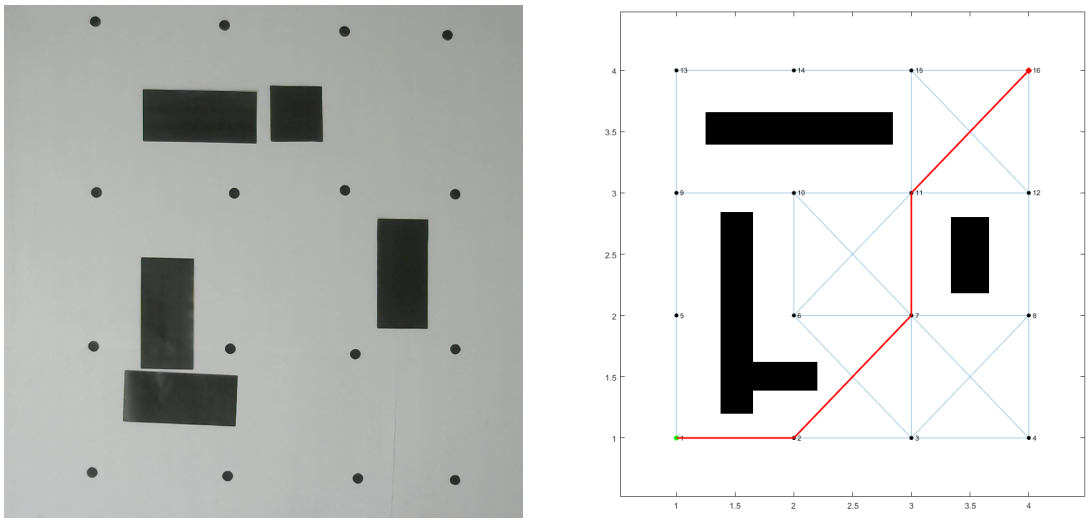


Figura 66: Proceso en la mesa de pruebas.



(a) Mapa en la mesa de pruebas.

(b) Trayectoria generada.

Figura 67: Tercera prueba con obstáculos.

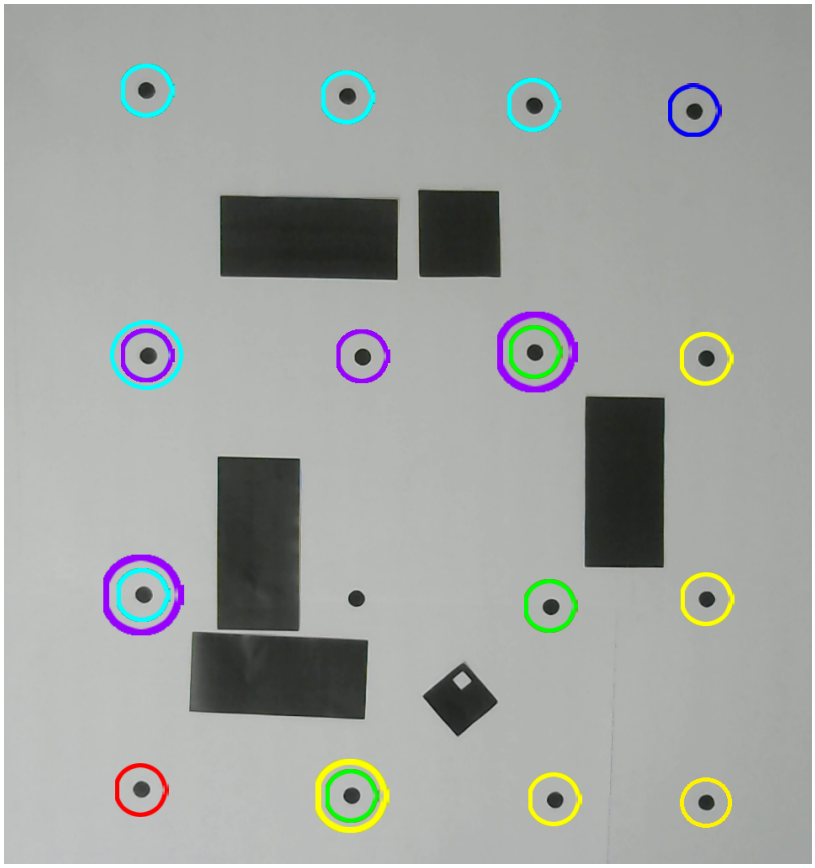


Figura 68: Proceso en la mesa de pruebas.

