Validación de los algoritmos de robótica de enjambre *Particle Swarm Optimization* y *Ant Colony Optimization* con sistemas robóticos físicos en el ecosistema Robotat

Jonathan Menéndez Cardona

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Validación de los algoritmos de robótica de enjambre *Particle Swarm Optimization* y *Ant Colony Optimization* con sistemas robóticos físicos en el ecosistema Robotat

Trabajo de graduación presentado por Jonathan Menéndez Cardona
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Validación de los algoritmos de robótica de enjambre *Particle Swarm Optimization* y *Ant Colony Optimization* con sistemas robóticos físicos en el ecosistema Robotat

Trabajo de graduación presentado por Jonathan Menéndez Cardona
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

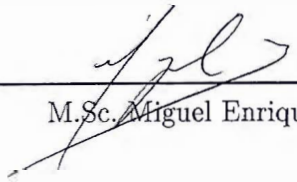
(f) 

Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) 

Dr. Luis Alberto Rivera Estrada

(f) 

M.Sc. Miguel Enrique Zea Arenales

(f) 

M.Sc. Pedro Iván Castillo Rivera

Fecha de aprobación: Guatemala, 13 de enero de 2024.

Prefacio

La concepción de este proyecto de graduación fue impulsada por un interés personal en el mundo de la robótica. Este interés desempeñó un papel fundamental en mi elección de cursar la carrera de Ingeniería Mecatrónica en la Universidad del Valle de Guatemala. Esa tesis requirió la aplicación de los conocimientos adquiridos a lo largo de los años en la carrera, por lo que quiero brindar un especial agradecimiento a todos los catedráticos que asistieron en mi formación y que me apoyaron durante las distintas etapas de aprendizaje.

Quiero agradecer el apoyo incondicional a mi familia, en especial a mis padres que me han acompañado tanto económica como moralmente en mi recorrido para convertirme en ingeniero. Sin su apoyo y formación a lo largo de mi vida no estaría en la posición de presentar este trabajo, por lo que estoy eternamente agradecido. Además, quiero agradecer al Dr. Luis Rivera, mi asesor, por su constante apoyo durante cada etapa de este proyecto de graduación y el tiempo que dedicó para que el resultado final fuera de la mejor calidad posible.

Índice

Lista de figuras

Lista de cuadros

Resumen

El siguiente trabajo de graduación se enfoca en la robótica de enjambre, en especial en la validación de los algoritmos de robótica de enjambre *Particle Swarm Optimization* (PSO) y *Ant Colony Optimization* (ACO) con robots móviles. Estos algoritmos fueron desarrollados y probados a nivel de simulación durante fases anteriores.

Para poder realizar pruebas en físico se llevó a cabo la migración de los algoritmos para usar los robots móviles Pololu 3pi+ en la plataforma de Robotat de la Universidad del Valle de Guatemala. Esta migración permitió la implementación de los algoritmos en un entorno real, lo que a su vez facilitó la comparación de los resultados obtenidos con las simulaciones previas. La comparación entre los resultados de las pruebas físicas y los resultados simulados fueron de gran importancia para validar y verificar los avances logrados en el desarrollo de los algoritmos.

Además, durante este proceso se llevaron a cabo pruebas y se establecieron criterios de comparación que permitieron realizar un análisis de los algoritmos. Se examinó en qué casos cada algoritmo muestra un mejor desempeño, y se identificaron los parámetros que ofrecen un funcionamiento óptimo en diferentes situaciones. Este análisis y comparación de los algoritmos en un entorno físico proporcionó una comprensión más profunda sobre su rendimiento, lo que permitió realizar ajustes y mejoras para optimizar su funcionamiento en robots móviles reales.

Abstract

The following graduation project focuses on swarm robotics, specifically on validating the Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) algorithms with mobile robots. These algorithms were developed and tested at the simulation level in earlier phases.

To conduct physical experiments, the algorithms were migrated for use with Pololu 3pi+ mobile robots on the Robotat platform at the Universidad del Valle de Guatemala. This migration enabled the implementation of the algorithms in a real-world environment, facilitating the comparison of results with previous simulations. The comparison between the outcomes of physical tests and simulated results was crucial for validating and verifying the progress achieved in algorithm development.

Additionally, during this process, tests were conducted, and comparison criteria were established to analyze the algorithms. The study examined under what circumstances each algorithm demonstrates better performance, and parameters offering optimal functionality in different situations were identified. This analysis and comparison of algorithms in a physical environment provided a deeper understanding of their performance, allowing for adjustments and improvements to optimize their operation on real mobile robots.

CAPÍTULO 1

Introducción

En las últimas décadas, la robótica de enjambre, inspirada en la naturaleza, ha experimentado un crecimiento considerable. El campo de la robótica de enjambre comenzó con pruebas en plataformas terrestres, pero no se limita a ellas. Trabajos recientes se han realizado en superficies acuáticas, robots submarinos y drones voladores[1]. La Universidad del Valle de Guatemala ha estado involucrada desde hace varios años en la investigación de temas relacionado con robótica de enjambre, fomentando así el desarrollo de algoritmos y la realización de pruebas que puedan abrir el paso a las aplicaciones en el mundo real.

En este documento se presenta un resumen parcial de los avances que ha tenido la Universidad del Valle de Guatemala en el campo de la robótica de enjambre, así como la nueva fase de los proyectos relacionados a los algoritmos de *Particle Swarm Optimization* (PSO) y *Ant Colony Optimization* (ACO). El avance se enfocó en la realización de pruebas físicas para los algoritmos mencionados, donde se buscaba utilizar los robots Pololu 3pi+ para recorrer las trayectorias otorgadas por un sistema centralizado. Se optimizaron las condiciones de los algoritmos y de los controladores para mejorar el rendimiento en físico.

Se presenta el proceso de migración que fue necesario seguir para poder adaptar el funcionamiento de los algoritmos de PSO y ACO, desde la simulación, hacia la aplicación en robots reales. Además, se presenta una serie de pruebas para ambos algoritmos, en las que se evalúa la mayor cantidad de alteraciones posibles para obtener información relevante sobre la robustez y adaptabilidad de los algoritmos.

CAPÍTULO 2

Antecedentes

2.1. Georgia Tech: Robotarium

El Instituto Tecnológico de Georgia desarrolló un laboratorio que permite la investigación y pruebas de robótica de forma remota [2]. Este proyecto provee una plataforma de libre acceso para realizar la verificación de algoritmos y trayectorias en la robótica de enjambre con robots reales, en lugar de quedarse puramente en simulaciones. Para poder realizar los experimentos deseados en los robots del Robotarium es necesario descargar un simulador, ya sea en Matlab o en Python, y verificar que el código a nivel de simulación en forma de un prototipo. Luego, se debe registrar con una cuenta en la página de Robotarium y esperar a que uno de los administradores apruebe la solicitud para poder acceder a la interfaz web, donde se podrá subir el código para su ejecución.

2.2. Avances en investigación de robótica de enjambres

Desde que el área de estudio de inteligencia de enjambre atrajo interés ha ido evolucionando hasta ser un proceso interdisciplinario, incluyendo inteligencia artificial, economía, sociología, biología, etc. La robótica de enjambre cuenta con características favorables al compararlos con otros sistemas, como que su control que puede ser descentralizado y autónomo, que es altamente flexible, que tiene una alta escalabilidad para seguir ejecutando acciones sin importar un cambio en la cantidad de agentes, entre otros. La robótica de enjambre puede ser aplicada en problemas que de otra forma sería difíciles de abordar, típicamente en la ayuda ante desastres naturales que impidan la intervención humana de forma directa, en la minería y hasta en el control de vehículos aéreos no tripulados [3].

Figura 1: Enjambres en la naturaleza que han inspirado la robótica de enjambre [3].

2.3. Megaproyecto Robotat

El departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala realizó un proyecto, desarrollado en varias fases, para promover el aprendizaje y realizar pruebas físicas de robótica de enjambre. La primera fase constó del diseño y posterior elaboración de una plataforma que implementa un algoritmo de visión de computadora para obtener datos de robots en una superficie plana que permitiera la comprobación futura de los algoritmos de enjambre.

Para tener una retroalimentación actual de cada agente dentro de la mesa de pruebas se utiliza el sistema de captura de movimiento OptiTrack. El software que se necesita para trabajar con OptiTrack es Motive, que nos permite obtener las coordenadas y orientación de unos marcadores que se colocan en cada uno de los agentes. Todo el sistema funciona por medio de un servidor elaborado por el Msc. Miguel Zea, que permite tanto la obtención de las coordenadas proporcionadas por las cámaras como el enviar coordenadas a la dirección IP que haga la petición de las mismas [4].

2.4. Particle Swarm Optimization (PSO)

En la segunda fase del Megaproyecto Robotat, Aldo Aguilar [5] implementó el algoritmo PSO clásico con modificaciones en una serie de parámetros que permita un mejor comportamiento del enjambre y que se adapta a las dimensiones físicas y cinemáticas de robots Bibots. El algoritmo incluye un planeador de trayectorias que ajusta el parámetro de inercia, el parámetro de constricción, dos parámetros de escalamiento y dos parámetros de uniformidad, optimizados para robots diferenciales.

Durante la investigación se pudo observar que era necesario el uso de controladores para poder seguir las trayectorias computadas por el planeador, generando velocidades suaves y continuas. Se realizó la evaluación de hasta siete diferentes controladores, por medio de simulaciones en el software Webots, donde se obtuvo que el controlador Transformación de Uniciclo con LQI (TUC-LQI) brinda los mejores resultados en cuanto a la genera tra-

yectorias, manteniendo sin cambio la velocidad pico debido a un amortiguador de control proporcional y evitando oscilaciones gracias a un amortiguador de control integral [6].

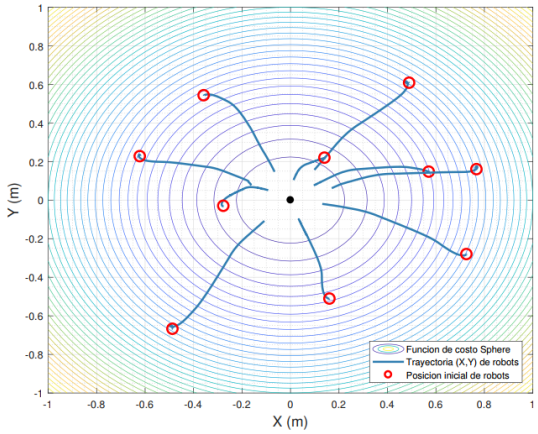


Figura 2: Trayectorias de enjambre en simulación con controlador TUC-LQI [6].

En la fase posterior, llevada a cabo por Alex Maas [7], se inició la implementación en físico del algoritmo Modified Particle Swarm Optimization (MPSO), evaluando distintas opciones de microcontroladores, sistemas embebidos, lenguajes de programación, entornos de desarrollo y robots móviles. La investigación necesitó el desarrollo de técnicas para la validación del algoritmo en un ambiente controlado debido a que no se contaban con plataformas móviles operativas en el momento. Durante esta etapa fueron desarrollados dos sistemas de comunicación, el primero para la obtención de la pose de cada agente dentro de la mesa de pruebas en tiempo real y el segundo que permite el intercambio de información útil entre los agentes para apoyar el algoritmo MPSO. Ambos sistemas de comunicación utilizan una programación multihilos para permitir la recepción de información de los agentes sin alterar la ejecución en paralelo del algoritmo de PSO modificado.

Por último, Rubén Lima [4] realizó las primeras pruebas en la plataforma de captura de movimiento de Robotat con plataformas móviles, que incluían un Bytebot y un Bytebot3B. Se encontró que los resultados obtenidos tenían ciertas diferencias en comparación de los logrados en simulación, tomando en cuenta que los robots tenían características diferentes entre sí y con la limitante de solo contar con dos robots para la validación.

2.5. Ant Colony Optimization (ACO)

Durante la tercera fase del proyecto Robotat, Gabriela Iriarte [8] implementó un algoritmo basado en Ant Colony que trata de replicar el comportamiento de la hormigas para encontrar un camino óptimo para buscar alimento y llevarlo al hormiguero. Uno de los objetivos principales de la investigación era compararlo con el algoritmo de MPSO [5] y comprobar si este podría ser una alternativa para la aplicación en pruebas físicas. Este algoritmo incluye parámetros como una feromona de la colonia para permitir al resto de agentes encuentren el camino entre el nodo inicial y el final deseado. Se evaluaron métodos de planificación de trayectorias con grafos y sin grafos, todo por medio de simulaciones,

donde los que si cuentan con grafos son más computacionalmente demandantes y los que no los incluyen tienen trayectorias no tan ideales entre el punto inicial y el punto final. De igual forma, una de las conclusiones obtenidas por la investigación es que si se desea controlar enjambres de robots de forma simultánea es mejor utilizar el algoritmo de MPSO debido a que puede llevar a varios agentes directamente al punto final deseado debido a su naturaleza vectorial.

En la siguiente fase Walter Sierra [9] tomo estos avances y el algoritmo planteado en [8] para poder migrarlos a una plataforma Raspberry Pi con la que se pudieran realizar pruebas en físico. Al no contar con una plataforma móvil funcional fue necesario realizar pruebas simples en ambientes controlados donde poder validar el funcionamiento correcto del algoritmo. Se implementó una programación multihilos y transmisión de datos mediante un protocolo UPD que permite la ejecución de varias tareas en simultáneo. Además, se implementó el controlador de pose y el controlador de pose de Lyapunov que en trabajos anteriores presentaron la respuesta más suave de velocidad [8]. Para la validación se realizaron dos escenarios, uno con un mapa sin obstáculos y otro con un mapa con varios obstáculos, donde se obtuvo que el funcionamiento del algoritmo al ser capaz de cambiar de ruta al detectar un obstáculo nuevo.

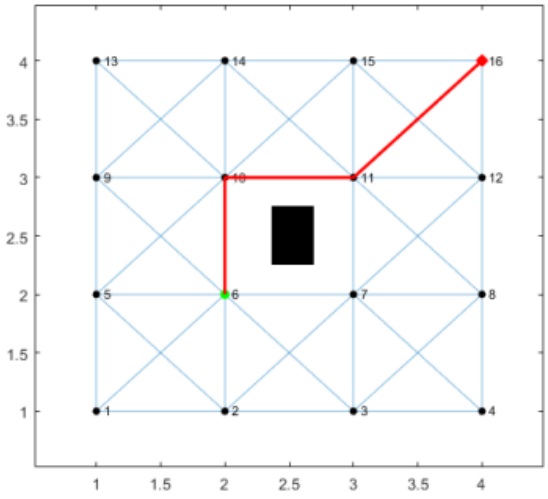


Figura 3: Ruta óptima generada por algoritmo ACO con un obstáculo [9].

CAPÍTULO 3

Justificación

La robótica de enjambre se inspira en los ejemplos de enjambres que encontramos en la naturaleza, como colonias de bacterias, colonia de hormigas, bancos de peces, bandadas de pájaros, entre otros. La idea es poder replicar estos comportamiento con robots que nos permitan utilizarlos en aplicaciones prácticas.

En el departamento de Ingeniería Macatrónica, Electrónica y Biomédica de la Universidad del Valle de Guatemala se lleva trabajando en distintos proyectos de robótica de enjambre durante varios años. Se han llevado a cabo distintas fases en las que se incluyen avances en la selección de distintos algoritmos de enjambre, el desarrollo de algoritmos para pruebas con plataformas móviles, el desarrollo de una plataforma de pruebas por medio de visión por computadora, la validación de algoritmos y selección de controladores en simulación, la evaluación de distintas opciones de microcontroladores, sistemas embebidos, lenguajes de programación, entornos de desarrollo y robots móviles. En la fase más reciente se iniciaron pruebas en físico para la validación de los avances en los algoritmos de *Particle Swarm Optimization* y *Ant Colony* con la restricción de no contar con las suficientes plataformas móviles capacitadas para realizar todas las pruebas necesarias.

Con este trabajo de graduación se busca avanzar en la línea de investigación de robótica de enjambre, aprovechando el ecosistema Robotat y la mayor cantidad de plataformas móviles con las que se cuenta. Se realizará una nueva validación de los algoritmos para encontrar trayectorias de agentes móviles. El proceso se continúa con la migración y desarrolló de los algoritmos necesarios que puedan ser aplicados en las robots móviles Pololu 3Pi+ que están disponibles, para luego hacer que los agentes ejecuten las trayectorias resultantes.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar y validar los algoritmos de robótica de enjambre *Particle Swarm Optimization* (PSO) y *Ant Colony Optimization* (ACO) en el ecosistema Robotat, utilizando las plataformas móviles Pololu 3Pi+.

4.2. Objetivos específicos

- Migrar los algoritmos de PSO y ACO desarrollados con anterioridad para poder ser utilizados en las plataformas móviles Pololu 3Pi+.
- Replicar pruebas y experimentos realizados en fases anteriores a nivel de simulación y con plataformas móviles y no móviles, ahora con las nuevas plataformas móviles disponibles para la mesa de pruebas del Robotat.
- Evaluar y comparar el funcionamiento de los algoritmos y las plataformas móviles en distintos escenarios y verificar bajo qué condiciones y parámetros se logran los mejores desempeños.

CAPÍTULO 5

Alcance

El alcance de este trabajo de graduación fue la migración de los algoritmos de PSO y ACO a sistemas móviles físicos, utilizando el sistema de captura de movimiento disponible para brindar información en tiempo real. Se ajustó el algoritmo de PSO para poder ser ejecutado dentro de un solo dispositivo, haciendo uso de ciclos y vectorización de las instrucciones del algoritmo, dando un enfoque centralizado al sistema. Se ajustó el espacio de trabajo para que las trayectorias generadas por los dos algoritmos se encuentren dentro de los límites reales de la mesa de pruebas del Robotat. Cabe resaltar que en ambos algoritmos se tomó la mesa como un espacio libre de obstáculos, con las únicas limitantes siendo los bordes de la mesa y las zonas exteriores, donde las cámaras del sistema de captura de movimiento pierden de vista a los agentes. Se limitaron las velocidades de los agentes a la hora de ejecución de las trayectorias, esto para agilizar la evaluación de la ejecución en las pruebas con múltiples agentes.

Se redujo la cantidad de controladores evaluados en comparación con fases anteriores, enfocándose en un controlador PID con acercamiento exponencial que permitiera un fácil ajuste de parámetros. Con el cambio se buscaba poder enfocarse en una migración completa y funcional de los algoritmo, antes que la mejor forma de control de las plataformas móviles. Además, se realizaron pruebas con cada algoritmo, evaluando las características óptimas para su funcionamiento eficiente y para lograr los objetivos de cada iteración. Sin embargo, se tuvo la limitante en la cantidad de agentes disponibles, al contar solo con diez Pololu 3pi+ disponibles y que estos son un recurso de uso compartido. En cuanto al espacio de trabajo, es importante resaltar que solo se pueden realizar pruebas dentro de la mesa del ecosistema Robotat, esto debido a que es la única ubicación que cuenta con el sistema de captura de movimiento OptiTrack que fue de vital importancia para la validación de los algoritmos.

CAPÍTULO 6

Marco teórico

6.1. Robótica de enjambre

La robótica de enjambre es la coordinación de múltiples robots para llevar a cabo una tarea de forma colectiva de forma más eficiente que un único robot. Comúnmente se desarrolla por medio de la comunicación autónoma entre los agentes del enjambre para poder interactuar entre sí y con el ambiente [10].

