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

Walter Andres Sierra Azurdia

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

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,

2021

Vo.Bo.:

(f) _____

Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) _____

Dr. Luis Alberto Rivera Estrada

(f) _____

(f) _____

Fecha de aprobación: Guatemala, 5 de diciembre de 2021.

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 *Particle Swarm Optimization* (PSO) para el movimiento de robots diferenciales. Para que los robots móviles puedan trasladarse del origen a la meta es necesario contar con un planificador de trayectorias. También fue implementado el algoritmo *Ant Colony Optimization* (ACO) con el objetivo de comparar ambos algoritmos y tener una alternativa de planificación de trayectorias para 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 de Raspberry Pi y lenguaje de programación de C++ para la implementación de este algoritmo en un sistema físico y se comparan los resultados obtenidos en los sistemas físicos con los obtenidos en las simulaciones.

Para lograr la comunicación entre los agentes, denominados hormigas, se implemento la programación multihilos y transmisión de datos mediante *broadcast* de cliente-servidor. Este implementación hace más eficiente el proceso de correr el algoritmo ya que 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.

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. In order for mobile robots to be able to move from the origin to the goal, it is necessary to have a trajectory planner. The *Ant Colony Optimization* (ACO) algorithm was implemented as well, with the aim of comparing both algorithms and having a trajectory planning alternative for 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 results obtained n physical systems are compared with those obtained in simulations.

In order to achieve communication between the agents, called ants, multithreaded programming and data transmission was implemented through *broadcast* client-server. This implementation makes the process of running the algorithm more efficient since it allows the execution of several tasks simultaneously. Likewise, the controllers that in the previous works presented the smoothest speed response were implemented and these are the pose controller and the Lyapunov pose controller.

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 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. También se presenta la migración del algoritmo *Ant System* en la plataforma Raspberry Pi y en lenguaje de programación C++. Asimismo se valida el funcionamiento de esta migración mediante simulaciones donde se permite la visualización de la feromona depositada por cada hormiga y el camino que el agente deberá seguir para que pueda llegar a la meta.

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 [1].

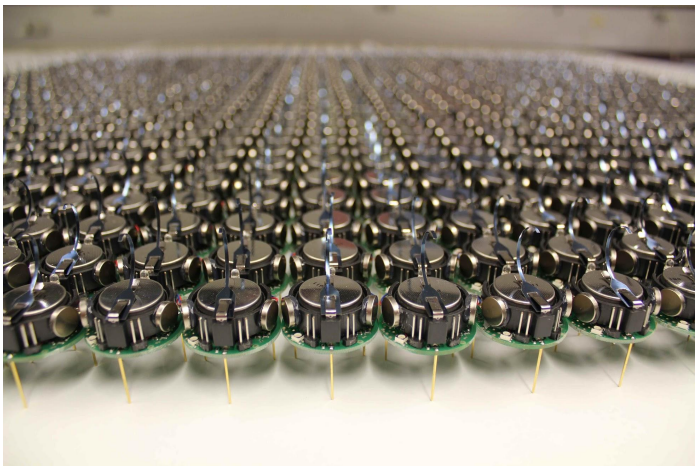


Figura 1: Kilobot del instituto Wyss [2].

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 ejecutara en el robot real a través de la interfaz web, registrarse en la pagina y esperar la aprobación para utilizar la plataforma [3].

Figura 2: Robotarium [4].

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 [5] 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 [5].

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 [6], 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 [7] 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 *Gridworld*, 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 trayectoria 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 [8], en el cual se implementó el algoritmo de inteligencia de enjambre *Ant Colony Op* 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 [7], 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. Este consiste en un método de optimización basado en el comportamiento de colonias de hormigas para determinar el camino mas corto entre la colonia y la fuente de alimento.

En la fase anterior, 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. También se implementaron distintos controladores para tener una métrica de comparación entre dichos controladores y poder comparar el desempeño con el algoritmo PSO. 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 se busca 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.

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 y enviando las señales. Con esta migración se podrá en un futuro proyecto adaptar esta migración a una plataforma móvil y una plataforma de rastreo por visión de computadora, para que de forma autónoma reciba el origen, la meta y las coordenadas de cada uno de los agentes y que estos de forma independiente realicen 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

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.

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 implementación del algoritmo *Ant Colony Optimization* en un sistema físico, en específico una Raspberry Pi. Para esto, se implemento 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.

En este trabajo no se realizan pruebas con una plataforma móvil que posea motores ya que esta plataforma esta siendo realizada en otro trabajo paralelo a este proyecto. La plataforma móvil mencionada se basa en la Raspberry Pi, por lo que los programas desarrollados en este trabajo podrán ser implementados en dicha plataforma.

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 también se encuentra en desarrollo actualmente. Por lo que no se pudieron realizar pruebas utilizando la plataforma mencionada.