10.2. Implementación de un sistema de planificación de trayectorias dinámico

Una vez se validó el funcionamiento del algoritmo con las pruebas presentadas anteriormente se procedió a realizar pruebas para poder implementar un sistema dinámico que permite al robot volver a calcular una nueva ruta una vez se empezó el movimiento. El objetivo de este sistema es poder tener un funcionamiento más versátil que permita modificar la trayectoria del robot si se llega a detectar un nuevo obstáculo que bloquee la ruta inicial.

La implementación de este sistema está pensada para que sea implementada en la plataforma móvil utilizando los sensores del robot en combinación con la plataforma de rastreo, de esta manera tener dos maneras de detectar obstáculos en el mapa.

10.2.1. Validación del sistema dinámico

Para la validación de este sistema, se simuló el envío de datos proveniente de la plataforma de rastreo y se realizó una prueba simple para verificar su funcionamiento. Como se puede observar en el rectángulo marcado en la Figura 69 se calculó una ruta con el mapa inicial y se comienza el movimiento del robot, una vez llega al primer nodo, se detiene los motores, se actualiza la nueva meta y se reanuda el movimiento de los motores (proceso marcado con un rectángulo celeste. Una vez el robot pasa por el primer nodo (nodo marcado con un círculo celeste en la Figura 70) se detecta un nuevo obstáculo como se observa en la Figura 71 (marcado con un rectángulo rojo), también se puede observar que se detienen los motores, se asigna el nuevo nodo inicial y se espera a recibir el nuevo mapa (marcado en el rectángulo celeste).

Luego de recibir el nuevo mapa con el obstáculo detectado se procede a calcular la nueva ruta a seguir. Como se observa en la Figura 72, el obstáculo impide que el robot siga la trayectoria inicial por lo que es necesario esquivar este objeto para poder llegar a la meta. En la Figura 74 se puede la nueva ruta calculada, esta ruta se puede visualizar en la Figura 73 donde se observa que se logró esquivar el obstáculo detectado. Luego se puede observar el proceso que se realizo para validar este sistema donde se envían las coordenadas del robot cuando se encuentra en los nodos de las trayectoria calculada hasta llegar a la meta como se observa en la Figura 75