6.1.1. Taxonomía

Un enjambre debe de contar con una serie de comportamientos colectivos principales para que los agentes puedan afrontar cualquier aplicación compleja en la vida real. Estos comportamientos colectivos son clasificados en cuatro categorías principales [11]:

- Organización espacial: Son los comportamientos que permiten a los enjambres moverse de manera coordinada en su entorno, logrando organizarse y distribuirse tanto entre sí como en relación a los objetos presentes en el área.
  - Agregación: La meta es reunir a todos los robots en una región específica del entorno. Es fundamental para permitir que los robots estén suficientemente cerca para interactuar entre sí.
  - Formación de patrones: Tiene como objetivo desplegar robots de manera regular y repetitiva. Un caso especial es la formación de cadena, donde la idea es posicionar a los robots de manera que se conecten dos puntos en el espacio. La cadena que forman luego puede utilizarse como guía para la navegación.
  - Agrupación y ensamblaje de objetos: Permite que los robots puedan manipular objetos distribuidos espacialmente y es esencial para procesos de construcción.

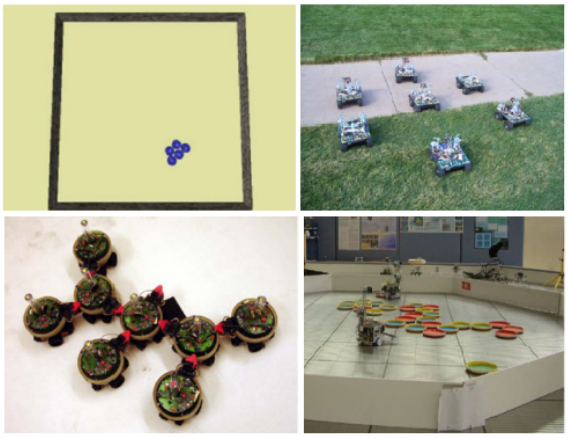


Figura 4: Ejemplos de comportamientos de navegación [12].

- Navegación: Son los comportamientos que permiten afrontar el problema de la coordinación de los movimientos para un enjambre de robots.
    - Exploración colectiva: El enjambre de robots navega de manera cooperativa a través del entorno con el objetivo de explorarlo. Es utilizado para obtener una visión general de una situación, buscar objetos, supervisar el entorno o establecer una red de comunicación.
    - Movimiento coordinado: Desplaza al enjambre de robots en una formación, donde se puede tener una forma bien definida, como una línea, o ser arbitraria.
    - Transporte colectivo: Permite al enjambre de robots mover colectivamente objetos que son demasiado pesados o grandes para un solo robot.
    - Localización colectiva: Permite al enjambre que encuentren su posición y orientación relativa entre sí mediante el establecimiento de un sistema de coordenadas local.

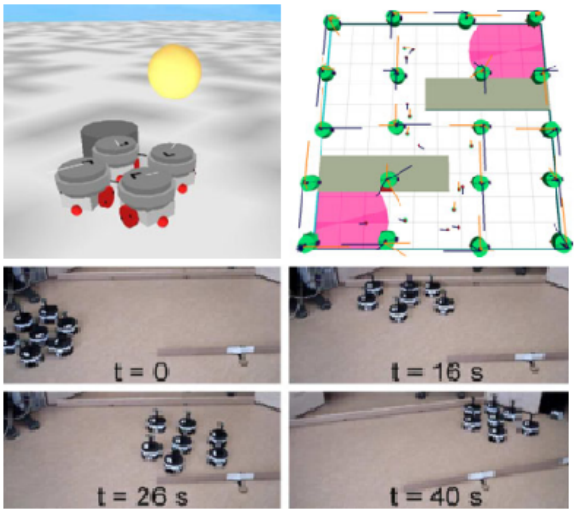


Figura 5: Ejemplos de comportamientos de organización espacial [12].

- Toma de decisiones colectiva: Se ocupa de como los robots se influyen entre sí para la toma de una decisión, donde se utiliza para responder las necesidades de ponerse de acuerdo y con la especialización.

    - Consenso: Permite que los robots individuales en el enjambre lleguen a un acuerdo o converjan hacia una única elección común entre varias alternativas.
    - Asignación de tareas: Comportamiento en donde los robots se distribuyen dinámicamente entre sí para realizar distintas tareas. El objetivo es maximizar el rendimiento de todo el sistema del enjambre.
    - Detección colectiva de fallas: Determinación autónoma por parte del enjambre de las deficiencias de robots individuales. Permite identificar robots que se desvían del comportamiento deseado del enjambre basado en la emisión de una señal síncrona.
    - Percepción colectiva: Se combinan todos los datos detectados localmente por los robots en el enjambre en una imagen global. Permite al enjambre tomar decisiones colectivas de manera informada, como puede ser la clasificación de objetos de manera confiable.
    - Regulación de tamaño del grupo: Permite que los robots en el enjambre se organicen en grupos del tamaño deseado, donde si se excede el tamaño, se dividen en múltiples grupos.

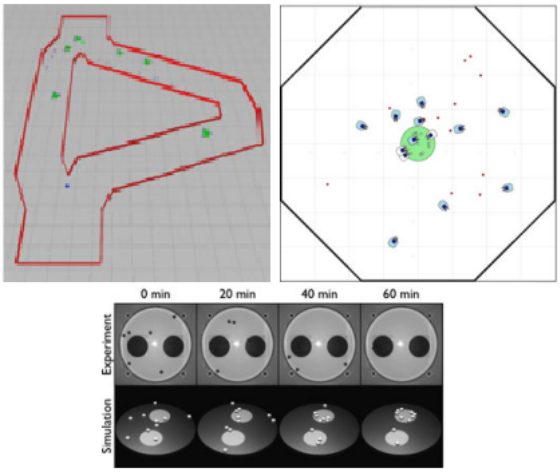


Figura 6: Ejemplos de comportamientos de toma de decisiones colectiva [12].

- Otros comportamientos colectivos

    - Auto-reparación: Permite que el enjambre se recupere de fallas causadas por deficiencias en algún robot. El objetivo es minimizar el impacto de las fallas de los robots en el resto del enjambre.
    - Auto-reproducción: Permite que un enjambre de robots cree nuevos robots o replique un patrón creado a partir de varios individuos. La principal idea es aumentar la autonomía del enjambre al eliminar la necesidad de un ingeniero humano para crear nuevos robots.

- Interacción humano-enjambre: Permite que los humanos controlen los robots en el enjambre o reciban información de ellos. La interacción puede ser remota o de forma próxima en un entorno compartido.

6.2. *Particle Swarm Optimization* (PSO)

El algoritmo original fue introducido en 1995 por James Kennedy y Russell Eberhart [13]. Esta técnica utiliza un mecanismo simple que imita el comportamiento de enjambre de algunos grupos de animales, como lo son las parvadas y los cardúmenes, para guiar a las partículas a buscar soluciones óptimas globales [14].

En PSO, las partículas se colocan en un espacio de búsqueda de un problema o función, y cada entidad evalúa una función objetivo en su ubicación actual. Luego, cada partícula determina su posición en el espacio de búsqueda combinando algún aspecto de su historial de ubicaciones actuales y las mejores ubicaciones con las de uno o más miembros del enjambre, incluyendo algunas perturbaciones aleatorias. La siguiente iteración se produce después de que se hayan movido todas las partículas. En su última instancia, es probable que el enjambre se acerque al valor óptimo de la función ajustada [15].

6.2.1. Estructura del algoritmo PSO

Primero se inicializa la población deseada, donde cada partícula tiene una solución actual particular dada por un vector

$$P_i = [x_1, x_2, \ldots, x_d]\,,$$

siendo d la dimensión del espacio de trabajo. Para cada partícula se evalúa la función de costo, f, o *fitness function*

$$f(P).$$

Después, cada partícula guarda en su memoria la solución con el menor costo que ha encontrado durante un número específico de iteraciones, conocida como el *local best* de cada partícula. Además, cada partícula transmite su mejor solución encontrada y se determina cuál de todas las soluciones es la mejor en el conjunto de todas las partículas, llamada *global best*. Utilizando esta información, cada partícula calcula los dos primeros factores de la ecuación de PSO:

Factor cognitivo: Es la resta vectorial entre la solución de *local best* y la solución actual.

$$P_{local} - P_i$$

Factor social: Es la resta vectorial entre la solución de *global best* y la solución actual.

$$P_{global} - P_i$$

Ya contando con ambos factores, puede estructurarse la ecuación para la actualización de la velocidad de las partículas:

$$V_{i+1} = V_i + (P_{local} - P_i) + (P_{global} - P_i)\,,$$

que se compone de los factores previamente mencionados, sumados a la velocidad actual de cada partícula. El algoritmo tiene en cuenta una variedad de factores, incluido el factor de escalamiento, que determina si las partículas tienen un área de búsqueda más amplia o se concentran en un área más estrecha, representado por los valores C_1 y C_2. El factor de uniformidad se refiere a dos números aleatorios generados en el rango de 0 a 1, denotados como r_1 y r_2.

Por otro lado, el factor de constricción φ controla la longitud de los pasos que cada partícula puede dar en cada iteración del algoritmo [16]. Este parámetro se calcula utilizando la fórmula específica:

$$\varphi = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}; \quad \phi = C_1 + C_2; \quad \phi > 4 \tag{1}$$

El factor de inercia w es una ponderación que se le da al factor de memoria de la ecuación de velocidad PSO. Según Eberhart y Shi , en su modificación para el algoritmo, existen varias ecuaciones para calcular este parámetro [17], como pueden ser:

Inercia constante:

$$0.8 < w < 1.2 \tag{2}$$

Inercia linear decreciente:

$$w = w_{\text{máx}} - (w_{\text{máx}} - w_{\text{mín}}) \frac{iter}{MAX_{iter}} \tag{3}$$

Inercia caótica:

$$\begin{gathered} cZ_i \in [0, \ldots 1] \\ Z_{i+1} = 4Z_i \left(1 - Z_i\right) \\ w = (w_{\text{máx}} - w_{\text{mín}}) \frac{MAX_{iter} - iter}{MAX_{iter}} w_{\text{mín}} Z_{i+1} \end{gathered} \tag{4}$$

Inercia aleatoria:

$$\begin{gathered} \text{rand}() \in [0, \ldots 1] \\ w = 0.5 + \frac{\text{rand}()}{2} \end{gathered} \tag{5}$$

Inercia exponencial:

$$w = w_{\text{mín}} + (w_{\text{máx}} - w_{\text{mín}})\, e^{\frac{MA-t}{10}} \tag{6}$$

Ya contando con todos los parámetros de ponderación a utilizar, la ecuación para la velocidad PSO resulta:

$$V_{i+1} = \varphi \left[\omega V_i + C_1 \rho_1 \left(P_{local} - P_i \right) + C_2 \rho_2 \left(P_{global} - P_i \right) \right], \tag{7}$$

donde en la ecuación, V_i representa la velocidad actual de la partícula y V_{i+1} representa la nueva velocidad calculada para la partícula. Después de actualizar las velocidades de todas las partículas, se procede a calcular las posiciones de cada una de ellas:

$$X_{i+1} = X_i + V_{i+1} \Delta t, \tag{8}$$

donde X_i representa la posición actual de la partícula y X_{i+1} representa la nueva posición calculada para la partícula. El término Δt representa el tiempo que lleva al algoritmo realizar cada iteración.

6.3. *Ant Colony Optimization* (ACO)

Este algoritmo se basa en el comportamiento natural de una colonia de hormigas, donde se observa como las hormigas tienen la capacidad de encontrar un camino óptimo entre el hormiguero y una fuente de alimento. A partir de este comportamiento, Marco Dorigo desarrolló un algoritmo denominado *Ant System* (SA), que luego de basarse en un método metaheurístico, en donde se ven a las hormigas como base del algoritmo ACO, que permite la solución de problemas discretos de optimización [18].

6.3.1. Funcionamiento ACO

Las colonias de hormigas proporcionan un ejemplo de estigmergia, donde el comportamiento de las hormigas individuales conduce a una inteligencia colectiva. En muchas especies de hormigas, cuando las hormigas se desplazan hacia y desde una fuente de alimento, liberan una sustancia llamada feromona en el suelo. Otras hormigas detectan la presencia de feromona y tienden a seguir los caminos con una mayor la concentración. Este mecanismo permite que las hormigas transporten eficientemente el alimento de regreso al hormiguero [18].

Deneubourg investigó exhaustivamente el comportamiento de las hormigas en cuanto a la colocación y seguimiento de feromonas, en un experimento conocido como el experimento del doble puente [19]. Inicialmente, cada hormiga elige aleatoriamente uno de los dos puentes, pero luego, debido a fluctuaciones aleatorias, uno de los puentes presenta una concentración más alta de feromonas que el otro y, por lo tanto, atrae a más hormigas. Esto a su vez lleva a una mayor cantidad de feromonas en ese puente, lo que lo hace que después de algún tiempo, toda la colonia converge en el uso del mismo puente. A partir de este experimento se desarrollo un modelo dado por:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \tag{9}$$

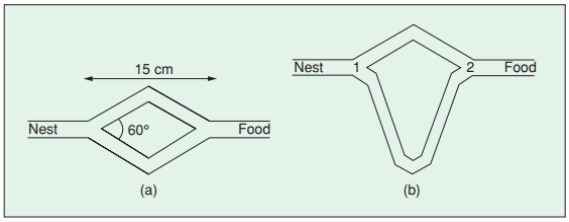


Figura 7: Configuración para el experimento del doble puente [18].

6.3.2. *Simple Ant Colony Optimization* (SACO)

Este algoritmo fue utilizado en las pruebas del experimento del doble puente, donde una hormiga artificial se encarga de navegar un grafo de construcción completamente conectado $G_C(\mathbf{V},\mathbf{E})$, donde $\mathbf{V}$ es un conjunto de vértices, nodos, y $\mathbf{E}$ es un conjunto de aristas. Las hormigas se mueven entre nodos, a lo largo de las aristas del grafo, construyendo incrementalmente una solución parcial. Además, las hormigas depositan una cierta cantidad de feromonas en los componentes, ya sea en los vértices o en las aristas que atraviesan [18].

Su principal característica es que, en cada iteración, los valores de feromona son actualizados por todas las hormigas, m que han construido una solución en la propia iteración. La feromona τ_{ij}, asociada con la arista que une los nodos i y j, se actualiza de la siguiente manera:

$$\tau_{ij} = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \tag{10}$$

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.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{si } k \in \text{aristas}(i,j) \\ 0 & \text{en otro caso} \end{cases}, \tag{11}$$

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

$$p_{ij}^{k}(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{c_{il}\in\mathbf{N}(s^p)}\tau_{il}^{\alpha}\cdot\eta_{il}^{\beta}} & \text{if } c_{ij} \in \mathbf{N}\left(s^p\right), \\ 0 & \text{en otro caso,} \end{cases} \tag{12}$$

Donde, $\mathbf{N}\left(s^p\right)$, es el conjunto de componentes factibles, es decir las aristas $(i;\ l)$ donde l son los nodos que no han sido visitado por la hormiga k. Los parámetros α y β controlan la importancia de la feromona [18].

6.4. Robots móviles

Son una clase de robots capaces de moverse a través del entorno. Existen robots capaces de moverse en el suelo, sobre el agua, bajo el agua y a través del aire. Una de las funciones más

importantes de estos robots es llegar a un punto de destino, sin importar cuál sea, tomando algún camino donde se enfrentara ante desafíos como obstáculos que puede bloquear su camino [20].

En el diseño de robots con ruedas, el equilibrio generalmente no es un problema de investigación, ya que estos robots suelen diseñarse de manera que todas las ruedas estén en contacto con el suelo en todo momento. Por lo tanto, tres ruedas son suficientes para garantizar un equilibrio estable. Cuando se utilizan más de tres ruedas, se requiere un sistema de suspensión para permitir que todas las ruedas mantengan contacto con el suelo cuando el robot encuentra con terrenos irregulares. En lugar de preocuparse por el equilibrio, la investigación en robots con ruedas tiende a centrarse en problemas como la tracción y estabilidad, maniobrabilidad y control [21].

6.5. Controladores de posición y velocidad de robots diferenciales

Para lograr describir las curvas requeridas con los robots diferenciales, se debe llevar a cabo la transformación de las velocidades lineales y angulares del robot a las velocidades angulares de cada una de las ruedas. Primero, se debe definir la transformación que se realizará para cada rueda individualmente y luego se toma en cuenta ambos actuadores para obtener la cinemática del robot. Para esto en [5] se utiliza el trabajo de [22] que describe los modelos de uniciclo y de robots diferenciales que permite lograr dicho control.

6.5.1. Modelo de uniciclo

Para iniciar el proceso de modelado completo de la cinemática de un robot diferencial, es necesario comenzar con el modelo de uniciclo. Este modelo establece la relación entre la velocidad angular de una rueda con un eje de rotación paralelo al suelo y la velocidad lineal de la rueda en relación con el suelo. Al emplear la relación convencional entre velocidades lineales y angulares, se deduce que:

$$v_R = \Phi_R r; \quad v_L = \Phi_L r \tag{13}$$

siendo v_R y v_L las velocidades lineales de las ruedas, Φ_R y Φ_L las velocidades angulares de las ruedas y r el radio de las ruedas [22].

6.5.2. Modelo diferencial

Con las ecuaciones previamente formuladas en la ecuación 13, ahora estamos listos para avanzar con el cálculo de la cinemática completa. En este contexto, el objetivo es establecer la relación entre las velocidades lineales de cada rueda del robot con la velocidad lineal del centro de masa y su velocidad angular. Se propone un modelo que se ilustra en la figura siguiente:

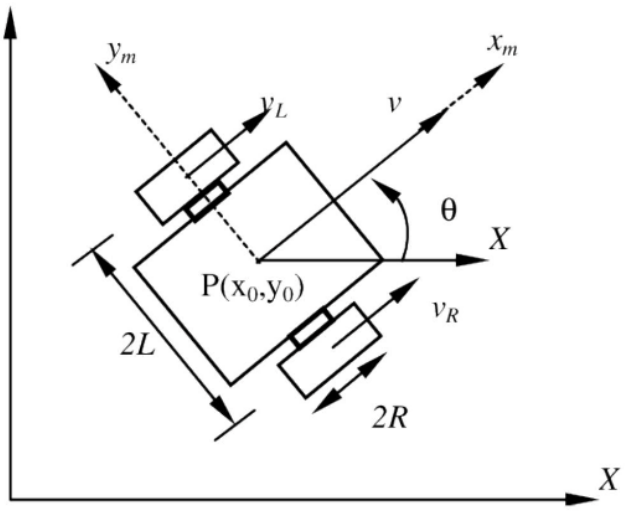


Figura 8: Modelo de robot diferencial [5].