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 esta 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 interaccion 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 termino 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 mas 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 mas 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 mas corto, así como se ejemplifica en la Figura 3, a partir de este experimento se desarrolló un modelo dado por la Ecuación 1 [13].

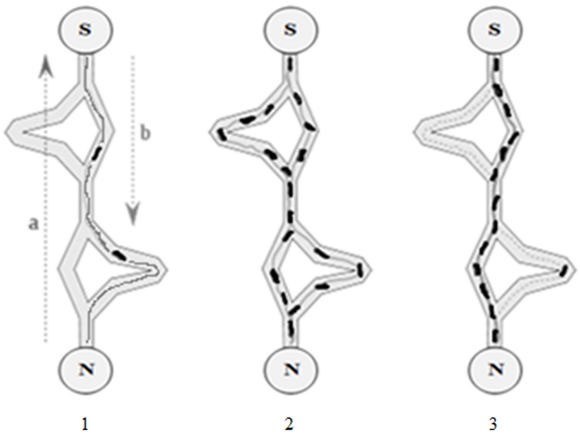


Figura 3: Proceso del algoritmo *Ant Colony Optimization* [14].

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

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} esta dado por 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 tambien considera el largo del camino L^K construido por la hormiga k y este se calcula como el numero 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 desplaza en los nodos disponibles para llegar a la meta. También notamos que, cada arista tiene un peso, la 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).

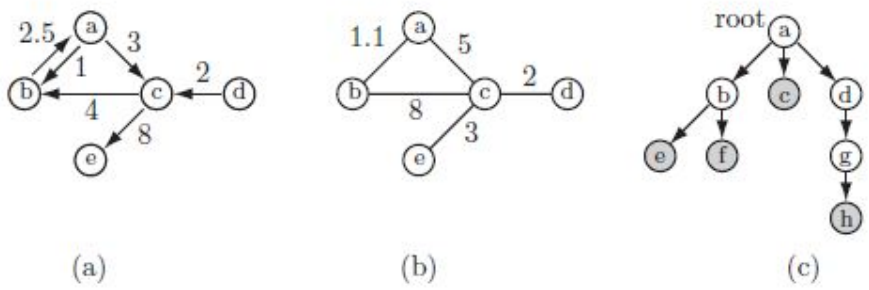


Figura 4: (a) Dígrafo con pesos. (b) Dígrafo con pesos sin dirección. (c) Grafo en forma de árbol [15]

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

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.

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

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

Con forme 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+

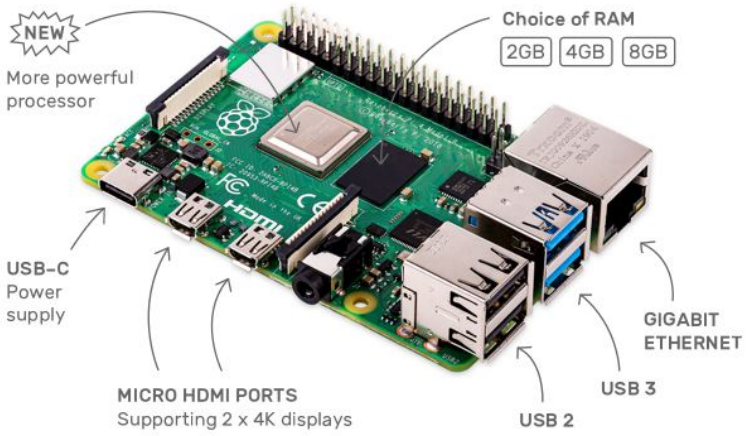


Figura 5: Raspberry Pi [16].

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 esta 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.

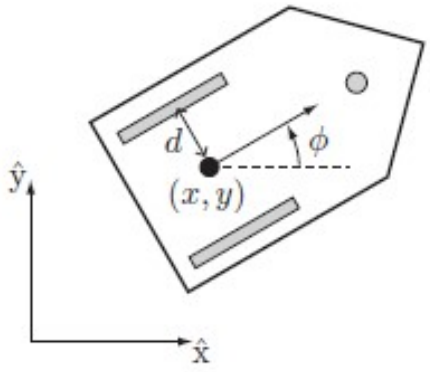


Figura 6: Robot diferencial móvil con dos llantas [18].

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 [5].

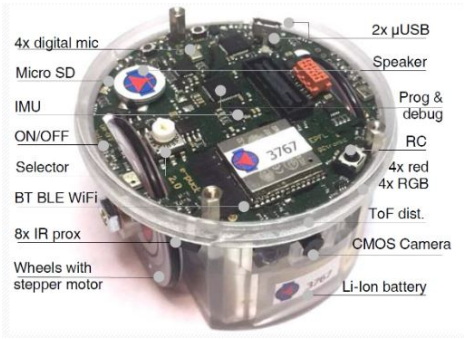


Figura 7: Sensores del robot E-Puck [19].