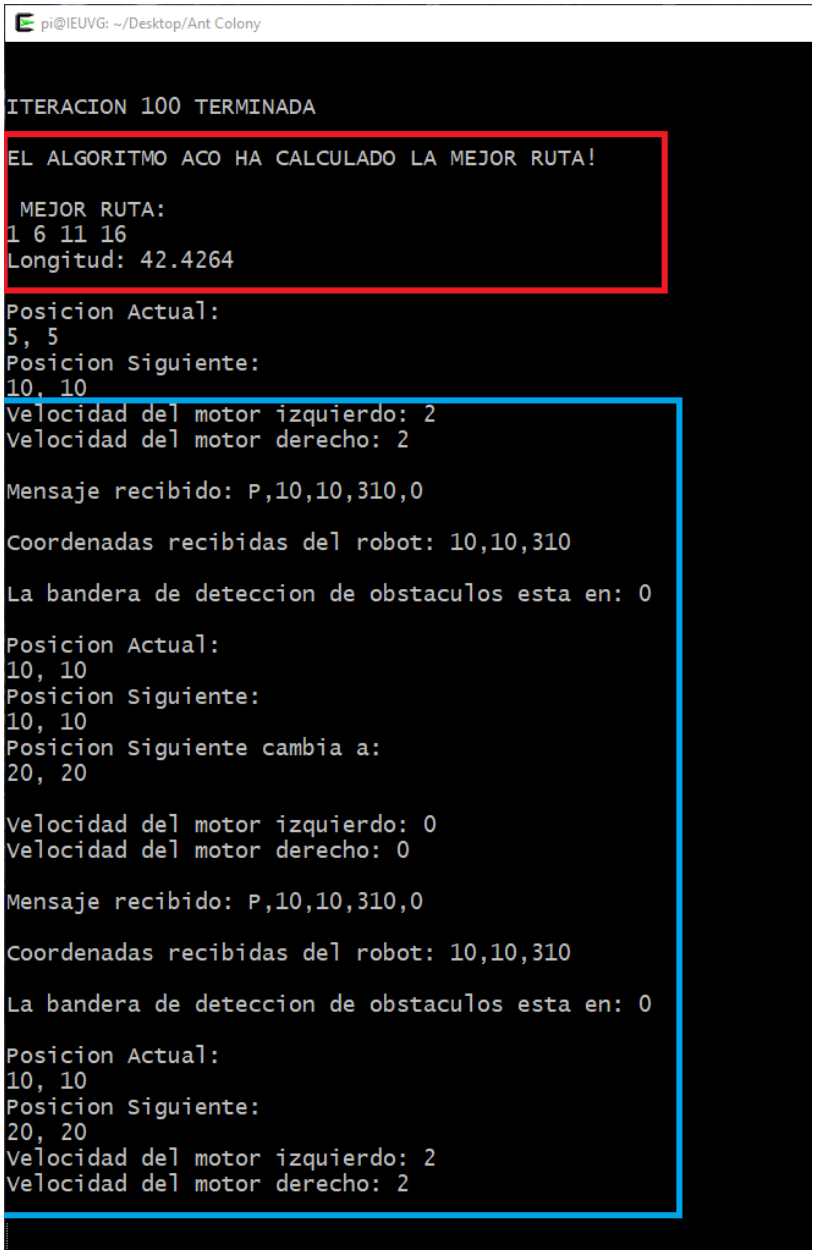


Figura 69: Proceso de validación.

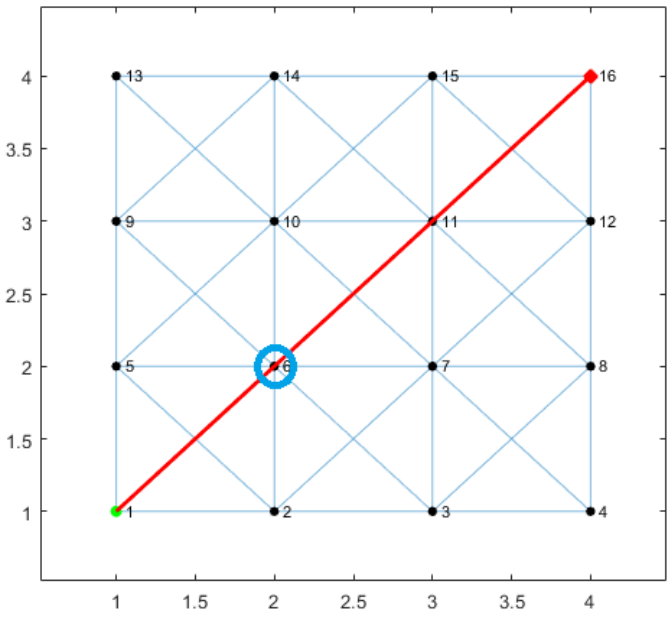


Figura 70: Ruta inicial.