Se nota que la velocidad lineal del centro de masa del robot diferencial es la media aritmética de las velocidades lineales de cada rueda. En consecuencia, podemos expresar esta relación mediante la siguiente ecuación:

$$v = \frac{v_R + v_L}{2} \tag{14}$$

y al combinar esta ecuación con el modelo uniciclo planteado en 13, se deriva la siguiente expresión de v en términos de Φ_R y Φ_L:

$$2v = \Phi_R r + \Phi_L r \tag{15}$$

Para determinar la velocidad angular, se emplea el concepto de centro instantáneo para vincular las velocidades lineales de cada rueda con la rotación completa del robot. Inicialmente, se posiciona el centro instantáneo en la rueda izquierda del robot, y se nota que:

$$\omega = \frac{v_R}{2l} \tag{16}$$

Luego, se coloca el centro instantáneo en la llanta derecha del robot y se observa que:

$$\omega = \frac{-v_L}{2l} \tag{17}$$

Utilizando el método de superposición, se suman las ecuaciones 16 y 17, lo que conduce a la obtención de la relación entre la velocidad angular del robot y las velocidades lineales de las ruedas como sigue:

$$\omega = \frac{v_R - v_L}{2l} \tag{18}$$

Luego se reescribe en términos del modelo uniciclo de la ecuación 13:

$$2\omega l = \Phi_R r - \Phi_L r \tag{19}$$

Para expresar las velocidades angulares de las ruedas en función de las variables de v y ω, primero se suman las dos ecuaciones que describen las velocidades lineales y angulares. El resultado de esta suma es:

$$\Phi_R = \frac{v + \omega l}{r} \tag{20}$$

Con el fin de formular la ecuación que describe la velocidad angular de la rueda izquierda, se realiza la resta de las mismas ecuaciones. El resultado de esta resta es:

$$\Phi_L = \frac{v - \omega l}{r} \tag{21}$$

Con estas ecuaciones establecidas, es posible enviar directamente el dato de velocidad angular a los actuadores de las ruedas. Para describir una curva de movimiento específica, se determinan las velocidades lineal y angular necesarias del robot con el fin de alcanzar las velocidades deseadas en cada eje plano y en el eje de rotación:

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

y por medio de las transformaciones establecidas con el modelo diferencial en 20 y 21 se calculan las velocidad para cada motor.

6.6. Controladores de posición y velocidad de robots diferenciales

Es fundamental controlar tanto la posición como la velocidad de los robots para asegurar que alcancen la configuración deseada. Además, se busca que las trayectorias seguidas por los robots sean suaves y controladas. Para una comprensión más detallada de este tema, se sugiere consultar el capítulo 6 de [5] y explorar las fuentes citadas en dicho capítulo.

6.6.1. Control PID de velocidad lineal y angular

En el ámbito de sistemas de control, uno de los controladores más ampliamente utilizados es el controlador PID. En este tipo de controlador, las constantes K_P, K_I, K_D se determinan empíricamente con el objetivo de compensar el error en estado estable y otros parámetros de rendimiento como t_p, t_s, M_p. La ecuación en el dominio del tiempo del control PID en función del error $e(t)$ es la siguiente:

$$\text{PID}(e(t)) = K_p \cdot e(t) + K_I \int_0^t e(\tau) \cdot d\tau + K_d \cdot \frac{de(t)}{dt} \tag{22}$$

Por lo tanto, la implementación se resume como:

$$\begin{aligned} w &= PID\left(e_o\right) \\ v &= PID\left(e_p\right) \end{aligned} \tag{23}$$

Donde e_o es el error de orientación y e_p es el error de posición. Estos pueden calcularse por medio de:

$$\begin{aligned} e_o &= \operatorname{atan} 2\left(\frac{\sin\left(\theta_g - \theta\right)}{\cos\left(\theta_g - \theta\right)}\right) \\ e_p &= \sqrt{\left(x_g - x\right)^2 + \left(y_g - y\right)^2} \end{aligned} \tag{24}$$

Donde θ_g es el ángulo calculado desde el punto actual hasta la meta y θ es el ángulo actual.

6.6.2. Control PID de acercamiento exponencial

En este controlador se corrige el comportamiento en espiral que se presenta en un controlador PID ordinario. En este contexto, es posible emplear parámetros k_x y k_y que dependan del error de posición del robot con respecto a su objetivo. El objetivo es lograr que la velocidad de convergencia hacia la meta disminuya a medida que el robot se acerca más a su objetivo, con el fin de evitar superar accidentalmente la posición deseada y provocar un comportamiento oscilatorio al alcanzarla.en función del error de posición del robot respecto a su meta. Se busca que la velocidad de convergencia hacia la meta decrezca mientras el robot se aproxima más a la meta para no sobrepasarla accidentalmente y causar un comportamiento oscilatorio al alcanzarla.

$$w = \text{PID}\left(e_o\right) \tag{25}$$

$$K_p = \frac{v_0 \cdot \left(1 - e^{-\alpha \|e_p\|^2}\right)}{\|e_p\|}; \quad \alpha, v_0 \quad \epsilon \quad \mathbb{R}_{>0} \tag{26}$$

Donde v_o es la velocidad máxima del robot y α es un parámetro de ajuste arbitrario.

6.7. Pololu 3Pi+ 32U4

El Pololu 3Pi+ es un una plataforma móvil versátil, de alto rendimiento y programable por un usuario. El robot cuenta con un microcontrolador ATmega32UA AVR de Microchip,

una interfaz de USB y viene precargado con un arranque compatible con Arduino. El 3pi+ incluye dos drivers puentes H y una variedad de sensores integrados, incluyendo un par de codificadores de cuadratura para control de motor de bucle cerrado, una unidad de medida inercial completa, cinco sensores de reflectancia orientados hacia abajo para seguimiento de línea o detección de bordes, y sensores de impacto izquierdo y derecho a lo largo de la cara frontal del robot. Tres botones integrados ofrecen una interfaz conveniente para la entrada del usuario y una pantalla OLED gráfica de 128 × 64, un zumbador y LEDs indicadores que permiten que el robot proporcione retroalimentación [23]. Los 3pi+ funcionan con cuatro baterías AAA, ya sean alcalinas o recargables de NiMH, que son las recomendadas.

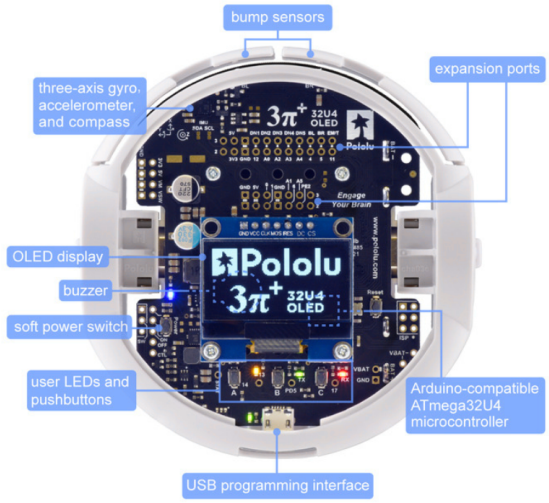


Figura 9: Estructura y ubicación de los componentes en Pololu 3pi+ [23].

6.8. OptiTrack

La plataforma de pruebas del Robotat utiliza seis cámaras Prime^x 41 de OptiTrack. Este es un sistema de captura de movimiento óptico altamente preciso con una resolución de captura de hasta 4.1 megapíxeles y una velocidad de captura de hasta 180 cuadros por segundo. La cámara Prime^x 41 ofrece una calidad excepcional en la captura de movimiento. debido al sensor CMOS y diseño óptico que le permiten una gran precisión para la captura de movimientos.

El sistema de seguimiento óptico de OptiTrack utiliza marcadores activos o pasivos que se colocan en los objetos o sujetos que se desean rastrear. La cámara Prime^x 41 detecta y registra en tiempo real la posición y la orientación de estos marcadores, lo que permite un análisis detallado y preciso del movimiento.

Motive es el software de captura de movimiento de OptiTrack, diseñado para aprovechar al máximo los datos obtenidos de los sistemas de seguimiento óptico. El software permite la configuración y calibración precisa de los sistemas de captura de movimiento de OptiTrack, asegurando una captura de datos precisa y confiable. Permite la sincronización de múltiples

cámaras y otros dispositivos de captura para lograr un seguimiento y una grabación de movimiento precisos y coordinados. Motive procesa los datos de las cámaras de OptiTrack para proporcionar posiciones globales en tres dimensiones, identificadores de marcadores e información de rotación con respecto al sistema de referencia local del objeto.

Figura 10: Cámara Prime^{x} 41 [24].

6.9. Protocolos de comunicación

El Protocolo de Datagramas de Usuario (UDP) y el Protocolo de Control de Transmisión (TCP) son protocolos de enrutamiento de la capa de transporte que se consideran parte de los protocolos principales del conjunto de protocolos de Internet. Estos protocolos manejan los datos de forma diferente.

TCP utiliza un enfoque orientado a la conexión, lo que significa que establece una conexión entre el emisor y el receptor antes de transmitir datos. Se busca proporcionar una forma confiable de enviar mensajes o información, ya que garantiza la entrega de los paquetes. Si se produce algún error durante la transmisión, el protocolo reenvía automáticamente los paquetes perdidos o dañados a través de la red.

Por otro lado, UDP utiliza un modelo de transporte más simple y directo. No establece una conexión previa entre el emisor y el receptor, lo que lo hace más liviano. Las aplicaciones informáticas que utilizan UDP pueden transmitir mensajes en forma de datagramas, que son paquetes independientes. Esto hace que UDP sea adecuado para aplicaciones que requieren una transmisión rápida de datos, como transmisiones en tiempo real de voz y video [25].

Una de las principales diferencias es que TCP ofrece una entrega ordenada y fiable de los datos desde el usuario hasta el servidor y viceversa, mientras que UDP se considera un protocolo sin conexión y no proporciona una entrega fiable de los datos. Además, TCP se destaca por su mayor fiabilidad en comparación con UDP. Esto se debe a que TCP utiliza retransmisiones y confirmaciones de mensajes en caso de pérdida de paquetes, lo que garantiza que no se pierdan datos. En cambio, UDP no garantiza que los datos hayan llegado al receptor y no ofrece retransmisiones, tiempos de espera ni confirmaciones de mensajes. [25].

6.10. Convención de ejes delimitados por la Tierra

Esta serie de sistemas de coordenadas son una convención de ejes acotados a la Tierra que a menudo se utiliza en algunas aplicaciones geodésicas, topográficas, en determinadas aplicaciones aeroespaciales y en la dinámica de vuelo para describir posiciones y orientaciones de aeronaves, naves espaciales y drones en relación con la superficie terrestre. Alguno se los sistemas que más se utilizan son [26]:

6.10.1. Sistema de Coordenadas Norte-Arriba-Este (NUE)

- x-Norte (Norte): El eje x positivo apunta hacia el norte a lo largo del meridiano de longitud.
- y-Arriba (Arriba): El eje y positivo apunta hacia arriba, perpendicular a la superficie de la Tierra.
- y-Este (Este): El eje z positivo apunta hacia el este a lo largo del paralelo de latitud.

6.10.2. Sistema de Coordenadas Este-Norte-Arriba (ENU)

- x-Este (Este): El eje x positivo apunta hacia el este a lo largo del paralelo de latitud.
- y-Norte (Norte): El eje y positivo apunta hacia el norte a lo largo del meridiano de longitud.
- z-Arriba (Arriba): El eje z positivo apunta hacia arriba, perpendicular a la superficie de la Tierra.

6.10.3. Sistema de Coordenadas Norte-Este-Abajo (NED)

- x-Norte (Norte): El eje x positivo apunta hacia el norte a lo largo del meridiano de longitud.
- y-Este (Este): El eje y positivo apunta hacia el este a lo largo del paralelo de latitud.
- z-Abajo (Abajo): El eje z positivo apunta hacia abajo, perpendicular a la superficie de la Tierra.

6.11. Webots

Webots es un simulador de robots 3D gratuito y de código abierto elaborado por Cyberbotics para uso de forma profesional en la industria, la educación y la investigación. Proporciona un entorno de desarrollo completo para modelado, programación y simulación de robots. Es una herramienta que facilita la creación de simulaciones completas de robots

utilizando su extensa biblioteca de recursos, que incluye robots, sensores, actuadores, objetos y materiales. Le permite importar modelos CAD existentes desde Blender o URDF, así como importar mapas desde OpenStreetMap.

Su interfaz gráfica facilita la edición de simulaciones y controladores de robots. Permite ahorrar tiempo al desarrollar proyectos de robótica y pueden usarse en una variedad de aplicaciones, desde robots de dos ruedas hasta armas industriales, drones voladores y vehículos submarinos autónomos. También se puede utilizar para crear entornos interiores y exteriores interactivos, construir prototipos de robots, diseñar, probar y validar inteligencia artificial y algoritmos de control, y enseñar a los estudiantes sobre robots [27].

6.11.1. Cambios importantes en la versión R2022a

A partir de la versión R2022a, todas las geometrías, entidades y prototipos incluidos con Webots utilizan la orientación del eje FLU (x hacia adelante, y hacia la izquierda y z hacia arriba). Esta actualización fue introducida para hacer las simulaciones compatibles con ROS y otros sistemas robóticos. Además, se cambió desde el eje de coordenadas NUE hacía el eje de coordenada ENU.

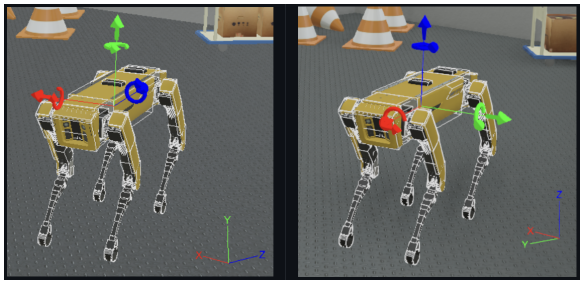


Figura 11: Cambios de la versión R2022a de Webots [28].

6.12. MATLAB

MATLAB es un entorno de programación de lenguaje avanzado para computación numérica, visualización de datos y programación de algoritmos. Es desarrollado y comercializado por MathWorks. MATLAB es un lenguaje de programación matricial, lo que significa que los datos se almacenan y manipulan en matrices. Esto lo hace ideal para la computación numérica, porque las matrices se pueden usar para representar cantidades físicas como vectores, matrices y tensores. Tiene una amplia gama de capacidades de visualización de datos, lo que lo convierte en una herramienta valiosa para el análisis de datos. Estas funciones le permiten crear gráficos, tablas y otros tipos de visualizaciones que ayudan a los usuarios a comprender sus datos [29].

6.12.1. Funciones

La vectorización y la paralelización son dos técnicas de optimización que se pueden utilizar para mejorar el rendimiento de los programas MATLAB.

La vectorización es el proceso de convertir un código secuencial en código que puede ser ejecutado por un procesador vectorial. Los procesadores vectoriales tienen unidades de ejecución especializadas que pueden procesar datos en paralelo. MATLAB proporciona una serie de funciones vectorizadas que pueden utilizarse para mejorar el rendimiento de los programas. Vectorizar el código permite mejorar la apariencia de las expresiones, lo que hace que el código sea más fácil de entender. Además, el vectorizar reduce el tiempo de ejecución del código, algo que puede ser de gran utilidad cuando se tienen programas extensos [30].

La paralelización es el proceso de dividir un problema en subproblemas que pueden ser ejecutados de forma independiente. Esto se puede hacer para aprovechar los recursos de múltiples procesadores o núcleos de procesador [31].

CAPÍTULO 7

Migración de algoritmos y ajustes para pruebas físicas

Para poder efectuar pruebas físicas con las plataformas móviles Pololu 3pi+ era necesario realizar la migración de los algoritmos desde su entorno de simulación original. Para esto, primero se realizó una migración parcial para poder recrear las simulaciones y mejorar el entendimiento de los algoritmos, así como analizar su desempeño. Luego se llevó a cabo una serie de ajustes que se presentan a lo largo de este capitulo, dentro de los que se incluyen cambios a los algoritmos para que permitan la comunicación con los agentes móviles y la plataforma de pruebas, incluyendo el sistema de captura de movimiento.

7.1. Generalidades de migración

A pesar de sus diferencias en ejecución y desarrollo, los algoritmos de MPSO y ACO comparten algunas similitudes en cuanto a la comunicación y obtención de datos. Estas similitudes permiten que ambos algoritmos puedan ser implementados de forma similar en algunos parámetros.

7.1.1. Actualización de sistema de coordenadas

Los algoritmos desarrollados en etapas anteriores para la implementación de controladores que permiten seguir trayectorias fueron diseñados para trabajar en el sistema de coordenadas NUE. Este era el sistema de coordenadas predeterminado en Webots y que permitía realizar las simulaciones en este entorno.

A partir de la versión R2022a de este entorno de simulación, el sistema de coordenadas y su respectiva orientación cambió al sistema ENU-FLU [28]. Este cambio de sistema de coordenadas requirió modificaciones en los algoritmos y en los mundos para simulación

dentro de Webots. Este nuevo ajuste se debió a que si se utilizaban los mundos de simulación de fases previas de formas directa, estos presentaban orientaciones y posiciones no deseadas, como puede observarse en la Figura 12. Para la recreación de las simulaciones en Webots se optó por la versión R2023a, la versión más reciente a inicios de la elaboración de este trabajo y con la que ya se tenía experiencia en otras aplicaciones.

En el caso de los mundos, se tuvo que modificar la orientación de los robots y demás componentes para que coincidiera con la orientación ENU que se utiliza en la versión más reciente del *software*. Se tomó en cuenta que para el sistema de coordenadas anterior el plano horizontal, en el cual hacen los recorridos los robots, estaba dado por los ejes Z y X y que ahora está delimitado por los ejes X y Y. Además, se decidió hacer el cambio hacia el uso de una herramienta especial para robots llamada *Supervisor*, que simula de mejor forma un entorno de captura de movimiento, en vez de utilizar sensores internos como *GPS* y *Compass*. El resultado del proceso descrito anteriormente puede observarse en la Figura 13.

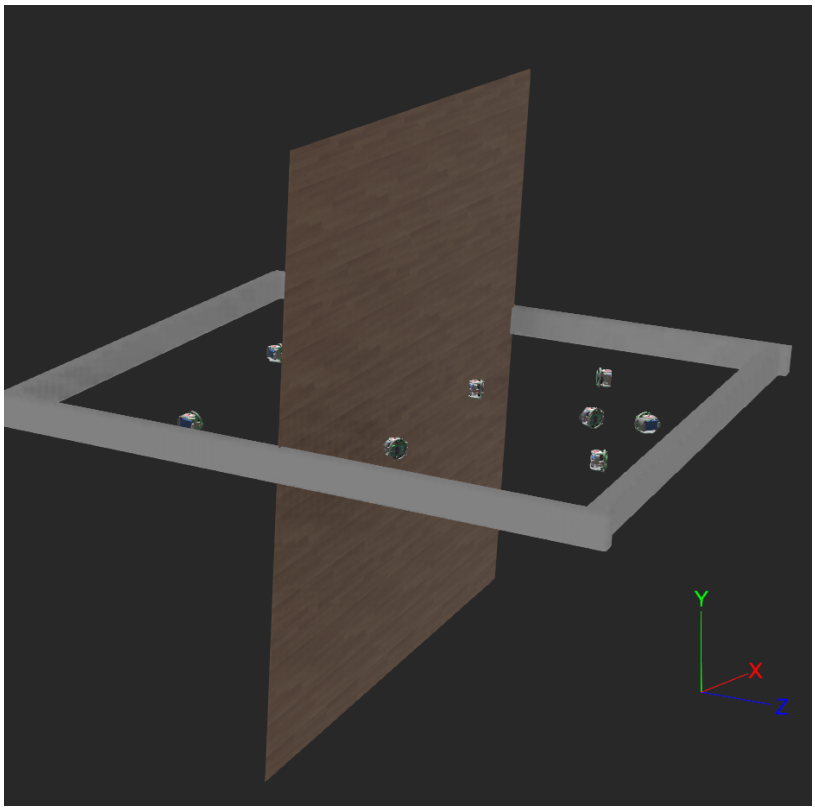


Figura 12: Objetos en Webots R2023a con sistema de coordenadas NUE.