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 numero 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 encargara 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 [8], 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 podemos observar en [8], 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.

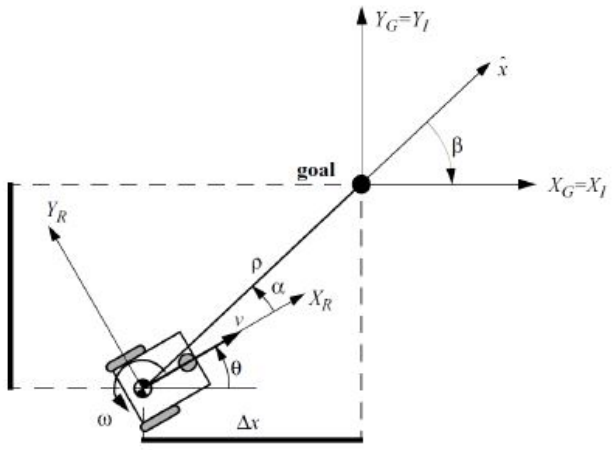


Figura 10: Cinemática de robot y referencias [26]

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 esta orientado en dirección a la meta esta dada por la Ecuación 14 y cuando el robot esta 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 contorlador 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 articulo [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 como en [28], incorporando elementos de recocido simulado y algoritmos genéticos, para mejorar el rendimiento del algoritmo ya planteado.

CAPÍTULO 7

Validación de plataforma

En este capítulo se presenta la validación de la selección del microcontrolador seleccionado 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 evaluados 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 calculo	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 ponderacion mas 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 %. En base a 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 entre agentes 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 [8], 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 calculo
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

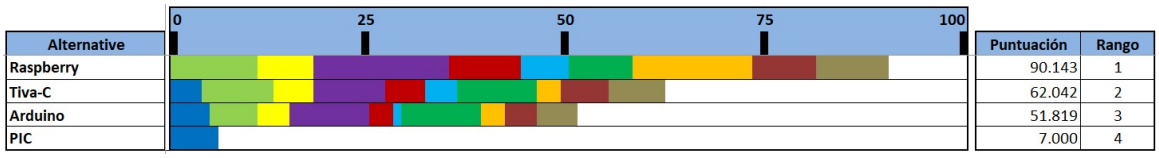
	Arduino Uno	Tiva C
SoC	ATmega328	TM4C123GH6PM
CPU	ATmega328p @ 16 MHz	ARM Cortex -M4F @80 MHz
Entorno de desarrollo	Arduino IDE	Energia
GPU	ATmega328	ARM Cortex
RAM	2 KB	32 KB SRAM
Almacenamiento	EEPROM 1 KB	EEPROM 2 KB
Ethernet	No	No
Wireless	No	No
Salidas de video	No	No
Precio	$ 15.00	$ 23.00

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 11, 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.



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*

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 13 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 12. Interpretar el código es más lento que correr un código nativo directamente.

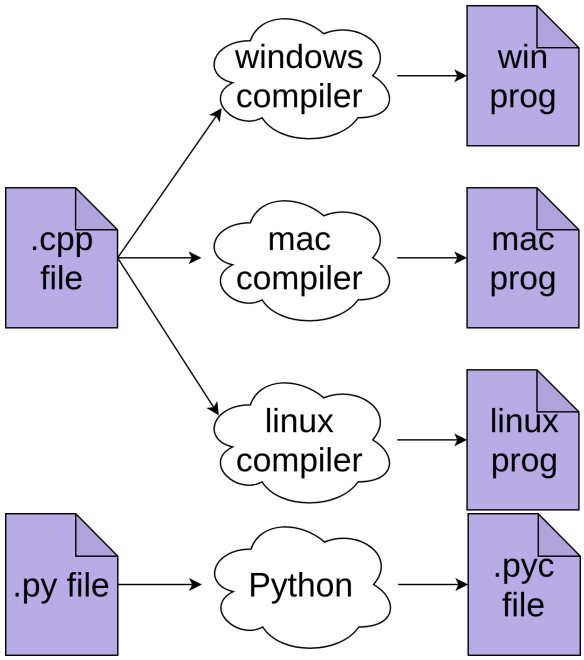


Figura 12: Tipo de compilación [34].

	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 13: 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 14: 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 15: 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. 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 que como resultado se obtuvo un mejor desempeño. Aunque estos parámetros pueden ser modificados para realizar las modificaciones pertinentes al observar el comportamiento del algoritmo.

8.1. Comunicación entre los agentes

Se implementa el uso de la comunicación multihilos para mejorar la eficiencia del programa, ya que requerimos estar tomando datos provenientes de un GPS, enviando coordenadas a los otros agentes u hormigas, correr el algoritmo para planificar el camino que se debe seguir y enviar los datos para el movimiento de los motores, con esta implementación no se interrumpe ningún proceso ya que en algunas ocasiones es necesario recibir algunos datos para poder trabajar.