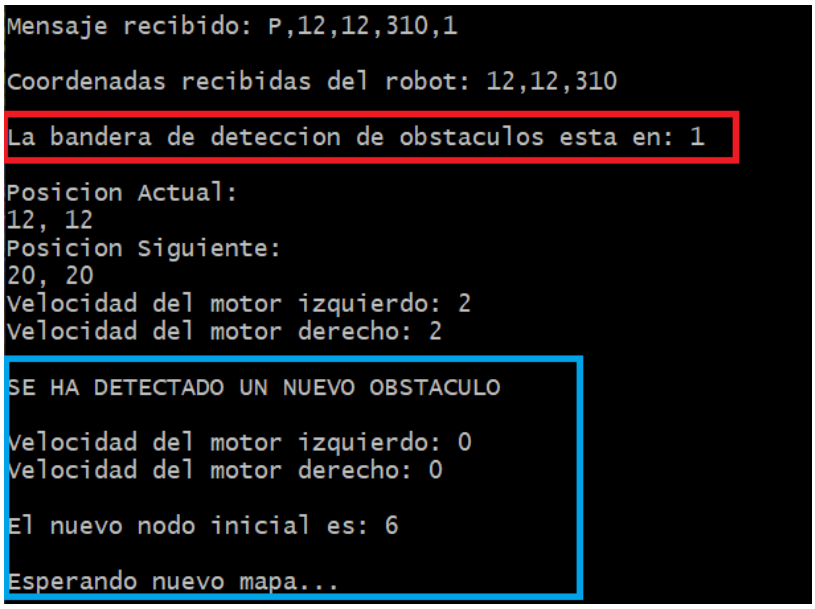


Figura 71: Proceso de validación.

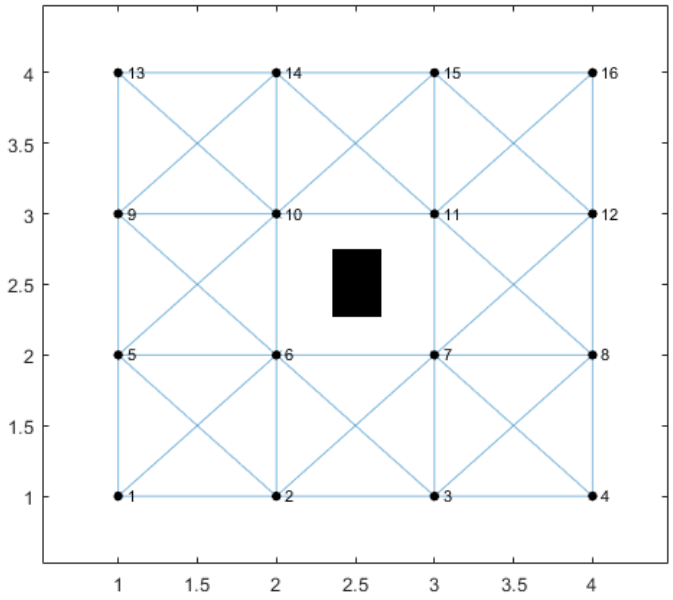


Figura 72: Obstáculo detectado.

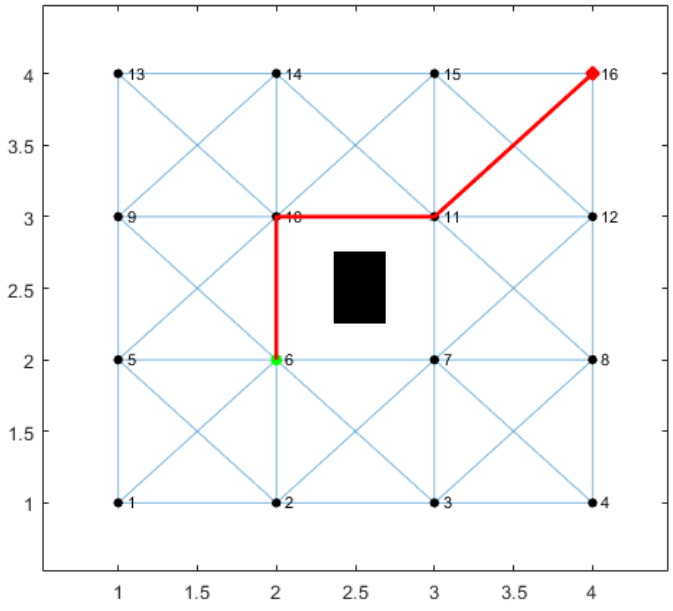


Figura 73: Ruta modificada.