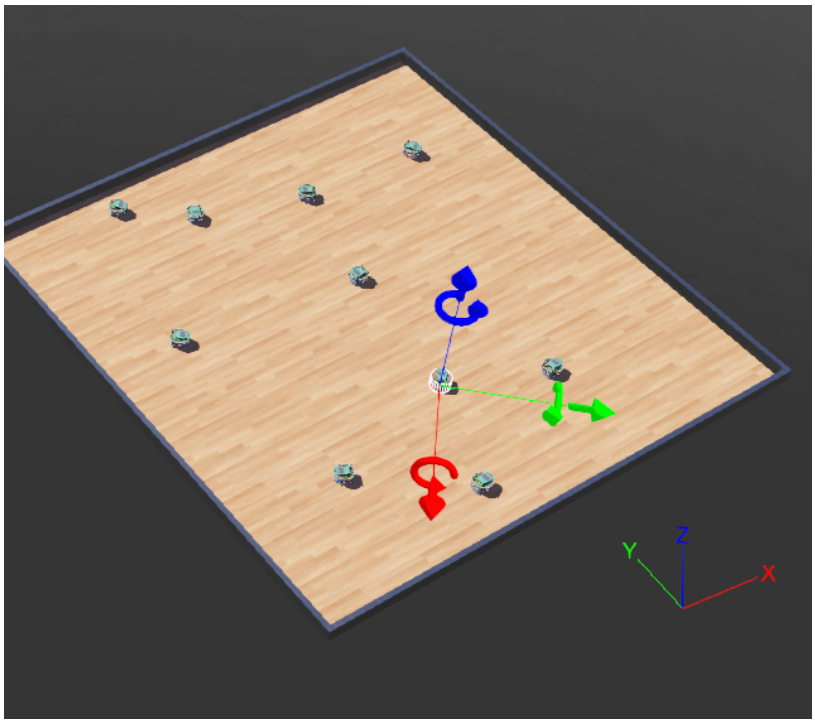


Figura 13: Objetos en Webots R2023a con sistema de coordenadas ENU.

El cambio del eje de coordenadas, además de posibilitar la recreación de las simulaciones anteriores, nos permite poder ajustar para la utilización de la mesa de pruebas del Robotat. El sistema de captura de movimiento emplea el plano horizontal con los ejes X y Y y la dirección hacia arriba de la mesa como el eje Z positivo.

Referente a la sección de los algoritmos que emplean coordenadas del mapa, fue necesario realizar el ajuste para que se tomaran los datos del plano horizontal, correspondiente a la superficie de trabajo, como los parámetros para la generación de trayectorias y para establecer la ubicación destino. De igual forma, fue necesario ajustar para que los algoritmos se orientaran correctamente hacia la dirección frontal de cada robot y permitir que sigan las trayectorias generadas de forma satisfactoria.

Un cambio relevante para el funcionamiento de los controladores fue el ajuste de los parámetros físicos de los pololu 3pi+, donde se tuvo que incluir valores para el radio de las ruedas, distancia entre ruedas radio del robot y velocidad máxima de las ruedas. Todos estos valores fueron obtenidos de los planos oficiales para las plataformas robóticas Pololu 3pi+.

Propiedad	**Valor**
Velocidad máxima de las ruedas	800 rpm
Radio de las ruedas	16.0 mm
Distancia entre ruedas y centro	48.0 mm
Velocidad máxima permitida	60 rpm

Cuadro 1: Propiedades físicas y ajustes del Pololu 3pi+.

7.1.2. Comunicación con sistema de captura de movimiento OptiTrack y robots Pololu 3pi+

Dentro de las simulaciones realizadas en las fases anteriores, las posiciones y orientaciones se obtenían por medio de sensores internos de los robots utilizados, los e-puck, algo con lo que los robots móviles Pololu 3pi+ no cuentan. Debido a esto fue necesario ajustar la forma de adquirir dichos parámetros, dando uso al sistema de captura de movimiento OptiTrack y los marcadores disponibles para cada robot.

Para poder establecer una comunicación para la recolección y envío de datos, ya sea hacia el servidor de OptiTrack o con los robots, es necesario establecer la estructura con la que se trabajan los Pololu 3pi+. Los robots cuentan con una microcontrolador ATmega32U4 que se encarga de controlar internamente las velocidades de ambos motores, además de algunos sensores internos. La placa cuenta con unos puertos de expansión, los cuales se utilizan para agregar un microcontrolador de la familia ESP32 que permite la conexión con la red de la mesa de pruebas Robotat. El ESP32 es el encargado de recibir los comandos para conectar con los robots deseados y los parámetros de velocidades de sus respectivos motores.

Ahora bien, el dispositivo encargado de establecer la posición y orientación es un marcador colocado en la parte superior de cada robot. Estos marcadores son detectados por las cámaras del sistema OptiTrack y dan la información necesaria para crear cuerpos rígidos que el sistema puede transmitir como coordenadas. Toda la estructura de los robots móviles para aplicaciones físicas puede observarse de forma clara en la Figura 14. Toda la comunicación esta dada por la librería Robotat, la cuál brinda las funciones necesarias para las operaciones con los marcadores y las plataformas móviles [32].

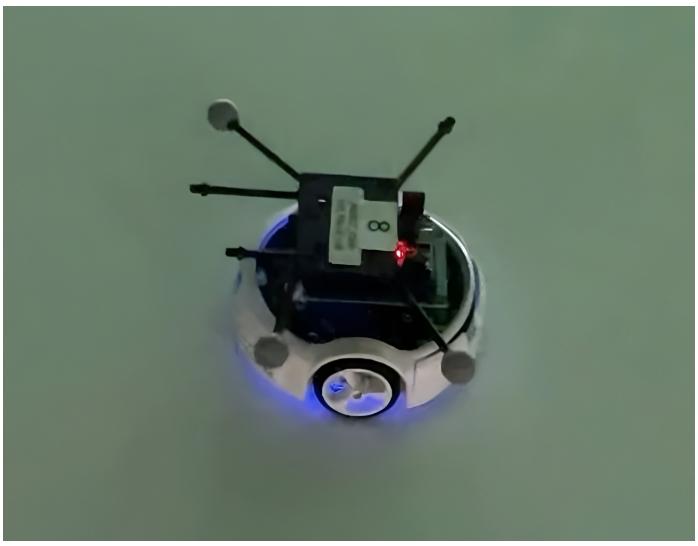


Figura 14: Pololu 3pi+ con ESP32 y marcador.

Las orientaciones de los cuerpos rígidos que crea cada marcador dentro del software del OptiTrack varían dependiendo del número de marcador. La solución fue encontrar un ángulo de desfase que añadir a cada marcador, logrando que todos los robots se orienten de la misma forma y tomando en cuenta la referencia de los ejes en la plataforma. Para conocer

los valores de desfase se obtuvieron los valores de rotación del eje Z cuando los marcadores estaban colocados con su posición frontal hacia el eje Y positivo, como puede notarse en la Figura 15.

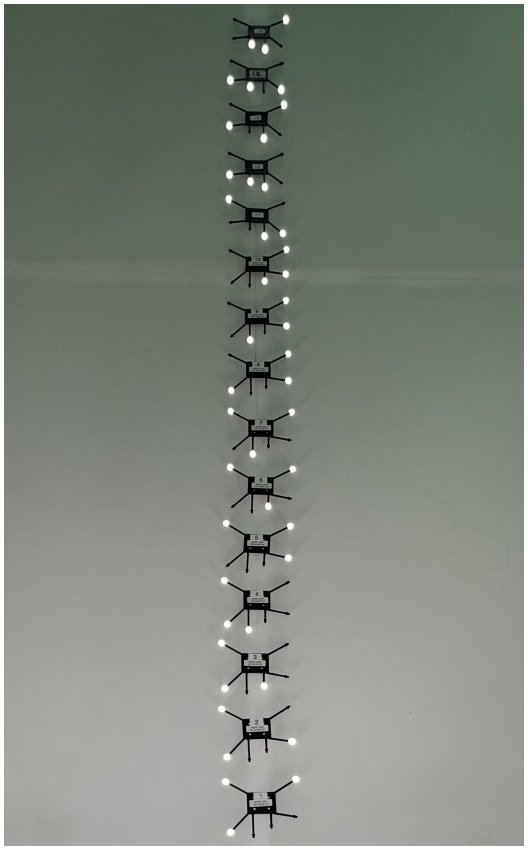


Figura 15: Posicionamiento para obtención de ángulos de desfase con respecto a mesa de pruebas.

No. de marcador	**Ángulo de desfase (°)**
1	92.0104
2	-46.8416
3	-92.4441
4	-138.1977
5	176.3751
6	-144.2067
7	-176.2966
8	-79.9578
9	-9.9146
10	139.4099

Cuadro 2: Ángulos de desfase aplicado a cada marcador.

Al realizar las primera pruebas con los Pololu 3pi+ se pudo observar que este ángulo no era suficiente para que se orientaran de forma correcta para las trayectorias que se estaban ejecutando. Luego de una verificación, fue necesario añadir un desfase de 90° a cada marcador, esto dado que los robots E-Puck que se utilizan en simulación trabajan con su

norte siendo el eje Y-, como se observa en la Figura . mientras que los robots Pololu 3pi+ trabajan con su eje X+ siendo el norte.

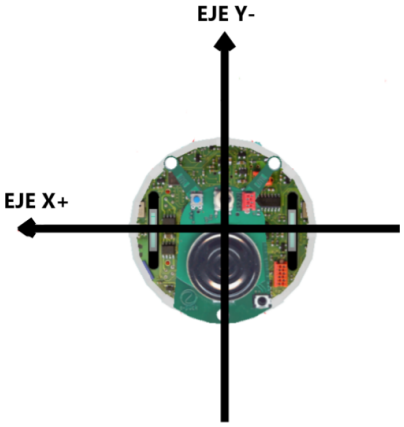


Figura 16: Ejes del robot E-Puck.

El ajuste para estos desfases fue ajustado por medio del siguiente código:

```
for b = 1:10
        bearing = abs(bearing_deg(b)) + 90;

        if (bearing > 180)
                bearing = bearing - 360;
        elseif (bearing < -180)
                bearing = bearing + 360;
        end

        offset(b) = bearing;
end
```

La letra **b** representa el número de marcador al que se le efectuará el cambio, el vector **bearing_deg** son los valores de desfases medidos que se muestran en el Cuadro 2 y **offset** es la variable que almacena los desfases actualizados para su utilización en el controlador de cada robot. Además se trabaja con el valor absoluto del desfase para normalizar el rumbo y a evitar posibles problemas relacionados con direcciones negativas. Luego se realizó un ajuste para que los robots se orienten en ángulos entre -180 y 180 grados.

7.2. Modificaciones para algoritmo MPSO

Como punto inicial en la migración de este algoritmo se pudo observar que estaba elaborado en lenguaje de programación C. Con la finalidad de poder utilizar las funciones

de la librería Robotat mencionada con anterioridad, que están desarrolladas en lenguaje de MATLAB, fue necesario realizar el cambio hacia este lenguaje de programación. Se tuvo que ajustar las funciones y variables necesarias para poder ejecutar el código desde la versión de MATLAB R2021a.

En la simulación de Webots se utilizaron diez robots que ejecutaban el algoritmo de forma independiente. Esto nos indica que se estaban realizando pruebas con un funcionamiento de enjambre descentralizado. Para poder comunicar los agentes entre sí era necesario dar uso de dos herramientas para robots en Webots, llamadas Emisor y Receptor. Las herramientas permitían la comunicación entre cada robot, donde la transmisión de datos se realizaba por medio del emisor y la recepción de datos por medio del receptor. Los robots enviaban y recibían de forma constantes sus posiciones para poder ser tomados en cuenta dentro del algoritmo de PSO, siendo factor importante para el funcionamiento correcto de la localización del *global best*. Se decidió agregar un margen extra entre los bordes del espacio de trabajo y el área que el algoritmo utilizará para encontrar la mejor solución. Esto se hizo para evitar los puntos ciegos del sistema de captura de movimiento.

Sin embargo, para poder implementar el algoritmo a nivel físico fue fundamental encontrar una forma de centralizar el funcionamiento. La importancia de este cambio se debe a que las plataformas móviles Pololu 3pi+ no son capaces de ejecutar el algoritmo de forma autónoma. Además, no existe actualmente una opción de dispositivos internos para efectuar la comunicación en tiempo real y en simultáneo entre los agentes. Con el soporte del sistema de captura de movimiento fue posible realizar el algoritmo de forma centralizada. Se logró utilizar el servidor para obtener las posiciones de los agentes a cada instante y estar actualizando no solamente las velocidades de los motores, sino que también poder ejecutar el algoritmo PSO para encontrar la mejor solución para el enjambre, denominado como el *global best*.

Para poder realizar la tarea de centralizado, un único dispositivo corre el algoritmo y controla las velocidades para cada robot en simultáneo. Fue necesario la introducción de ciclos *for* que permiten realizar tareas en bucle durante un número de repeticiones. Los ciclos *for* trabajan de forma secuencial, lo que para la ejecución en simultáneo de hasta diez agentes puede tomar un tiempo bastante elevado entre cada iteración. Fue importante trabajar de la mano con pruebas de ciclos *parfor* y vectorización del código que permiten una ejecución más veloz del algoritmo. La utilización de las funciones de *parallel pooling* en MATLAB permitieron la implementación de una mayor cantidad de agentes, siempre tomando en cuenta un incremento en el tiempo de ejecución del algoritmo, pero siendo más eficiente que los ciclos *for* convencionales.

El primer controlador utilizado fue el PID con acercamiento exponencial. Para poder trabajar con plataformas físicas, se ajustaron los valores del controlador de la siguiente manera:

El segundo controlador que se implementó fue un LQI (Lineal Cuadrático Integral) para el control de robots móviles en las trayectorias encontradas por los algoritmos. El controlador LQI combina las ventajas del control proporcional-integral-derivativo (PID) con la teoría de control óptimo, lo que nos permite lograr un equilibrio fino entre el seguimiento de la trayectoria deseada y la estabilidad del sistema. El Cuadro 4 presenta los ajustes realizados para la utilización de este controlador, donde se evaluó la mejor condición para las matrices

Parámetro	Valor en simulación	Valor en físico
K proporcional (Kp)	0.5	0.5
K integral (Ki)	0.1	0.01
K derivativo (Kd)	0.001	0.0001
$v0$	3.12	3.2
alpha	0.5	0.9

Cuadro 3: Parámetros de PID-MPSO en simulación vs físico

QQ y R para los pesos de estado y control, respectivamente.

$$\sigma = 0 \qquad Cr = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad Dr = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$AA = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad BB = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$QQ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2.2 & 0 \\ 0 & 0 & 0 & 0.7 \end{bmatrix} \qquad Klqi = \begin{bmatrix} 1.996 & 0 & 1.483 & 0 \\ 0 & 1.635 & 0 & 0.837 \end{bmatrix}$$

Cuadro 4: Parámetros utilizados para controlador LQI físico.

7.3. Modificaciones para algoritmo ACO

El algoritmo de ACO fue desarrollado originalmente en el lenguaje de MATLAB, algo que facilitó la migración en cuanto a la generación de la trayectoria encontrada por el algoritmo. Uno de los ajustes necesarios fue cambiar el espacio de trabajo del generador de trayectorias de la plataforma en el simulador de Webots a la mesa de pruebas del Robotat. Esto se hizo para que la trayectoria se ajustara a las dimensiones disponibles de la mesa de pruebas real, que son de 4 × 5, y no a las dimensiones de la plataforma en el simulador, que son de 2 × 2. Para evitar que las trayectorias se generen demasiado cerca de los límites de detección de las cámaras del sistema de captura de OptiTrack, se incluyó un límite perimetral a los cálculos. Este límite se ajustó a las dimensiones de la mesa de pruebas física y está dado por las variables b_x y b_y. El código para implementar lo descrito anteriormente dentro del software de MATLAB puede observarse en el siguiente recuadro:

```
b_x = 0.75;
b_y = 0.8;
ratio_x = 1/(grid_size/4);
ratio_y = 1/(grid_size/5);
bpath = [G.Nodes.X(best_path), G.Nodes.Y(best_path)];
robot_path = (bpath-grid_size/2).*[(b_x*ratio_x) (b_y*ratio_y)];
```

De igual forma que el algoritmo de PSO, se decidió utilizar controlador con acercamiento exponencial para la primera serie de pruebas con ACO. Para poder trabajar con plataformas físicas, se ajustaron los valores del controlador de la siguiente manera:

Parámetro	**Valor en simulación**	**Valor en físico**
K proporcional (Kp)	1.0	1.0
K integral (Ki)	0	0.001
K derivativo (Kd)	0.001	0
alpha	0.9	0.9

Cuadro 5: Parámetros de PID-ACO en simulación vs físico

CAPÍTULO 8

Desempeño de experimentos con Particle Swarm Optimization

Uno de los objetivos principales de este trabajo era evaluar el funcionamiento del algoritmo en físico y a la capacidad de generar trayectorias para cada agente . Para ello, se recrearon las pruebas de la fase simulada, ahora con los robots físicos, utilizando una serie de distintos puntos iniciales y finales para comprobar la robustez del algoritmo ante distintas condiciones. Para evaluar que el funcionamiento sea adecuado, en comparación con las simulaciones, se trabajó utilizando las *fitness function* seleccionadas para evaluar el rendimiento del PSO en [5].

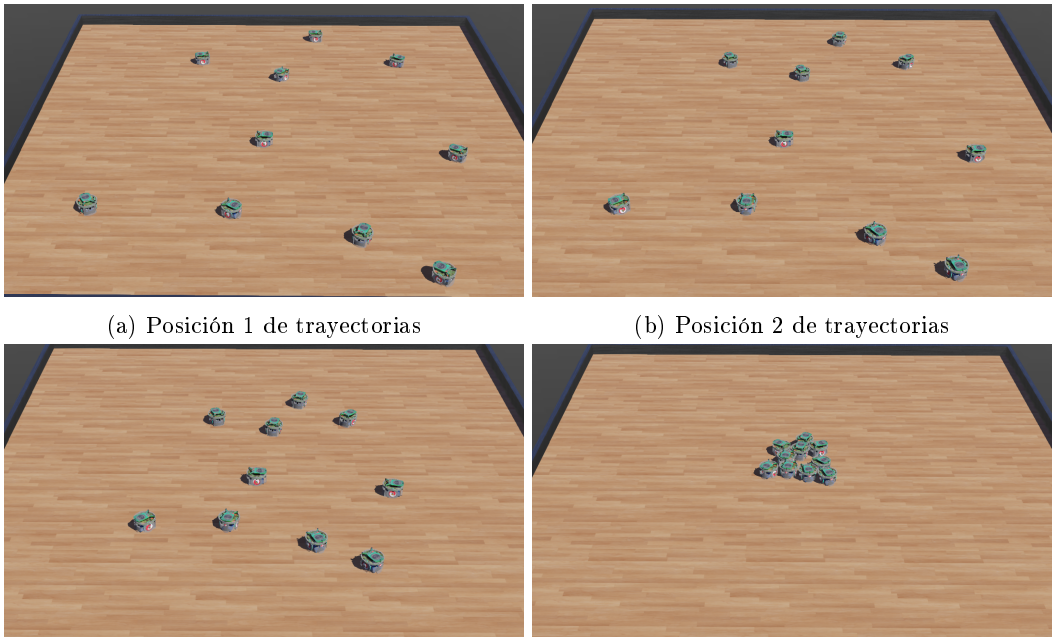
8.1. Pruebas iniciales para validar parámetros del algoritmo y controlador

Las primeras pruebas se realizaron con la finalidad de comprobar que la migración del algoritmo generara trayectorias óptimas que los agente puedan ejecutar por medio de un controlador para robots diferenciales. La prueba inicial constó con 3 agentes físicos para facilitar el seguimiento visual y asegurar su correcto funcionamiento. Los siguientes ajustes se establecieron al algoritmo para esta prueba:

La prueba fue primero evaluada a nivel de simulación con la finalidad de poder tener un punto de comparación con la prueba físicas. Para esto se utilizaron los mismo parámetros, previamente descritos en el Cuadro 6, tomando en cuenta las diferencias en los controladores y velocidades de los robots utilizados en simulación. Como puede observarse en la Figura 17, todos los robots convergen hacia una zona cercana al centro del espacio de trabajo, como se presenta en las pruebas iniciales en [5].