Para poder inicializar la comunicación, todos los agentes que se comunicaran deben estar conectados a una misma red y se inicializa el hilo de transmisión definiendo la transmisión a través de la IP de *broadcast* de la red. Con el hilo definido se procede a crear la funcion para recibir los datos enviados para esto se define la función *receiving* que servirá para obtener y separar la data del buffer proveniente del broadcast.

8.2. Migración del algoritmo

Para la implementación del ACO, se implementa el uso de clases en C++. Para la creacion de la clase se debe crear un *header file* en el cual se define el nombre de la clase como ACO, dentro de este archivo se definen todos los miembros tanto privados como públicos se la clase ACO, tales como funciones necesarias para la implementación del algoritmo, asimismo se definen todos los espacios de memoria y los tipos para las variables a utilizar dentro del algoritmo así como las variables que se usaran para recibir información del GPS y mandar los datos a las otras hormigas.

8.2.1. Creación de la clase ACO

Ya con la definición que se realizo en el *header file*, en el cual ya se definió el tipo de variables y los argumentos de entrada en el caso de las funciones.

Se procede a crear todas las funciones ya definidas. 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 16 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 ultimo se colocaron las funciones que servirán para inicializar las variables y matrices con sus respectivos valores.

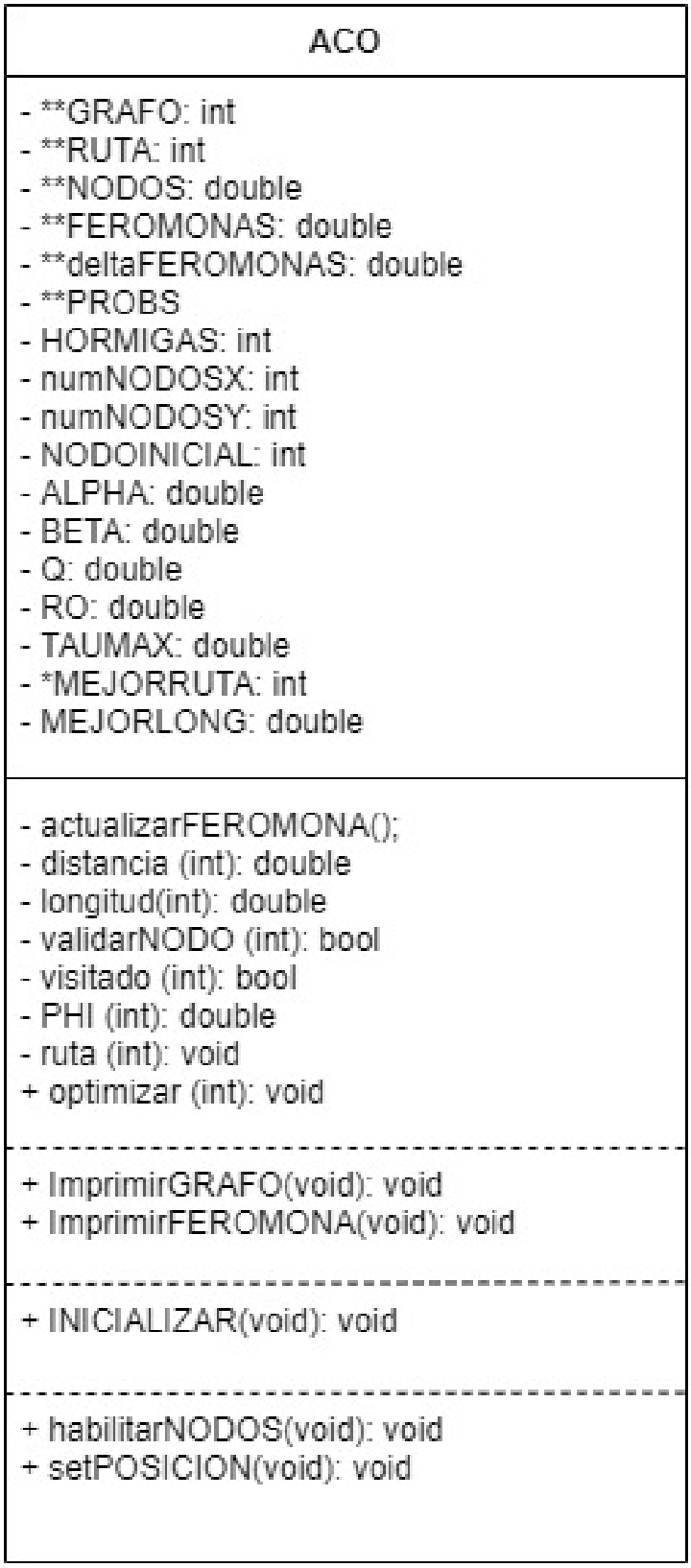


Figura 16: Diagrama de la clase

8.2.2. 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 (en este caso se utilizan doble punteros) se le asignarán los espacios de memoria corresponientes 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 y servirá para saber si el nodo donde se encuentra la hormiga ya fue visitado y por cuantos agentes este ha sido visitado.