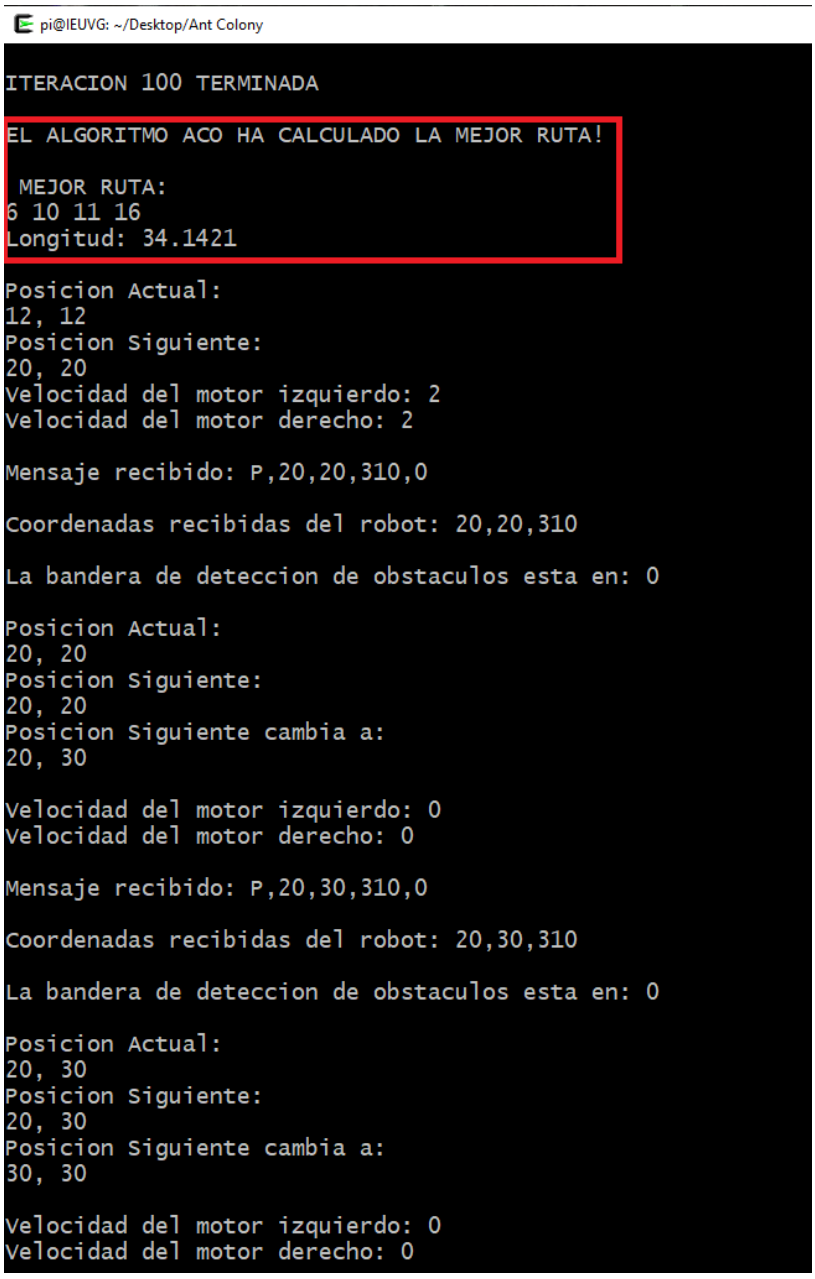


Figura 74: Proceso de validación.