Descripción	Ajuste seleccionado
Fitness function	Sphere
Controlador	PID con acercamiento exponencial
Tipo de inercia	Aleatoria
Factor de constricción	0.6
Peso cognitivo	2.0
Escalador de velocidad	0.35

Cuadro 6: Parámetros de PSO para prueba 1.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 17: Trayectoria generada en simulación para prueba con parámetros del Cuadro 6.

Tomando en cuenta la naturaleza de la *fitness function* utilizada y los resultados de simulación, se tomó el marcador de otro agente y fue colocado en la ubicación aproximada donde se conoce que estaría el *global best* para este caso, con la finalidad de verificar que la comunicación entre los agentes en el algoritmo fuera la adecuada para efectuar las trayectorias hacia el punto de convergencia en cada función de costo.

La prueba tuvo un resultado satisfactorio, todos los agentes de la prueba convergieron hacia el *global best* de la función y se pudieron obtener las trayectorias tomadas por los distintos agentes para llegar al destino, como se comprueba en la Figura 18.

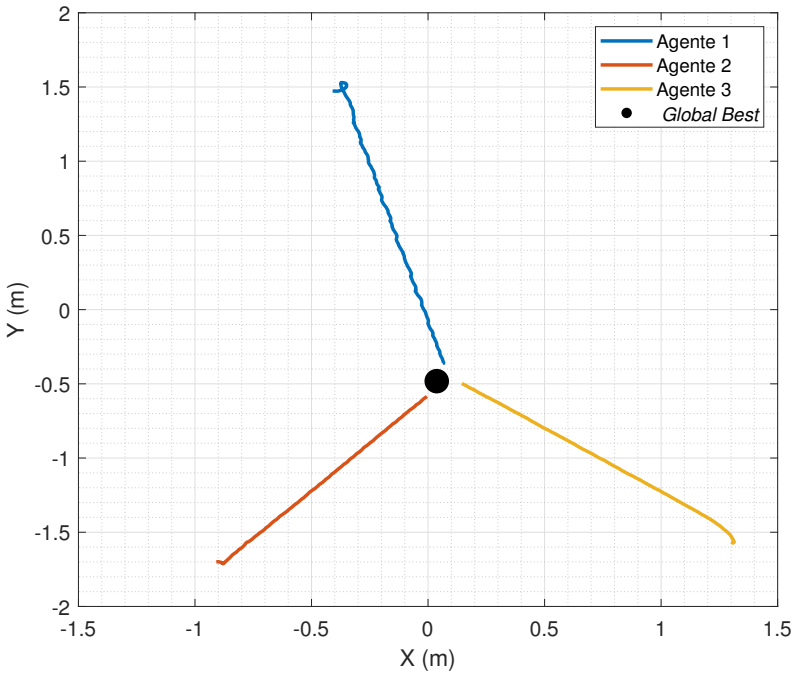
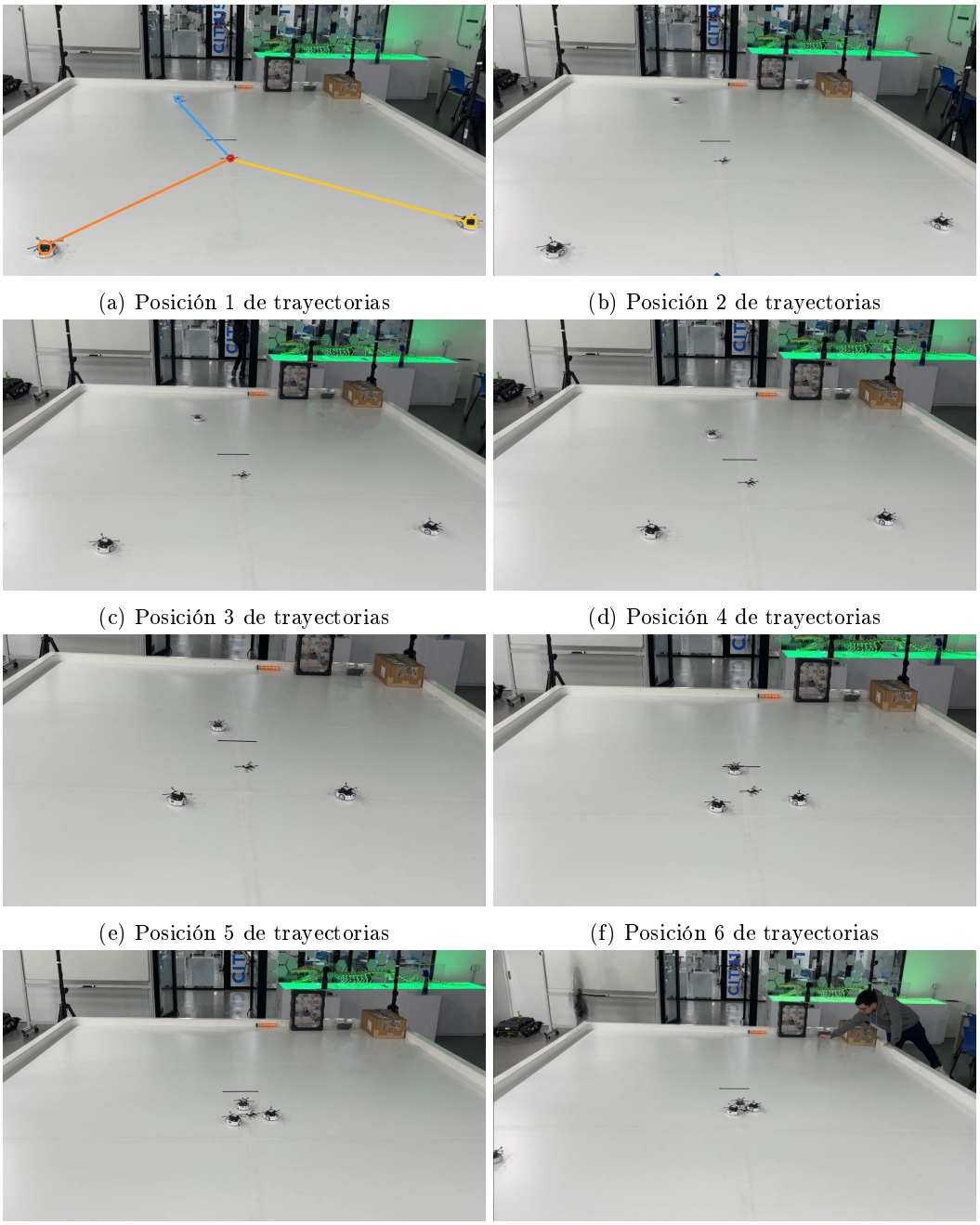


Figura 18: Trayectorias en 2D realizadas por los agentes.

En la Figura 19 se pueden observar distintos momentos durante la trayectoria de los robots hacia el punto ideal encontrado por el algoritmo, donde en 19a se marcan las trayectorias que necesitaban recorrer los agentes y el punto *global best* encontrado.

Una de las principales diferencias presentadas entre la ejecución en simulación y en físico fue el tiempo que tomaron los agentes para llegar al punto encontrado como el *global best*. Los agentes dentro de la simulación en Webots tardaron 25 segundos en converger, mientras que los agentes en la mesa del Robotat requirieron 85 segundos para llegar al punto de convergencia. Cabe resaltar que esta prueba fue realizada con el algoritmo iterando de forma lineal y con un sistema centralizado para la prueba física, lo que explica esta diferencia en el tiempo de ejecución.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

(f) Posición 6 de trayectorias

(g) Posición 7 de trayectorias

(h) Posición 8 de trayectorias

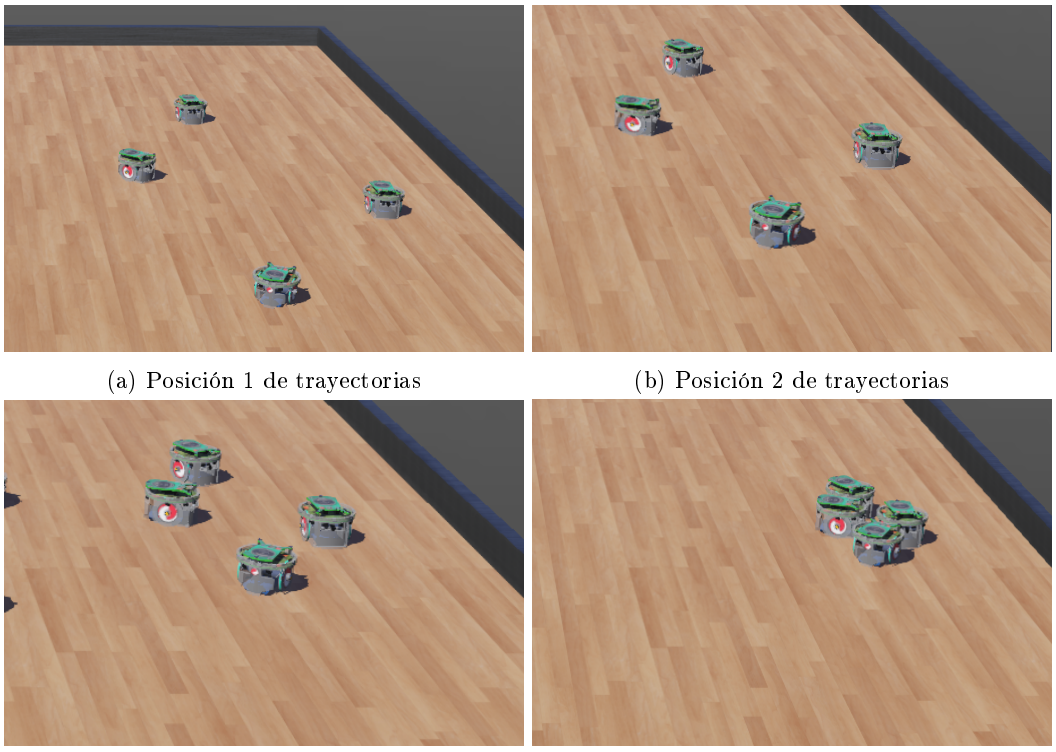
Figura 19: Trayectoria generada en físico para prueba con parámetros del Cuadro 6.

Para la segunda prueba se deseaba evaluar otra de las funciones de costo que el algoritmo puede implementar y el controlador LQI desarrollado para los robots móviles. De igual forma que en la prueba anterior, se decidió utilizar una cantidad reducida de agentes para un mejor control. Para esta segunda prueba se utilizaron los siguientes parámetros del algoritmo:

Descripción	Ajuste seleccionado
Fitness function	Schaffer No.4
Controlador	LQI
Tipo de inercia	Aleatoria
Factor de constricción	0.8
Peso cognitivo	2.0
Peso social	10
Escalador de velocidad	0.032

Cuadro 7: Parámetros de PSO para prueba 2.

Como puede observarse en las trayectorias generadas en las figuras 20 y 21, el algoritmo en físico logró converger al punto *global best* encontrado por el algoritmo durante su ejecución de forma similar que en la simulación. En las pruebas simuladas, el destino encontrado tiende a estar ubicado mucho más cerca de los bordes del espacio de trabajo que en las pruebas físicas, siendo esto debido al margen añadido a los límites de la mesa de pruebas que fue mencionado en el capítulo anterior.



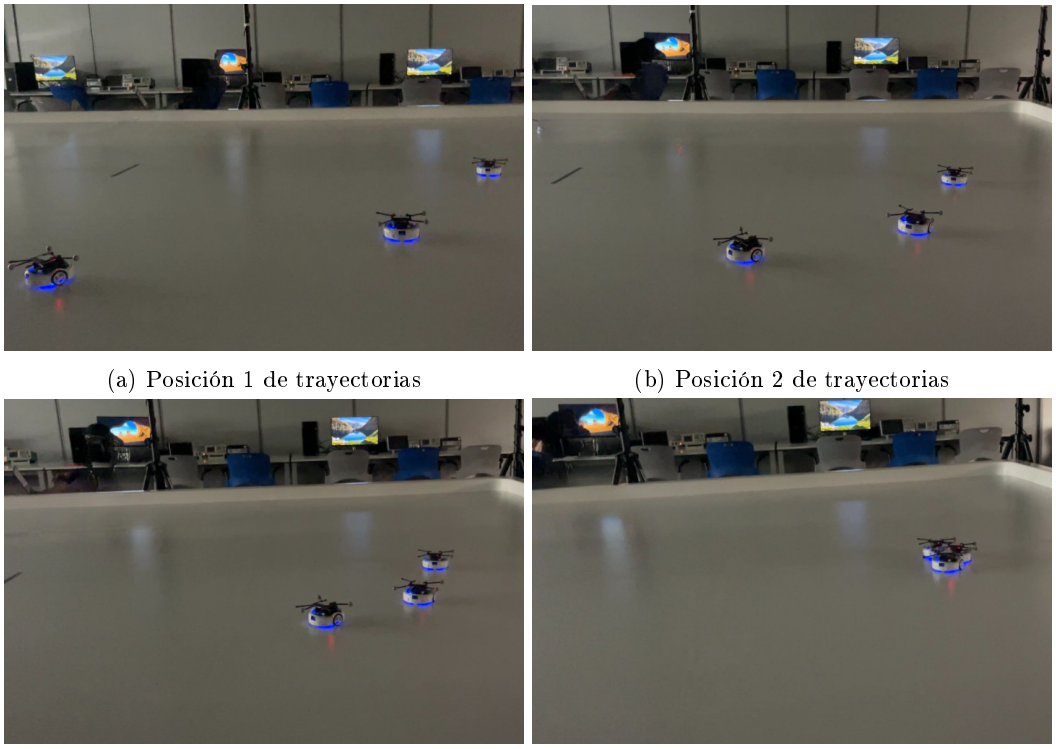
(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 20: Trayectoria generada en simulación para prueba con parámetros del Cuadro 7.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 21: Trayectoria generada en físico para prueba con parámetros del Cuadro 7.

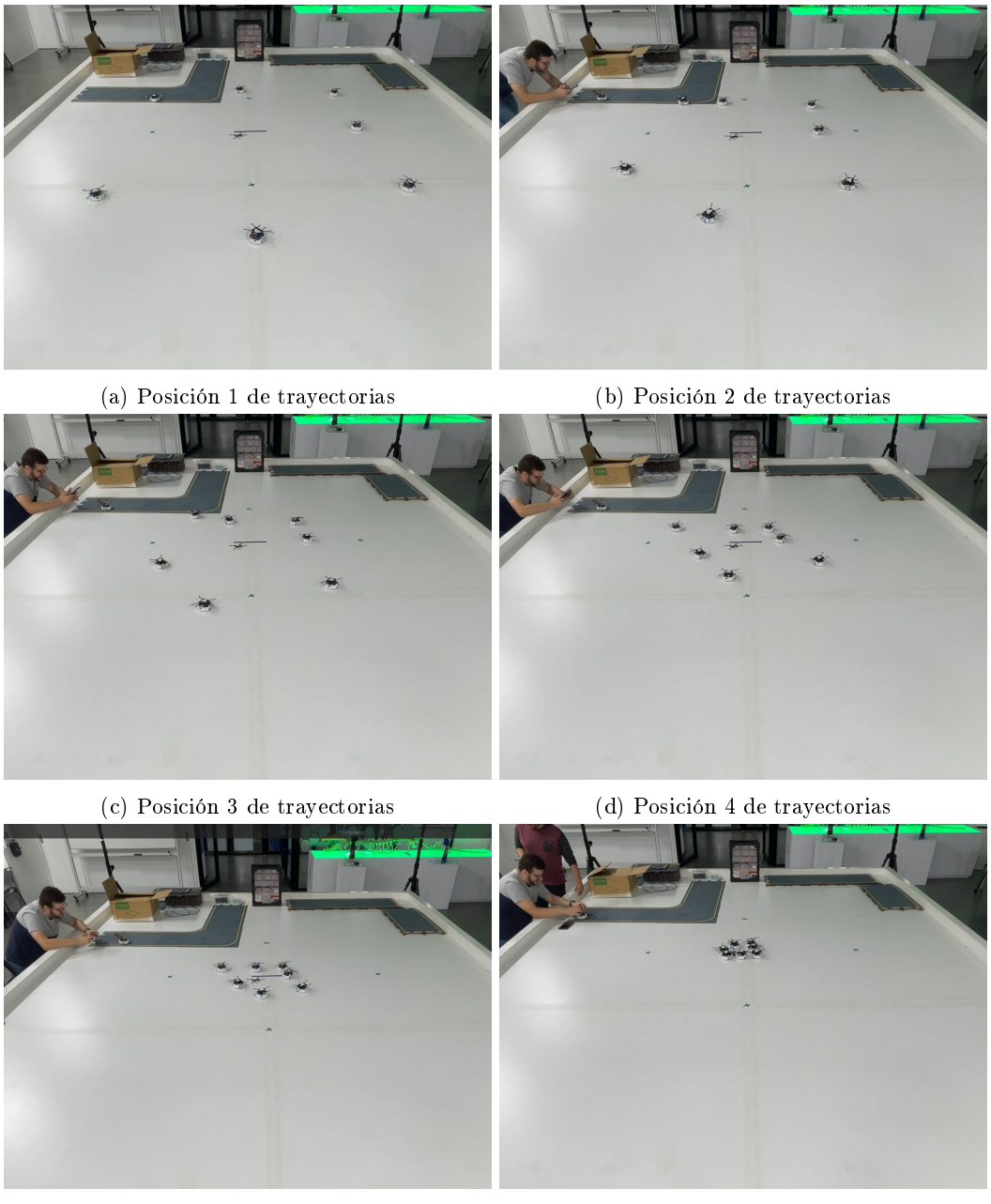
8.2. Pruebas con las distintas *fitness functions*

Luego de evaluar que las trayectorias que se estaban generando eran satisfactorias se prosiguió con pruebas con una mayor cantidad de agentes. Para simplificar la evaluación de cada función se utilizó el mismo controlador en todos los casos, ya que se pudo observar que el PID con acercamiento exponencial era capaz de controlar varios agentes en simultáneo. Se realizaron pruebas con las distintas *fitness function* con las que se cuenta en la fase de simulación. Para todos los casos se colocaron los agentes en posiciones aleatorias de la mesa de pruebas, evitando proximidad a los muros para prevenir pérdida de visión en el sistema de captura de movimiento.