8.2.3. Determinación de la trayectoria

Para iniciar el proceso de planificar la trayectoria, para lograr la mejor ruta si inicializa dando a cada hormiga un nodo inicial en el cual partirán todas. 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 función *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 que un porcentaje de las hormigas siga el mismo camino :
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
```

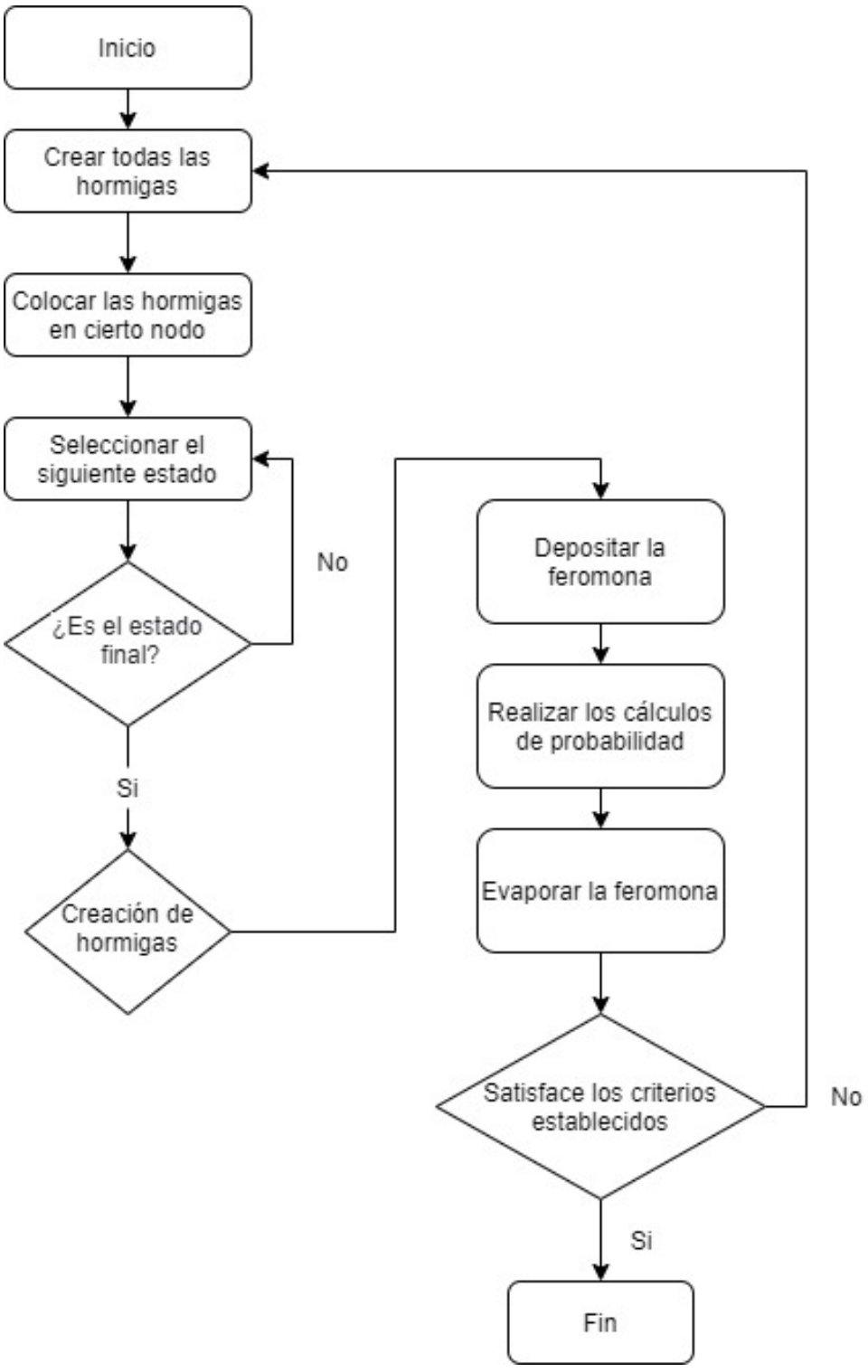


Figura 17: Diagrama de flujo del ACO.

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	8
Iteraciones	5
Tau maximo	2

8.3. Validación del ACO

En esta sección se presenta la validación la planificación de la trayectoria mediante el algoritmo *Ant System* implementado en la RPi, así como la validación de la comunicación entre los agentes y la recepción de las coordenadas mediante un módulo GPS.

8.3.1. 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 el 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.

Como se observa en la Figura 18, se identifican con un número uno los nodos validos y con cero los nodos inválidos. Como se puede observar en la matriz de la feromona, los valores se muestran únicamente en los nodos que son válidos.

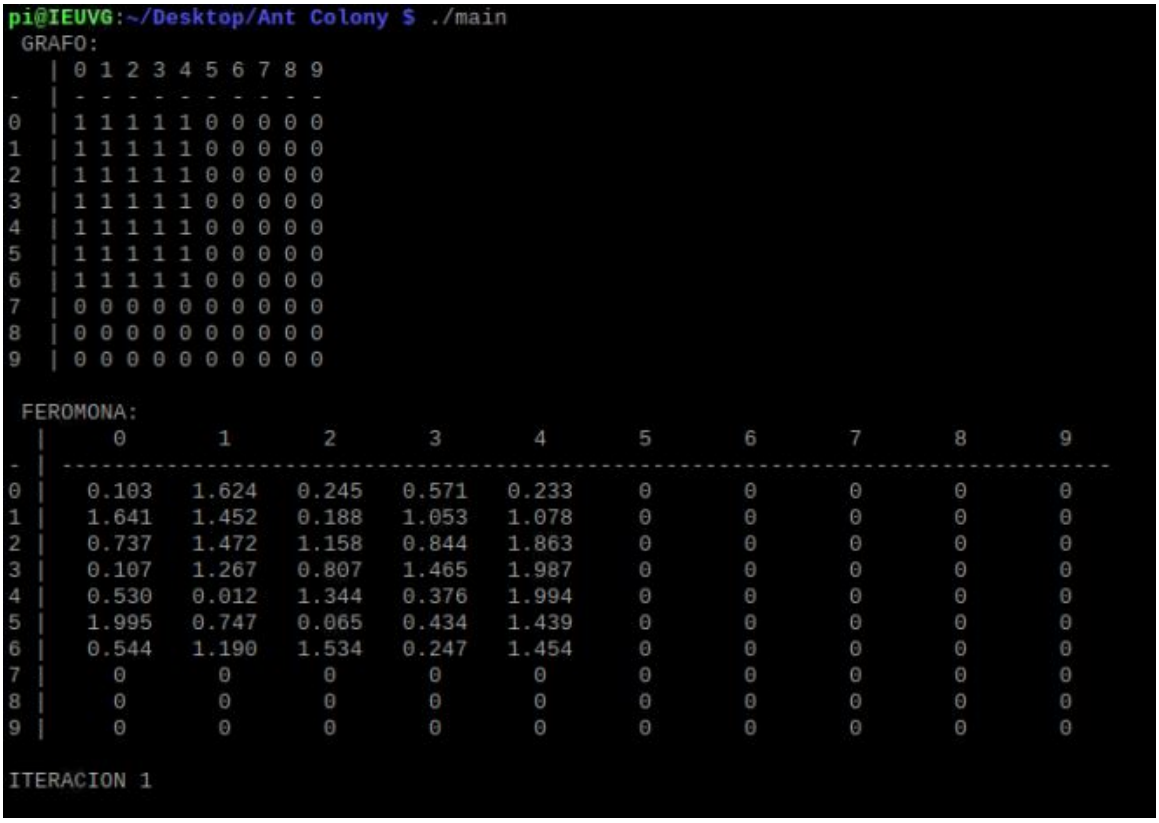


Figura 18: Variables de almacenamiento

8.3.2. Comunicación entre agentes

Para la validación del algoritmo los agentes deben ser capaces de comunicarse entre ellos y transmitir información para que los otros lo tomen en cuenta. Como punto inicial ya que no se cuenta con la plataforma de transmisión, esta se hace mediante una red local de un router en el cual se conectan todos los agentes en este caso varias RPi. Por medio de una transmisión a través de un broadcast se envía y recibe un buffer con la información deseada. Para validar la comunicación se envían desde dos RPi valores distintos de coordenadas, estos datos son recibidos por el otro agente y separados.