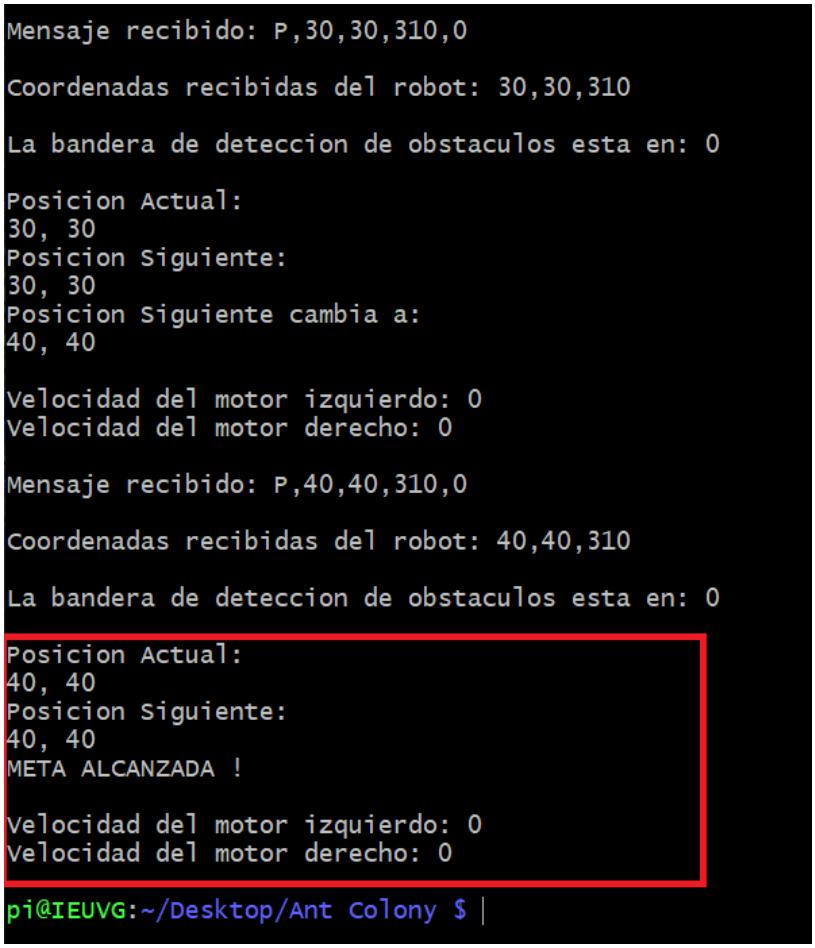


Figura 75: Proceso de validación.

CAPÍTULO 11

Conclusiones

- Según los criterios definidos en esta investigación se concluye que el ordenador Raspberry Pi y lenguaje de programación C++ son las mejores opciones para poder realizar la migración del algoritmo *Ant Colony Optimization*.
- Es posible verificar la migración del algoritmo AS al ordenador RPi, debido a que mediante esta plataforma y a través del algoritmo AS se encuentra un camino óptimo entre el nodo inicial y final.
- La ruta encontrada por el AS es una ruta óptima debido a que la longitud de la trayectoria seleccionada es la menor dentro de todas las posibilidades.
- El protocolo de comunicación UDP permitió el intercambio de información, debido a que se logra recibir toda la información del grafo y también la información sobre la pose del robot (marcador).
- Se obtuvieron resultados satisfactorios en la mesa de pruebas utilizando la plataforma de rastreo con visión por computadora con un error de posición de 1 cm.
- Se valida el funcionamiento del algoritmo ante cambios en el mapa. Esto se verifica con los resultados del sistema dinámico implementado en las últimas pruebas donde el algoritmo es capaz de cambiar de ruta al detectar un obstáculo nuevo.

CAPÍTULO 12

Recomendaciones

- Se recomienda para futuras implementaciones considerar el uso de la Raspberry Pi Zero, ya que esta posee características similares a la RPi 3 y RPi 4 que fueron utilizados en este trabajo. La ventaja de utilizar este dispositivo es que tanto el tamaño como el peso son menores comparados con las otras plataformas. Con el uso de un módulo de conexión wireless se puede realizar una mejor adaptación a una plataforma móvil.
- Para una futura implementación cuando se tenga una plataforma móvil disponible, adaptar los resultados de este trabajo para verificar el funcionamiento del planificador de trayectorias.
- Al momento de tener disponible la plataforma móvil sería interesante implementar los robots y realizar las mismas pruebas que se realizaron en este trabajo en la mesa de pruebas con la plataforma de rastreo con visión por computadora.
- Probar el AS con un grafo no cuadrado, es decir, construir un grafo con nodos en puntos aleatorios en la mesa de pruebas y observar el comportamiento del algoritmo.
- Realizar las mismas pruebas que se hicieron en este trabajo, pero utilizando la mesa de pruebas de la plataforma de experimentación robótica Robotat del laboratorio de robótica del CIT la cual utiliza sistema de captura de movimiento OptiTrack.
- Adaptar el sistema dinámico para que funcione con sensores, es decir implementar unos sensores ultrasónicos o similares para que junto con la plataforma de rastreo con visión por computadora sean ambos los responsables enviar la señal que indica la detección de un nuevo obstáculo y al tener la plataforma móvil utilizar los sensores de esta plataforma.