La primera prueba que se realizó con el incremento de agentes fue con la función Sphere, donde todos los parámetros seleccionados se muestran en el Cuadro 8. Como se evidencia en la Figura 22, al igual que en la primera prueba, el algoritmo fue capaz de generar las trayectorias para que cada robot convergiera al global best encontrado para la función Sphere. Cabe destacar que esta función es de gran utilidad para evaluar la convergencia y velocidad del enjambre debido a su simplicidad al tender a las coordenadas [0,0] y estar completamente en el centro del espacio de trabajo.

Descripción	Ajuste seleccionado
fitness function	Sphere
Controlador	PID con acercamiento exponencial
Tipo de inercia	Exponencial
Factor de constricción	0.2087
Peso cognitivo	2
Peso social	7
Escalador de velocidad	0.4

Cuadro 8: Parámetros de prueba con función Sphere.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

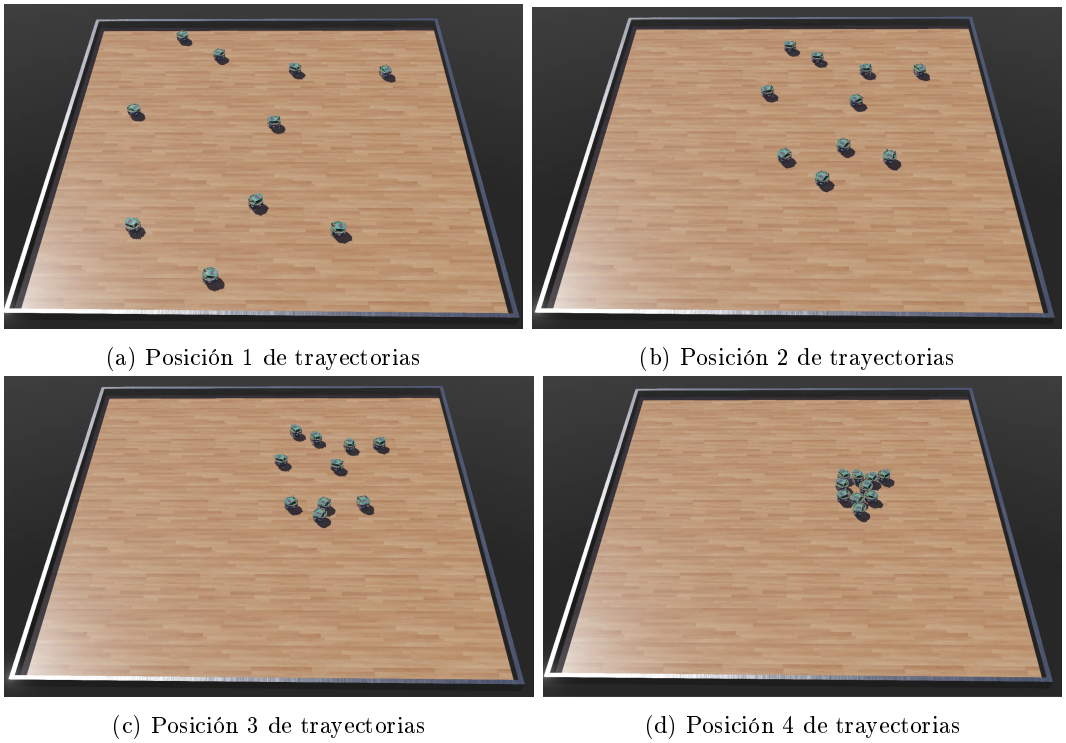
(f) Posición 6 de trayectorias

Figura 22: Trayectorias para función Sphere con parámetros del Cuadro 8.

La segunda *fitness function* que se evaluó fue la función de Rosenbrock. Al igual que la de Sphere, esta es bastante simple, pero presenta un mínimo en las coordenadas [1,1]. Esto proporciona la oportunidad de examinar las trayectorias de los agentes hacia un punto conocido y significativamente central. Para esta prueba se ajusta el PSO de forma no estándar, tomando los valores directamente. La idea era evaluar los rangos permitidos para realizar trayectorias satisfactorias y verificar el funcionamiento de ambos casos del algoritmo. Los valores aplicados pueden apreciarse en el Cuadro 9.

Descripción	**Ajuste seleccionado**
fitness function	Rosenbrock
Controlador	PID con acercamiento exponencial
Tipo de inercia	Aleatoria
Factor de constricción	0.8
Peso cognitivo	2
Peso social	10
Escalador de velocidad	0.032

Cuadro 9: Parámetros de prueba con función de Rosenbrock.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

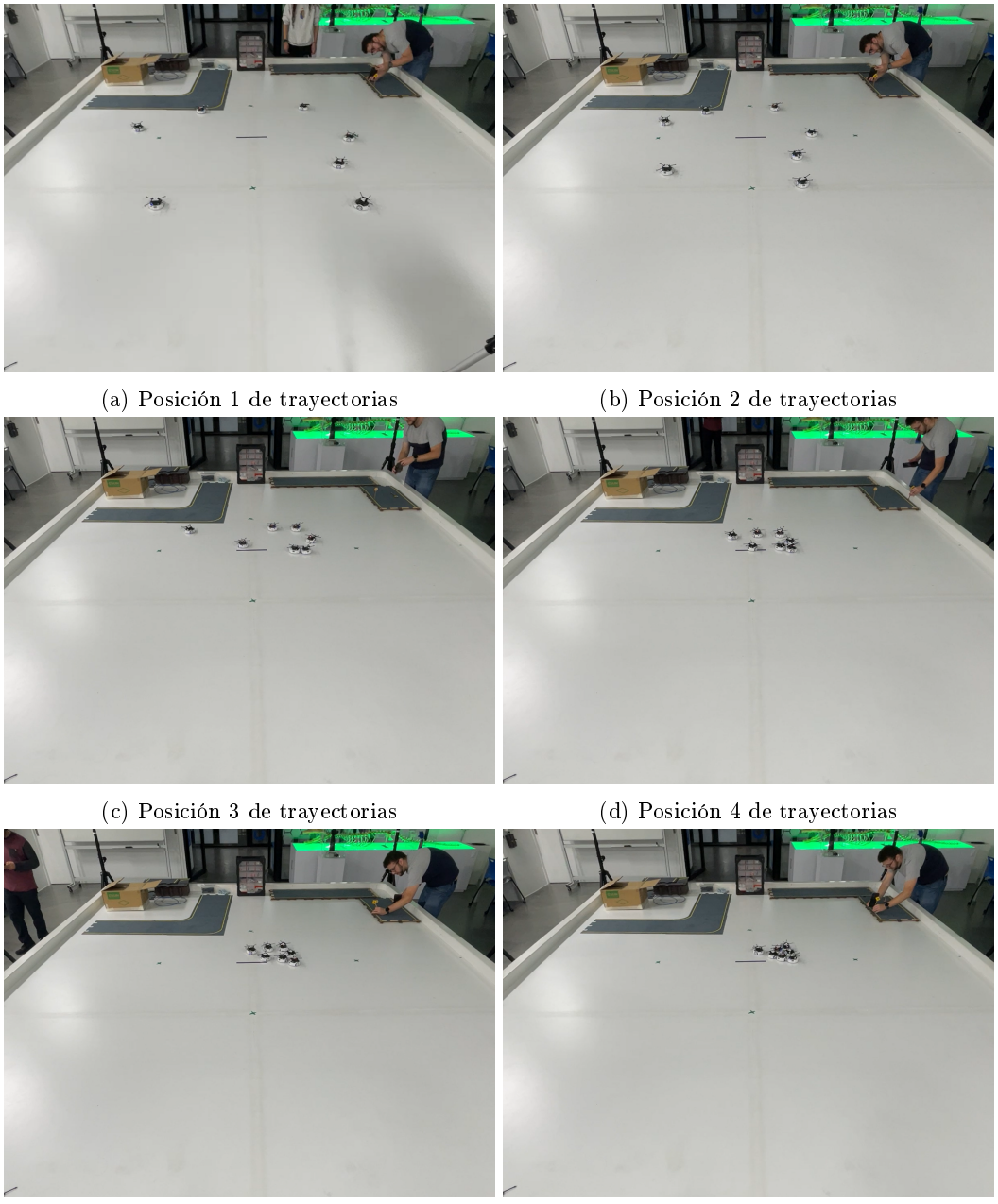
(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 23: Trayectorias para función de Rosenbrock con parámetros del Cuadro 9.

Al revisar la Figura 23 se puede evidenciar como los robots convergen a un punto cercano al previsto. Ahora bien, en la Figura 24 se puede observar que, si bien los robots tienden hacia la dirección de la coordenada esperada, no logran llegar tan cerca como en la simulación. Esto puede explicarse al notar las posiciones iniciales de los robots en cada caso. En la simulación uno de los agentes se encuentra cercano al punto [1,1] esperado, por lo que todos los demás agentes convergen en esa dirección. En la prueba física se puede notar como ninguno de los

agentes se encuentra tan cerca de la coordenada del mínimo para la función Rosenbrock, por lo que el algoritmo encuentra el agente más cercano a este punto y lo establece como el punto de convergencia de los demás agentes.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

(f) Posición 6 de trayectorias

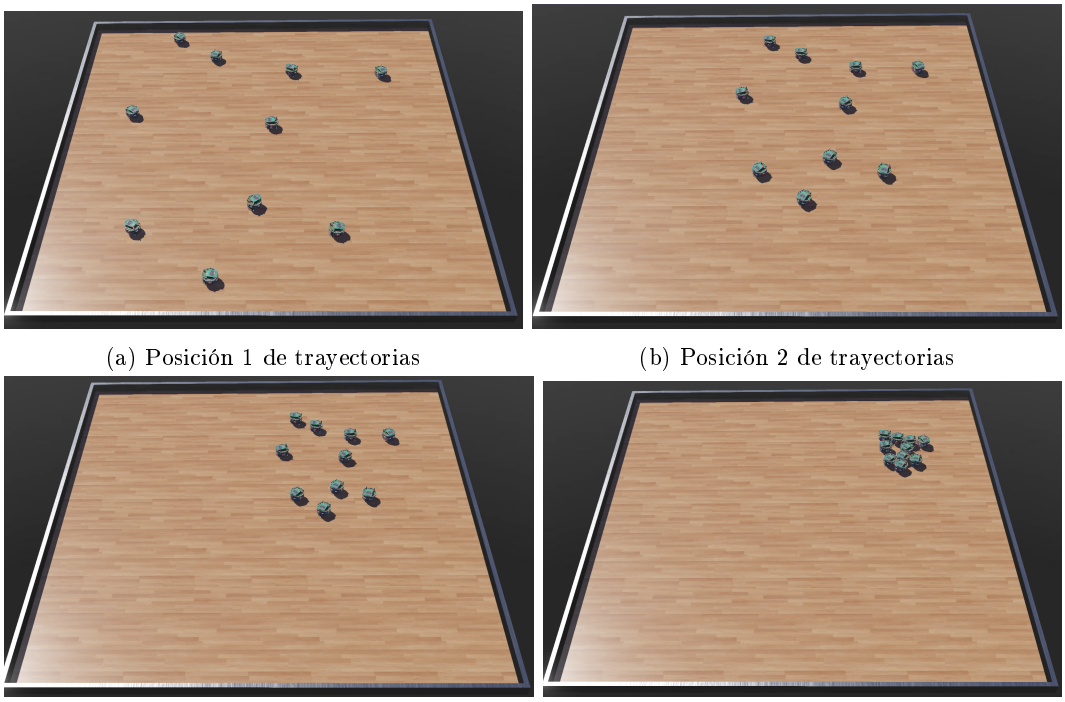
Figura 24: Trayectorias para función de Rosenbrock con parámetros del Cuadro 9.

La función de Booth es relativamente sencilla, pero tiene la particularidad de tener un mínimo absoluto alejado del centro del origen. Esto implica que los agentes deben converger hacia un punto cercano al borde de la mesa de trabajo. Por lo tanto, fue importante garantizar que ningún agente se ubicara demasiado cerca de los extremos para asegurar una

correcta lectura de posiciones por parte del sistema de captura de movimiento y, por ende, una ejecución adecuada de trayectorias para todos los agentes. La función de Booth fue evaluado bajo los parámetros establecidos en el Cuadro 10.

Descripción	**Ajuste seleccionado**
fitness function	Booth
Controlador	PID con acercamiento exponencial
Tipo de inercia	Constante
Factor de constricción	0.2087
Peso cognitivo	1
Peso social	6
Escalador de velocidad	0.5

Cuadro 10: Parámetros de prueba con función Booth.



(a) Posición 1 de trayectorias

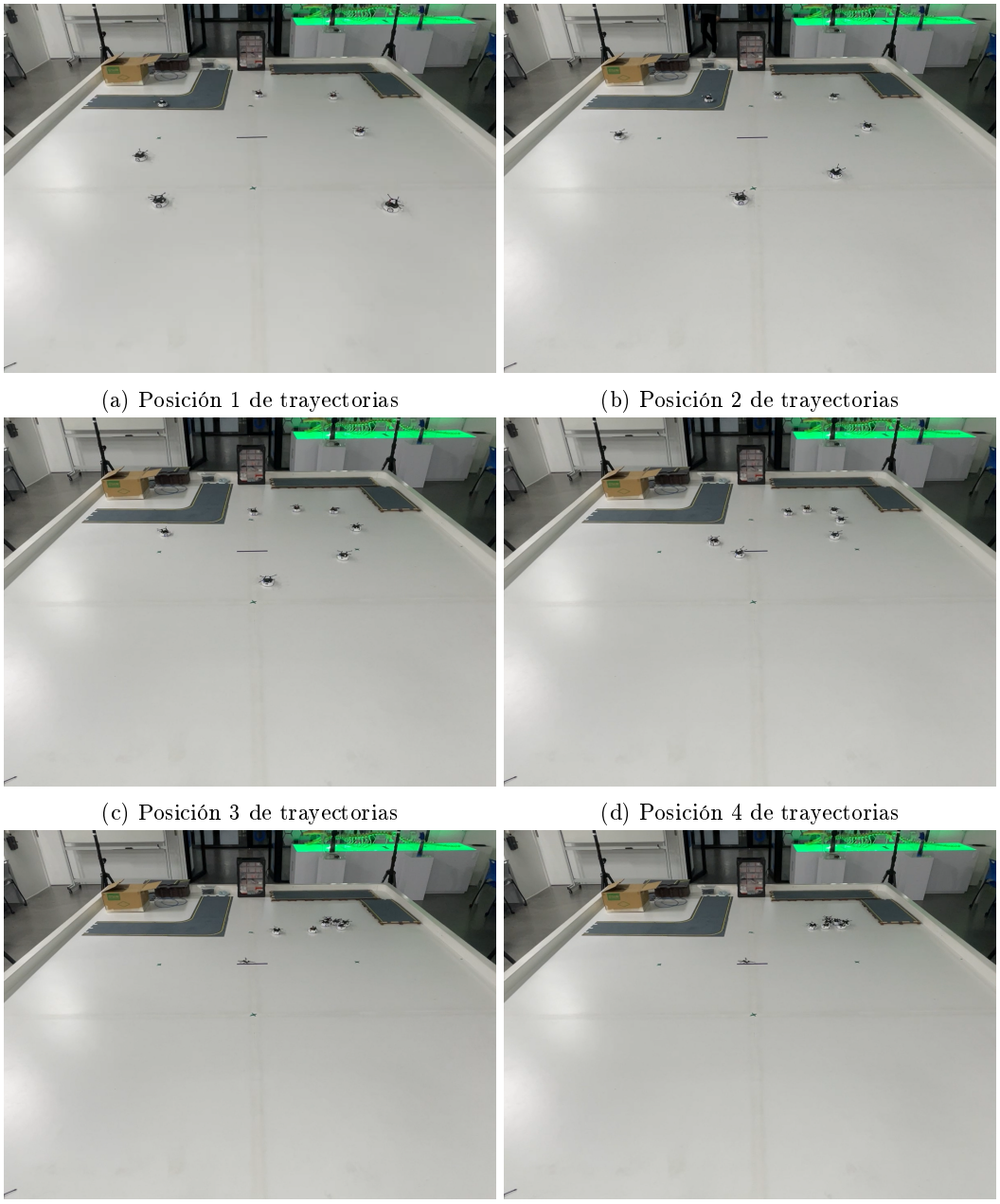
(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 25: Trayectorias para función Booth con parámetros del Cuadro 10.

Se puede apreciar al revisar las figuras 25 y 26 que el algoritmo fue capaz de replicar lo observado en simulación. Todos los agentes convergieron hacia un *global best* que se acerca al mínimo absoluto de la función de Booth. Se pudo notar que dos de los agentes necesitaron mayor tiempo para poder llegar al destino, algo esperado debido a que eran los dos agentes más distanciados del punto de convergencia y a la naturaleza del controlador utilizado.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

(f) Posición 6 de trayectorias

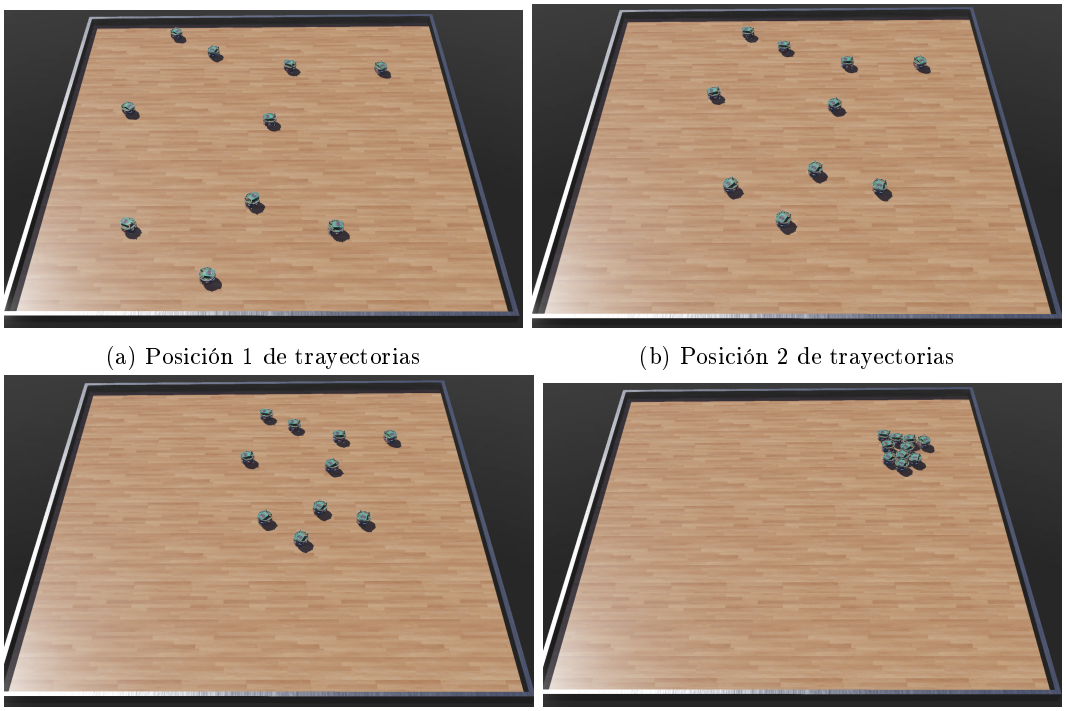
Figura 26: Trayectorias para función Booth con parámetros del Cuadro 10.