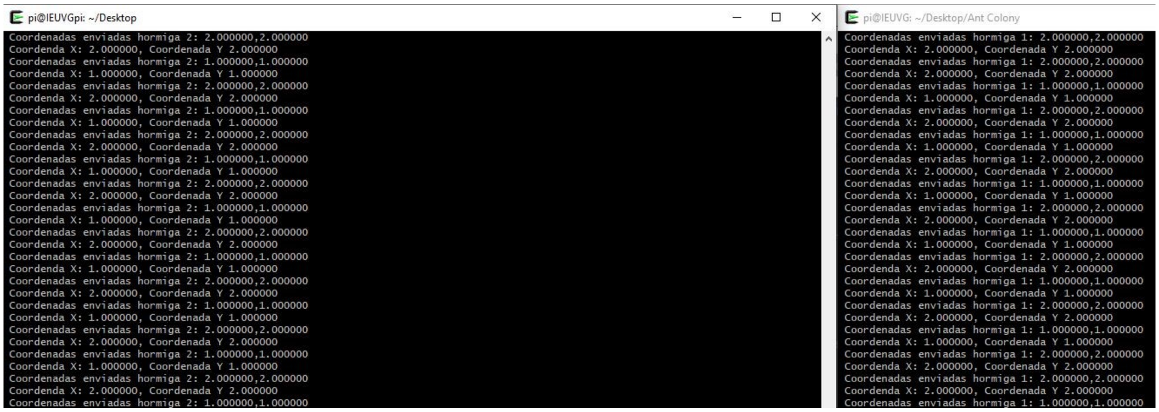


Figura 19: Envío y recepción de datos

8.4. Recepción de coordenadas

Para poder implementar el algoritmo ACO, es necesario que el agente sea capaz de saber su posición y ordinación para que este se pueda desplazar hasta llegar a la meta.

8.4.1. Módulo GPS

Para poder recibir coordenadas en x y y en tiempo real, se decidió implementar el uso del módulo Adafruit Ultimate GPS Breakout.

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 excelente 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 increíblemente bajo, solo 20 mA durante la navegación.

Para la recepcion de los datos enviados por el módulo se ejecutan los comando directos en la consola linux de la RPi descritos en el artículo [36].

Para que este módulo funcione de manera correcta, es necesario hacer una configuración a la RPi. Se debe desabilitar 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 20 y 21 y con esto se valida la recepción de datos por medio del puerto serial 0 de la RPi.

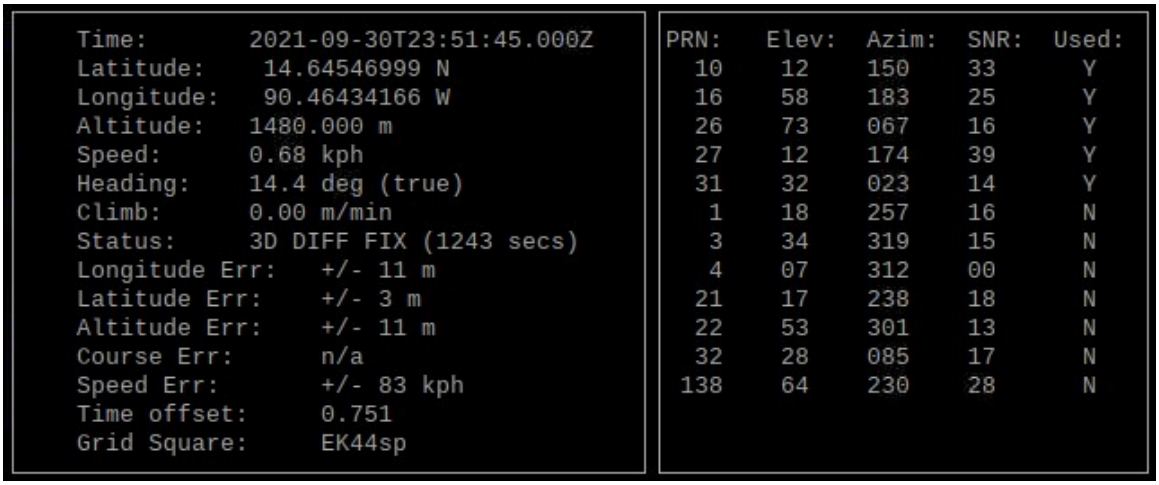


Figura 20: Recepcion de datos

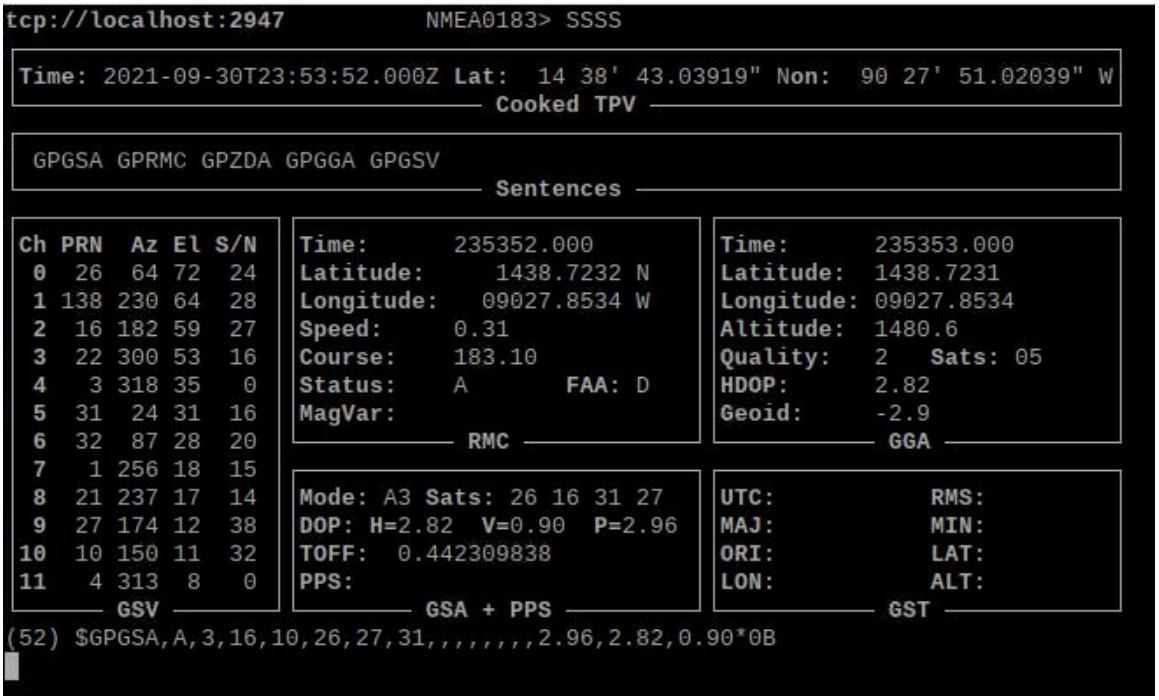


Figura 21: Recepción de datos

CAPÍTULO 9

Conclusiones

- En base a los resultados observados en el análisis del *trade study* se concluye que el ordenador Raspberry Pi es la mejor opción para poder realizar la migración del algoritmo *Ant Colony Optimization*.
- Es posible verificar el funcionamiento de la comunicación entre agentes, debido a que las RPi son capaces de enviar y recibir y estructurar datos mediante la transmisión de un buffer.
- Se puede verificar la recepción de datos mediante el módulo GPS, debido a que se puede visualizar una ventana con los valores que se reciben de dicho módulo.