CAPÍTULO 13

Bibliografía

[1] G. Iriarte, "Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2020.

[2] Wyss Institute, *Progrmable Robot Swarms*, https://wyss.harvard.edu/technology/programmable-robot-swarms/, 2014.

[3] Self-Organizing Research Group, *Kilobot*, https://ssr.seas.harvard.edu/kilobots, 2021.

[4] Georgia Institute of Technology, *Robotarium*, https://www.robotarium.gatech.edu/, 2017.

[5] Hera Laboratory, *Georgia Institute of Technology*, https://herainc.com/portfolio/georgia-institute-of-technology/, 2021.

[6] L. Marín, M. Vallés, Á. Valera **and** P. Albertos, "Implementation of a bug algorithm in the e-puck from a hybrid control viewpoint," *2010 15th International Conference on Methods and Models in Automation and Robotics, Miedzyzdroje, Poland*, **pages** 174–179, 2010.

[7] A. Aguilar, "Algoritmo Modificado de Optimización de Enjambre de Particulas (MPSO)," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2019.

[8] E. Santizo, "Aprendizaje Reforzado y Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2020.

[9] National Geographic Society, *Theory fo evolution*, https://www.nationalgeographic.org/encyclopedia/theory-evolution/, 2019.

[10] T.-P. Hong, C.-K. Ting **and** O. Kramer, "Applied Computational Intelligence and Soft Computing," *Hindawi Publishing Corporation*, **journvol** 2010, **number** 360796, 2015. doi: https://doi.org/10.1155/2010/360796.

[11] A. P. Engelbrecht, *Computational Intelligence: an introduction*. Pretoria, Sudáfrica: Willey, 2008.

[12] S. Wang, Y. Zhang **and** G. Ji, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," *Hindawi Publishing Corporation*, **jourvol** 2015, **number** 931256, 2015. doi: https://doi.org/10.1155/2015/931256.

[13] M. Dorigo, M. Birattari **and** T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, **jourvol** 1, **number** 4, doi: 10.1109/MCI.2006.329691. **pages** 28–39, 2006.

[14] M. Wahab, N.M. Samia y A. Atyabi, *Ant Colony Optimization Algorithm processes*, DOI: https://doi.org/10.1371/journal.pone.0122827.g002, 2015.

[15] K. Doshi **and** P. Varman, "Optimal Graph Algorithms on a Fixed-Size Linear Array," *IEEE Transactions on Computers*, **jourvol** 36, **number** 04, **pages** 460–470, **april** 1987, ISSN: 1557-9956. DOI: 10.1109/TC.1987.1676928.

[16] Raspberry Pi Foundation, *Raspberry Pi documentation*, https://www.raspberrypi.org/documentation/faqs/#introduction, 2020.

[17] P. Corke, *Robotics, Vision and Control*. Berlin, Heidelberg: Springer, ISBN: 978-3-642-20144-8, 2017.

[18] K. Lynch **and** F. Park, *Modern Robotics: mechanics, planning and control*. Cambridge, UK: Cambridge University Press, ISBN: 978-1-107-15630-2, 2017.

[19] GCtronic, *E-Puck Education Robot*, http://www.e-puck.org/, Accessed: 2018-02-23, 2018.

[20] University of York, *Pi-puck documentation*, https://pi-puck.readthedocs.io/en/latest/, 2020.

[21] W. Fan, M. Liu, P. Lu **and** Q. Yin, "Graph Algorithms with Partition Transparency," *IEEE Transactions on Knowledge Data Engineering*, **number** 01, **pages** 1–1, **july** 2020, ISSN: 1558-2191. DOI: 10.1109/TKDE.2021.3097998.