La siguiente función evaluada fue la de Himmelblau, que posee múltiples mínimos con el mismo valor. Por lo tanto, todos son mínimos absolutos y el enjambre puede tener a cualquiera de estos punto, tomando en cuenta al agente que se encuentre más cercano a uno de estos mínimos absolutos. Como se puede apreciar en el Cuadro 11 se realizaron unos ajustes para seguir evaluar la ejecución del algoritmo con cambios dentro de los parámetros de PSO.

Descripción	Ajuste seleccionado
fitness function	Himmelblau
Controlador	PID con acercamiento exponencial
Tipo de inercia	Constante
Factor de constricción	0.1716
Peso cognitivo	1
Peso social	7
Escalador de velocidad	0.5

Cuadro 11: Parámetros de prueba con función Himmelblau.

Se puede notar en las figuras 27 y 28 que en ambos casos el agente que se encontraba más cerca de un mínimo absoluto era el ubicado sobre la esquina superior derecha de la imagen, donde todos los agentes convergen hacia la posición de este al ser el *global best* para esta función.



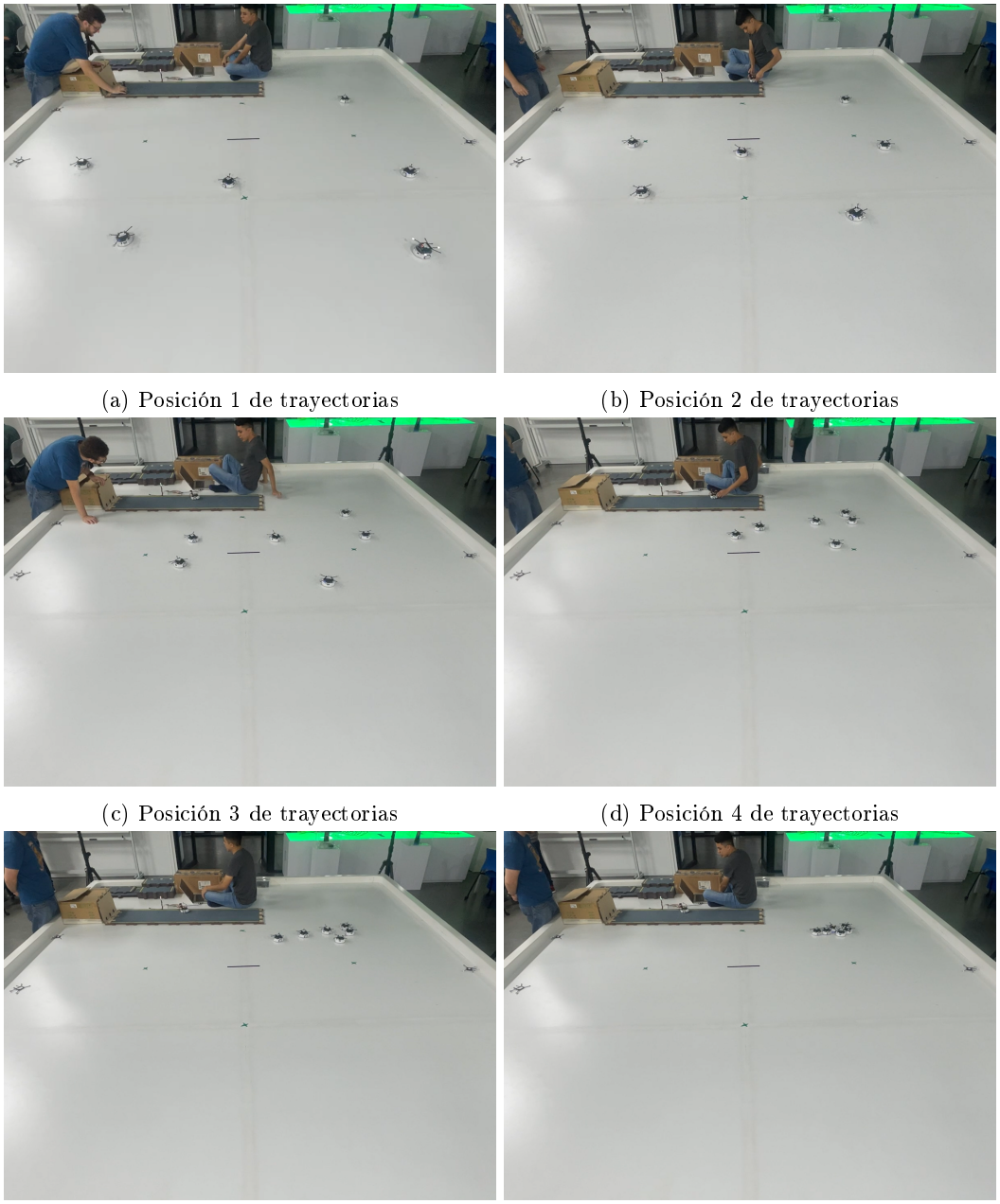
(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 27: Trayectorias para función Himmelblau con parámetros del Cuadro 11.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

(f) Posición 6 de trayectorias

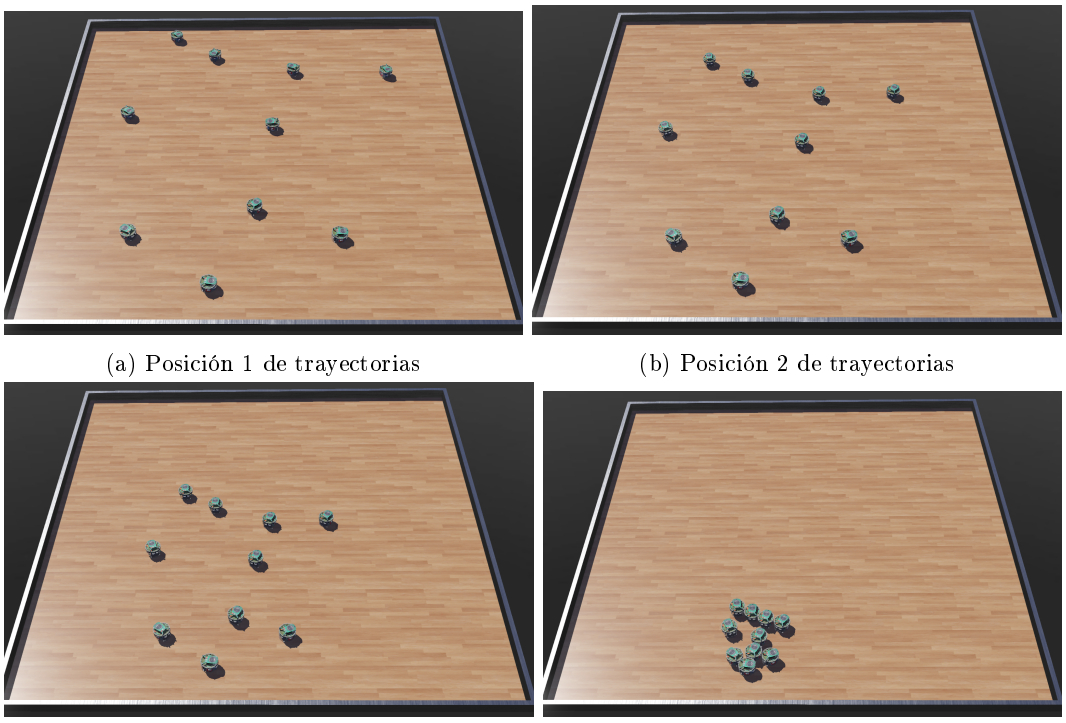
Figura 28: Trayectorias para función Himmelblau con parámetros del Cuadro 11.

La función de Schaffer posee múltiples mínimos con el mismo valor que se ubican en forma de anillo en el exterior de la función. Es decir que para está función los agentes tienden sobre cualquier extremo del espacio de trabajo donde se encuentre el robot más alejado del origen. Los parámetros utilizados para esta prueba se pueden ver en el Cuadro 12.

Descripción	Ajuste seleccionado
fitness function	Schaffer
Controlador	PID con acercamiento exponencial
Tipo de inercia	Aleatoria
Factor de constricción	0.2087
Peso cognitivo	2
Peso social	5
Escalador de velocidad	0.6

Cuadro 12: Parámetros de prueba con función Schaffer.

En las figuras 29 y 30 se puede notar que para ambos casos, simulación y prueba física, los agentes tienden hacia el robot más alejado del centro de la mesa de trabajo, cumpliendo con lo esperado por la naturaleza de la función aplicada.



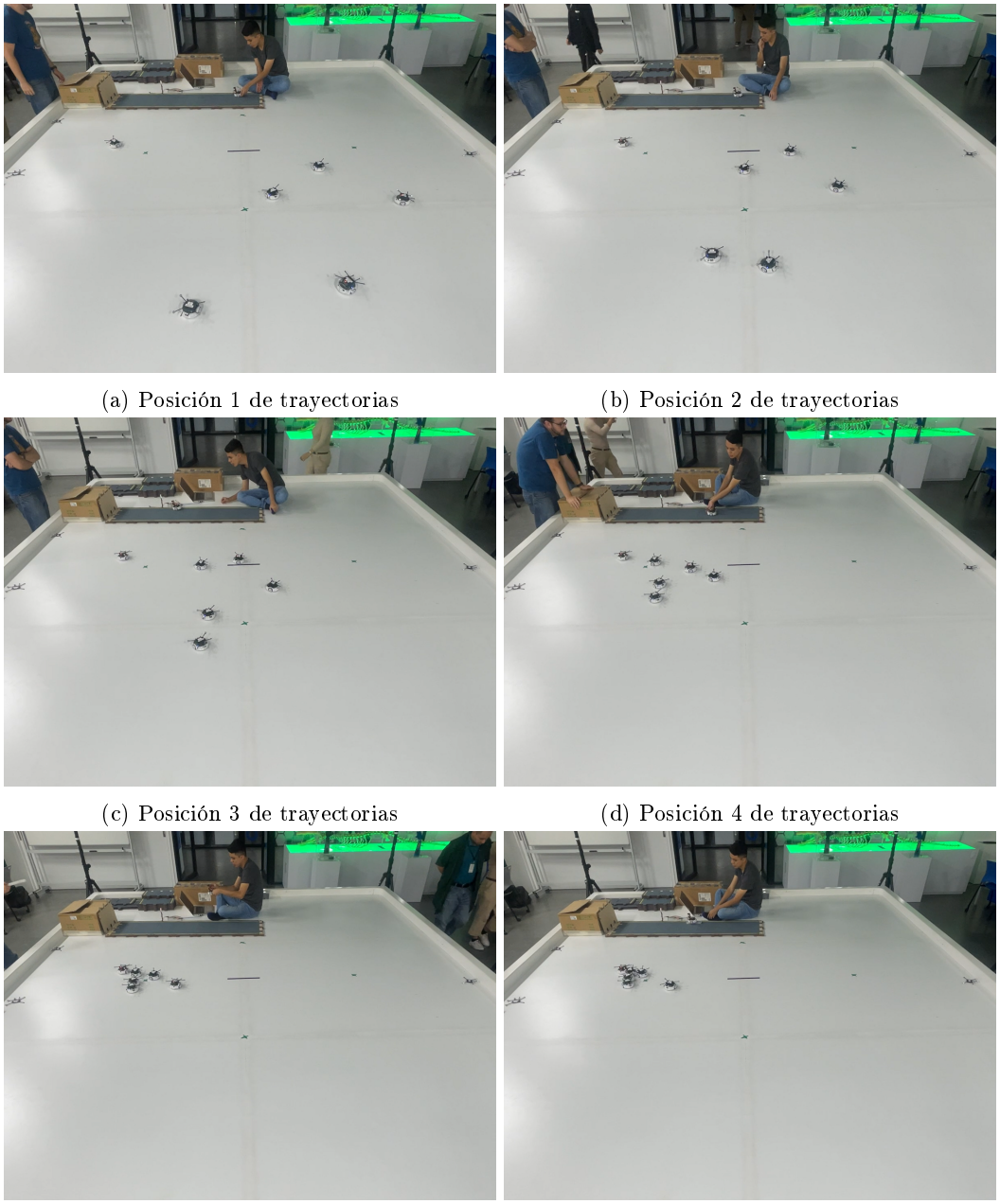
(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 29: Trayectorias para función Schaffer con parámetros del Cuadro 12.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

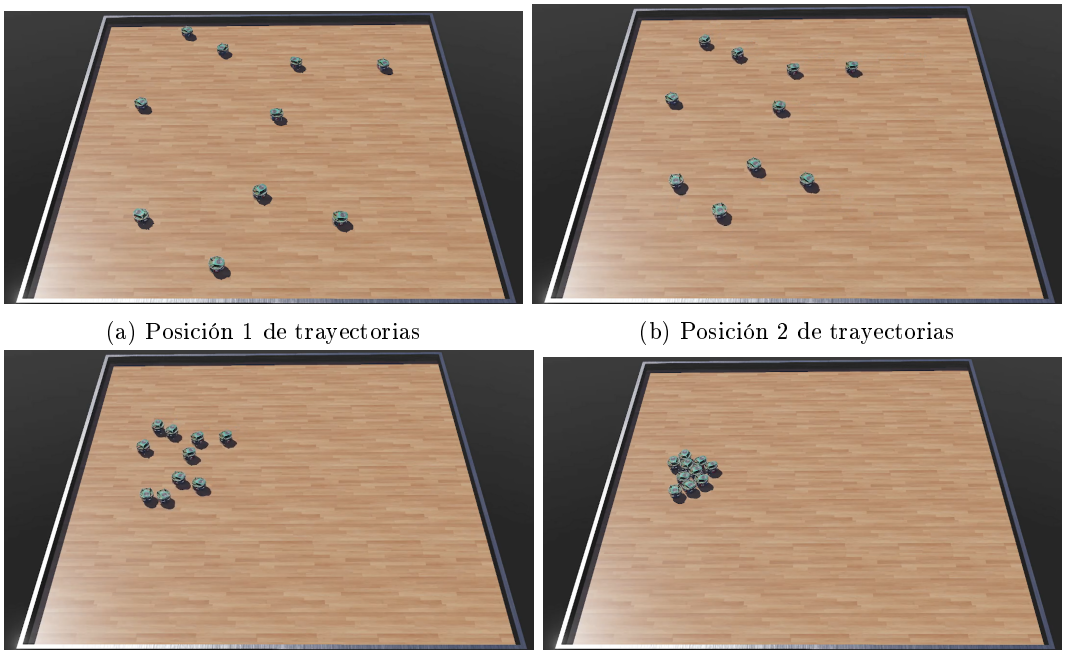
(f) Posición 6 de trayectorias

Figura 30: Trayectorias para función Schaffer con parámetros del Cuadro 12.

La última prueba se realizó con la función de Keane, que tiene dos mínimos absolutos de igual valor. Estos mínimos absolutos se encuentran alineados con los ejes del plano, uno al eje Y y el otro con el eje X, con la misma distancia desde el origen en ambos casos. Para la aplicación de esta función se pueden observar los parámetros utilizados en el Cuadro 13.

Descripción	Ajuste seleccionado
fitness function	Keane
Controlador	PID con acercamiento exponencial
Tipo de inercia	Caótica
Factor de constricción	0.1459
Peso cognitivo	2
Peso social	6
Escalador de velocidad	0.5

Cuadro 13: Parámetros de prueba con función Keane.



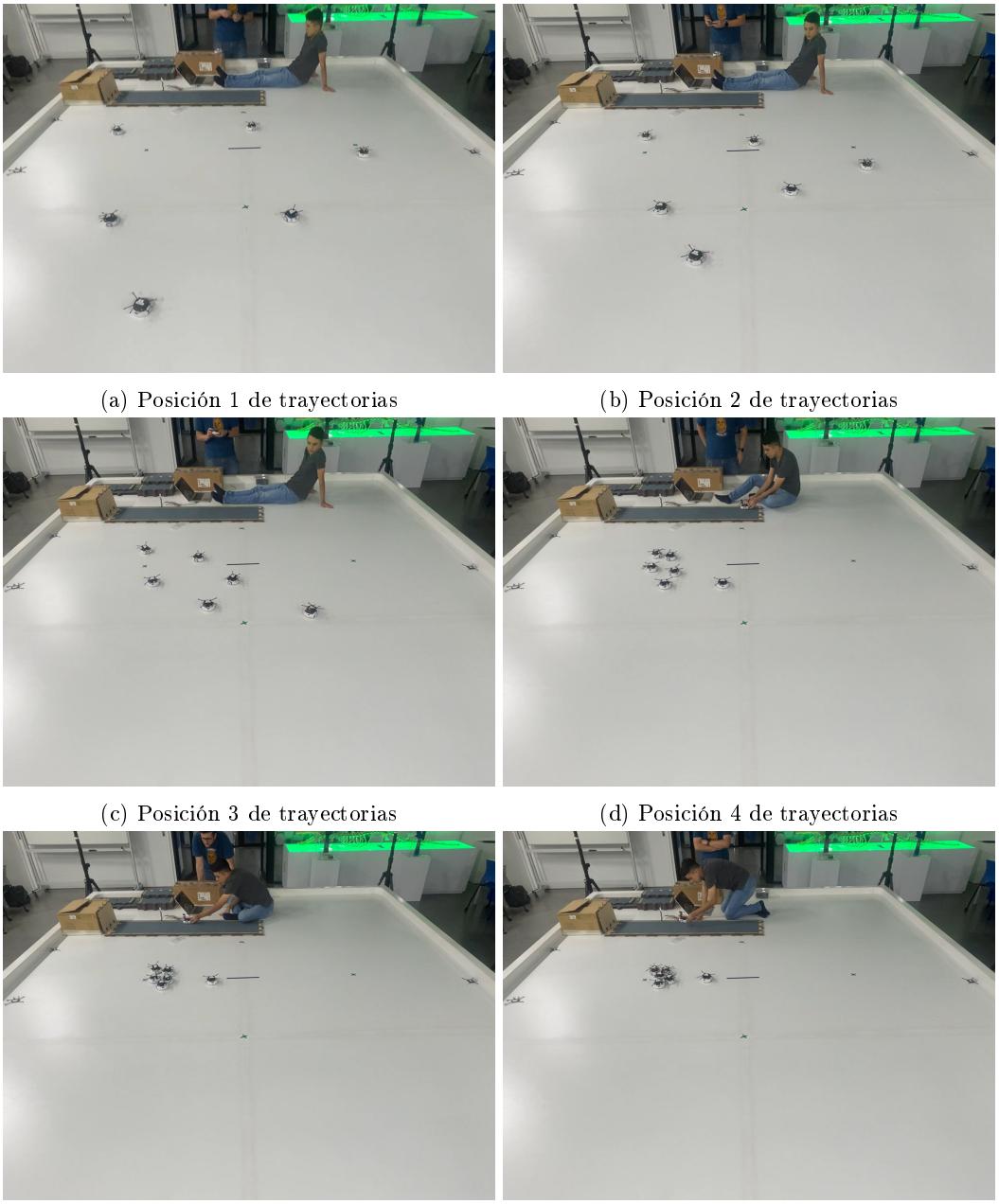
(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

Figura 31: Trayectorias para función de Keane con parámetros del Cuadro 13.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

(e) Posición 5 de trayectorias

(f) Posición 6 de trayectorias

Figura 32: Trayectorias para función de Keane con parámetros del Cuadro 13.