CAPÍTULO 10

Recomendaciones

- 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 igual que la plataforma móvil, al momento de tener disponible en la mesa de pruebas la plataforma de rastreo por visión por computadora, sería interesante implementar esta plataforma a los robots y ver como se desempeña el algoritmo.
- Se recomienda utilizar *Visual Studio Code* ya que este editor de texto posee herramientas muy útiles como resaltado de texto y auto completado de texto. Herramientas que al tener un código muy grande resulta muy útil tener herramientas que faciliten la visualización del código.
- 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 una módulo de conexión wireless se puede realizar una mejor adaptación a una plataforma móvil.

CAPÍTULO 11

Bibliografía

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

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

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

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

[5] L. Marín, M. Vallés, Á. Valera y 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*, págs. 174-179, 2010.

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

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

[8] G. Iriarte, "Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica," 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 y O. Kramer, "Applied Computational Intelligence and Soft Computing," *Hindawi Publishing Corporation*, vol. 2010, n.º 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 y G. Ji, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," *Hindawi Publishing Corporation*, vol. 2015, n.º 931256, 2015. doi: https://doi.org/10.1155/2015/931256.

[13] M. Dorigo, M. Birattari y T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, n.º 4, doi: 10.1109/MCI.2006.329691. Págs. 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 y P. Varman, "Optimal Graph Algorithms on a Fixed-Size Linear Array," *IEEE Transactions on Computers*, vol. 36, n.º 04, págs. 460-470, abr. de 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 y 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 y Q. Yin, "Graph Algorithms with Partition Transparency," *IEEE Transactions on Knowledge Data Engineering*, n.º 01, págs. 1-1, jul. de 2020, ISSN: 1558-2191. DOI: 10.1109/TKDE.2021.3097998.

[22] B. Siciliano, L. Sciavicco, L. Villani y 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 y S. Arumugam, *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*. (1st ed.): Chapman y Hall/CRC. https://doi.org/10.1201/b19163, (visitado 05-09-2021).

[24] F. Martins, M. Sarcinelli y R. Carelli, "A Velocity-Based Dynamic Model and Its Properties for Differential Drive Mobile Robots," *J Intell Robot Syst*, n.º 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 y R. Siegwart, *Introduction to Autonomous Mobile Robots*. MIT Press, ISBN: 9780262195027, 2004.

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

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

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

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

[31] W. Ricks, M. Guynn y 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 y 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 y M. Rehman, "C++ or Python? Which One to Begin with: A Learner's Perspective," en *2014 International Conference on Teaching and Learning in Computing and Engineering (LaTiCE)*, Los Alamitos, CA, USA: IEEE Computer Society, abr. de 2014, págs. 64-69. DOI: 10.1109/LaTiCE.2014.20. dirección: https://doi.ieeecomputersociety.org/10.1109/LaTiCE.2014.20.

[36] 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.