[22] B. Siciliano, L. Sciavicco, L. Villani **and** G. Oriolo, *Robotics: modeling, planning and control*. Nápoles, Italia: Springer, DOI: 10.1007/978-1-84628-642-1, ISBN: 978-1-84628-641-4, 2009.

[23] T. Nishizeki, A. Brandsradt **and** S. Arumugam, *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*. (1st ed.): Chapman **and** Hall/CRC. https://doi.org/10.1201/b19163, (visitado 05-09-2021).

[24] F. Martins, M. Sarcinelli **and** R. Carelli, "A Velocity-Based Dynamic Model and Its Properties for Differential Drive Mobile Robots," *J Intell Robot Syst*, **number** 277-292, 2017. doi: https://doi.org/10.1007/s10846-016-0381-9.

[25] M. Egerstedt, "Control of Mobile Robots, Intruduction to Controls," *Birkhäuser Boston*, 2014. doi: https://doi.org/10.1007/0-8176-4404-0_33.

[26] I. R. Nourbakhsh **and** R. Siegwart, *Introduction to Autonomous Mobile Robots*. MIT Press, ISBN: 9780262195027, 2004.

[27] J. Barton **and** L. Nackman, *Scientific an Engeneering C++. An indtroduction with advanced techniques and examples*. New York, USA: Addison-Wesley, 2004.

[28] F. Martinez **and** A. Murillo, "Multi-threaded Spotted Hyena Optimizer with thread-crossing techniques," *Elsevier Science B.V.*, **number** 360796, 2021. doi: https://doi.org/10.1016/j.procs.2021.01.026.

[29] P. Baker **and** J. Whalen, "Survey of trade study methods for practical decition makinng," *University Drive, Fairmont, WV*, 2010.

[30] D. Beale **and** J. Bonometti, *System engeneering tools*. Capítulo 4, 2002.

[31] W. Ricks, M. Guynn **and** A. Hahn, "NASA Systems Analysis and Concepts Directorate Mission and Trade Study Analysis," *ResearchGate*, 2006. DOI: 10.2514/6.2006-7026.

[32] L. Young, J. Yetter **and** M. Guynn, "System Analysis Applied to Autonomy: Application to High- Altitude Long-Endurance Remotely Operated Aircraft," *ResearchGate*, 2005. DOI: 10.2514/6.2005-7103.

[33] W. Gutjahr, "First Steps to the Runtime Complexity Analysis of Ant Colony Optimization," *Department of Statistics and Decision Support System, University of Vienna*, visitado en 2021.

[34] J. Anderson, "Python vs C++: Selecting the Right Tool for the Job," 2019.

[35] M. Ateeq, H. Habib, A. Umer **and** M. Rehman, "C++ or Python? Which One to Begin with: A Learner's Perspective," **in** *2014 International Conference on Teaching and Learning in Computing and Engineering (LaTiCE)*, Los Alamitos, CA, USA: IEEE Computer Society, **april** 2014, **pages** 64–69. DOI: 10.1109/LaTiCE.2014.20. **url**: https://doi.ieeecomputersociety.org/10.1109/LaTiCE.2014.20.

[36] Adafruit. (2020). "Adafruit Ultimate GPS Breakout," **url**: https://www.adafruit.com/product/746.

[37] Daniel Hertz, *Learn how to connect and configure a GPS receiver to your Raspberry Pi 4 for a variety of fun projects!* https://maker.pro/raspberry-pi/tutorial/how-to-use-a-gps-receiver-with-raspberry-pi-4, 2020.

[38] J. I. Ramirez, "Herramienta de software de visión por computadora para aplicaciones de robótica de enjambre en una mesa de prueba - Fase III," Tesis de licenciatura, Universidad Del Valle de Guatemala (Tesis en desarrollo), 2021.

[39] J. P. Guerra, "Algoritmos de visión por computadora para el reconocimiento de la pose de agentes empleando programación orientada a objetos y multihilos," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2020.