Un criterio que se tomó en cuenta para comparar las versión simulada contra la física del algoritmo fue el tiempo de ejecución promedio para cada prueba. Como se puede apreciar en el Cuadro 14, el algoritmo implementado físicamente demanda un tiempo de convergencia que es al menos un 87.5 % y hasta un 212.5 % mayor en comparación con la simulación. Era esperado que el algoritmo migrado para su aplicación de forma centralizada tomará un mayor tiempo, esto debido a que una sola máquina deber calcular y ejecutar las trayectorias de todos los agentes. El tiempo en físico estuvo entre los valores de la Cuadro 14 dependiendo de los ajustes específicos para el controlador en cada prueba, siempre basados en los valores

de la Cuadro 3.

Se pudo observar un aumento en el tiempo de ejecución a medida que incrementaba el número de agentes que el algoritmo debía controlar desde MATLAB. Este fenómeno señala que el costo computacional de gestionar una mayor cantidad de agentes se refleja en el tiempo de ejecución del sistema de control.

Descripción	**Duración (seg)**
Simulación	24
Físico	45-75

Cuadro 14: Tiempo de ejecución promedio del algoritmo.

8.3. Análisis de trayectorias generadas por el sistema de control

El desempeño en la convergencia de los agentes hacia el *global best*, identificado por el algoritmo, fue un aspecto crucial en [5]. Además, se consideró fundamental que las trayectorias generadas fueran lo suficientemente suaves para preservar la integridad física de los robots a la hora de realizar pruebas en el mundo real, siendo evaluado dentro de las simulaciones en dicha fase. Con este propósito, se evaluó detalladamente el comportamiento de las trayectorias generadas por el controlador PID con acercamiento exponencial utilizado a lo largo de todas la pruebas en físico para cada *fitness function*.

Se llevaron a cabo mediciones de la posición de cada agente, junto con el registro de la velocidad de los motores en cada uno de ellos. Se realizaron con la finalidad de analizar la convergencia en el espacio de trabajo y detectar posibles fluctuaciones en la velocidad que pudieran afectar negativamente los actuadores de los agentes. Cabe resaltar que en [5] se llevó a cabo dicho análisis en relación al tiempo en simulación, utilizando la herramienta de Webots. Mientras que las pruebas en entorno físico se presentan con respecto a cada iteración del algoritmo, aspecto que se debió considerar al analizar el comportamiento del control y seguimiento de trayectoria para los robots. El cambio en esta modalidad de medición se debe a la disparidad en la velocidad de iteración del algoritmo y a la facilidad de medir el tiempo en las simulaciones, en contraste con el entorno real.

El primer caso evaluado fue una prueba con la función *Sphere*, con parámetros distintos que en la prueba del Cuadro 8, presentado a continuación:

Descripción	Ajuste seleccionado
fitness function	Sphere
Controlador	PID con acercamiento exponencial
Tipo de inercia	Aleatoria
Factor de constricción	0.1459
Peso cognitivo	2
Peso social	6
Escalador de velocidad	0.5

Cuadro 15: Parámetros de prueba Sphere para análisis de trayectorias y velocidades.

La Figura 33 presenta las trayectorias efectuadas por cada agente para converger al punto *global best*. En ella, se aprecia que las trayectorias exhiben un comportamiento suave, dirigiéndose de manera clara hacia la convergencia de todos los agentes. Además, en la Figura se pueden observar las velocidades de cada rueda para todos los agentes del enjambre.

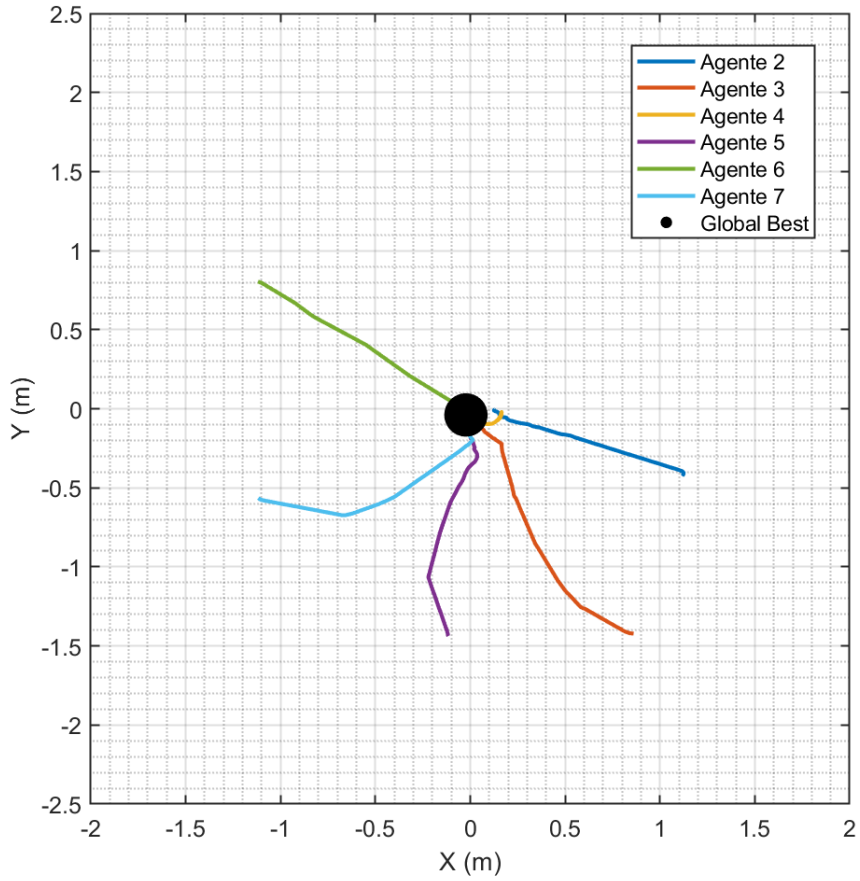
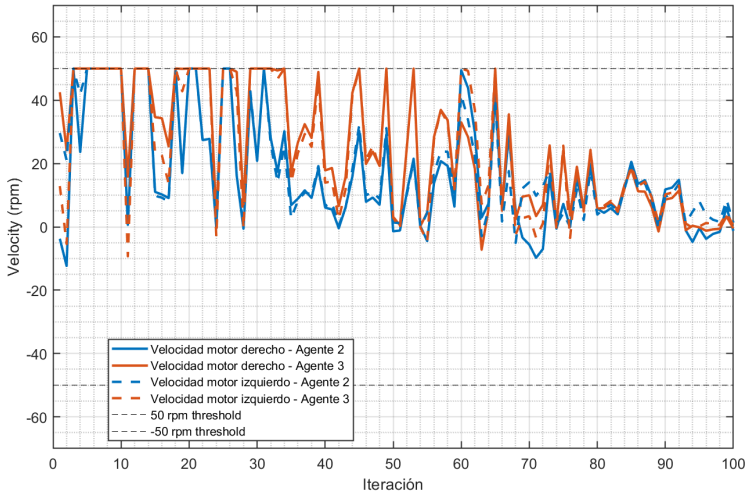
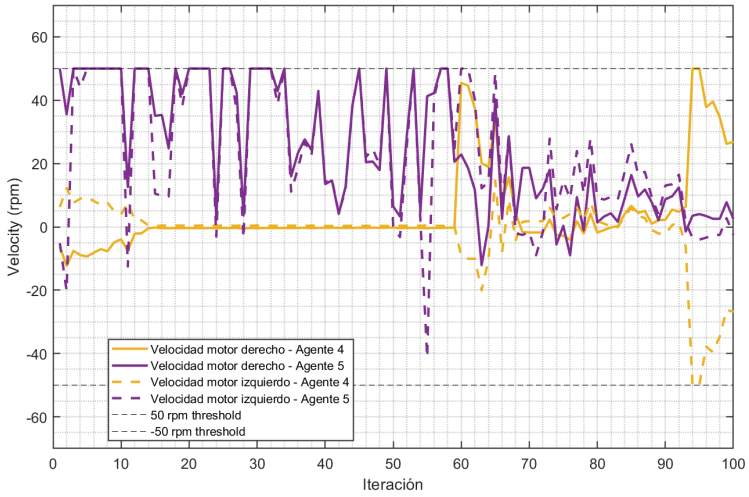


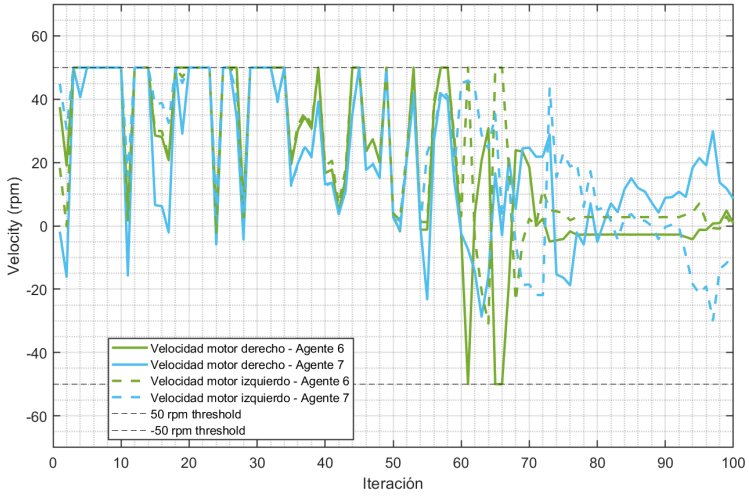
Figura 33: Trayectorias resultantes del enjambre con parámetros de Cuadro 15.



(a) Velocidad en cada rueda de agentes 2-3



(b) Velocidad en cada rueda de agentes 4-5



(c) Velocidad en cada rueda de agentes 6-7

Figura 34: Velocidades de los agentes para prueba con parámetros de Cuadro 15.

Aunque las velocidades parecen experimentar variaciones considerables en su magnitud, es crucial notar que estos cambios ocurren con menor frecuencia que en las pruebas simuladas. Esto se atribuye al mayor tiempo requerido para llevar a cabo todos los pasos necesarios para la convergencia a nivel físico, como se detalla en el Cuadro 14. Además, estos cambios se presentan dentro de los limites de velocidad establecidos para las ruedas de los agentes, representados como lineas punteada horizontales dentro de cada gráfica, evitando magnitudes que puedan dañar los motores en cada robot.

Para asegurar que el comportamiento de los agentes fuera el mismo para otros casos, se realizó una segunda prueba, alterando los parámetros del algoritmo y una *fitness function* distinta. Esta prueba contó con las siguientes características:

Descripción	**Ajuste seleccionado**
fitness function	Schaffer No.4
Controlador	PID con acercamiento exponencial
Tipo de inercia	Exponencial
Factor de constricción	0.2087
Peso cognitivo	2
Peso social	5
Escalador de velocidad	0.6

Cuadro 16: Parámetros de prueba Schaffer No.4 para análisis de trayectorias y velocidades.

La Figura 16 presenta las trayectorias efectuadas por cada agente para converger al punto *global best* en este caso. Como se puede observar, el comportamiento se mantiene suave a lo largo de las trayectorias y con una convergencia directa hacia la solución encontrada por el algoritmo. Esto reafirma la capacidad del conjunto de controlador y algoritmo para dirigir de manera estable y directa a todos los agentes hacia su punto de convergencia.

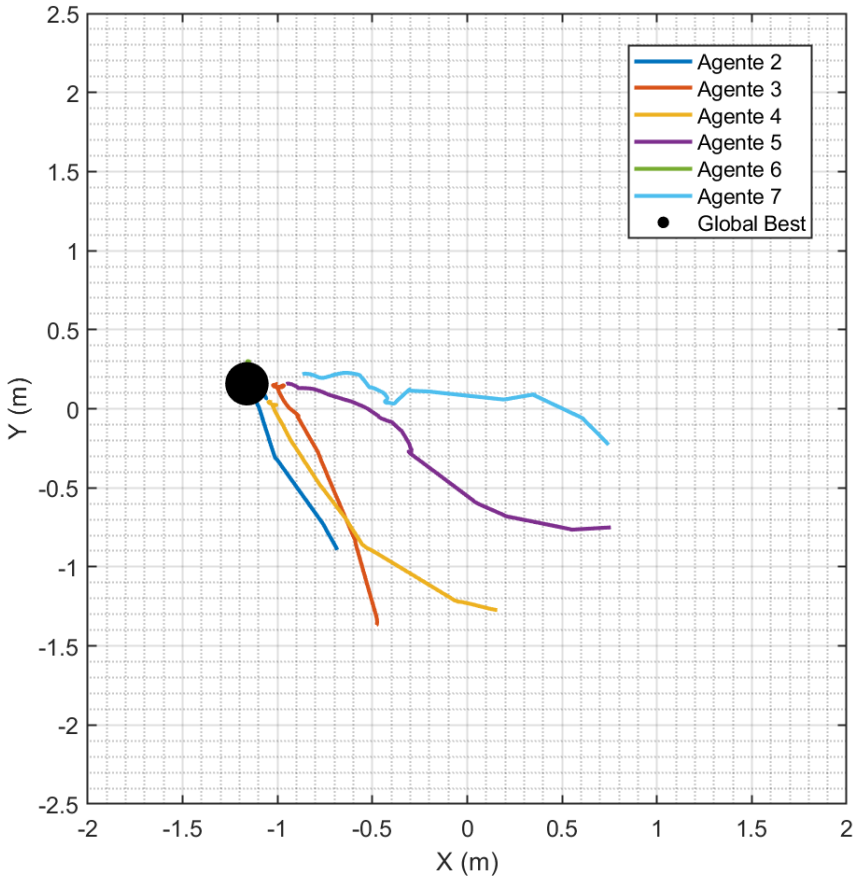
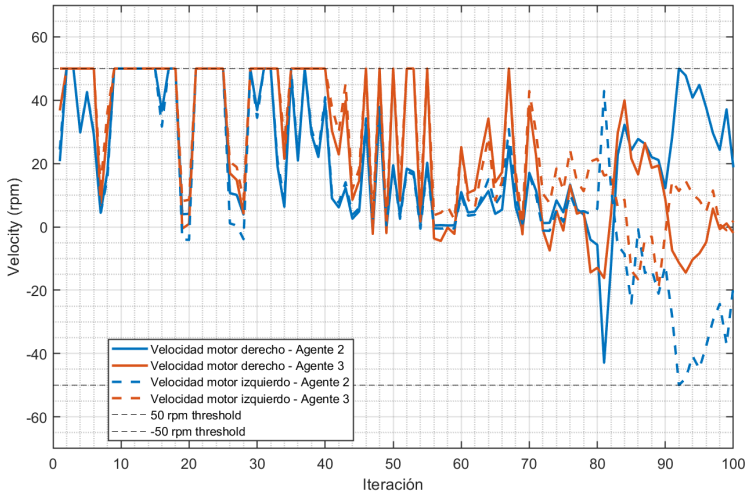
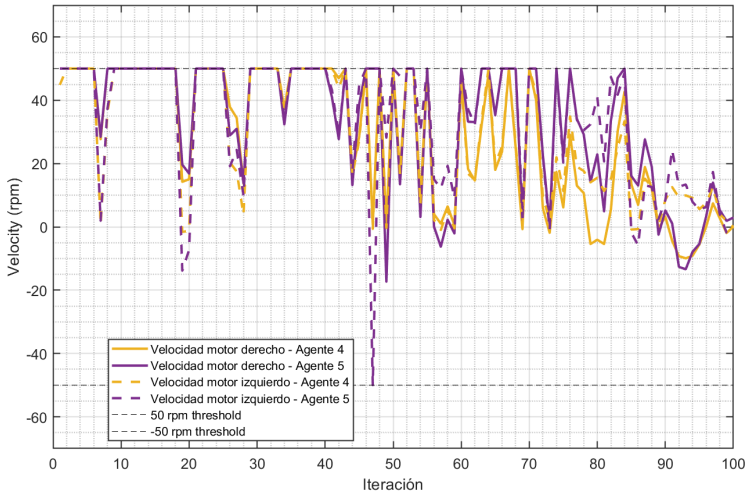


Figura 35: Trayectorias resultantes del enjambre con parámetros de Cuadro 16.

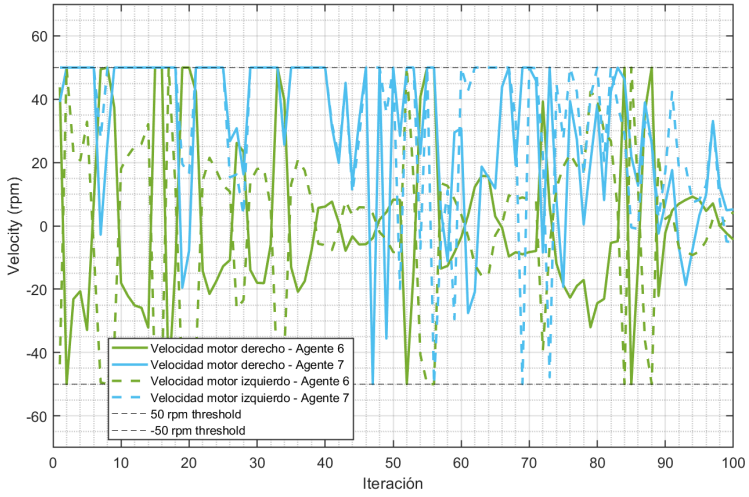
En el caso de las velocidades, presentadas en la Figura 36, se puede confirmar que los resultados obtenidos en esta prueba son consistentes con los presentados en la prueba inicial. Las variaciones en la velocidad se mantienen en niveles moderados y siempre dentro de los límites establecidos para el control de los robots. En el caso de los agentes 6 y 7, se observa un aumento en las fluctuaciones de velocidad, atribuidas al contacto inevitable entre los agentes, especialmente en los marcadores utilizados para la lectura por parte del sistema de captura de movimiento durante la convergencia. A pesar de esta situación, el comportamiento del enjambre fue el adecuado, tanto en términos de trayectoria como en el control seguro de los actuadores de cada robot.



(a) Velocidad en cada rueda de agentes 2-3



(b) Velocidad en cada rueda de agentes 4-5



(c) Velocidad en cada rueda de agentes 6-7

Figura 36: Velocidades de los agentes para prueba con parámetros de Cuadro 16.

CAPÍTULO 9

Desempeño de experimentos con Ant Colony Optimization

El algoritmo de ACO permite generar trayectorias para una zona determinada, pero el rendimiento del algoritmo puede verse afectado por la distancia entre el nodo destino y el nodo final. Por ello, en este capítulo se evalúan una serie de pruebas para determinar los ajustes necesarios en función de la distancia. Los ajustes se realizan en los valores de la cantidad de hormigas desplegadas, la evaporación de la feromona y los pesos asignados a distintos factores del algoritmo.

9.1. Experimentación con diferentes ajustes del algoritmo

El algoritmo de ACO se evaluó primero para replicar el experimento de la fase previa[8], en el que se estableció una cuadrícula de 10×10 con el nodo 56 como destino y el nodo 1 como inicio. Se utilizaron los siguientes parámetros para la generación de la trayectoria:

Descripción	**Ajuste seleccionado**
Controlador	PID con acercamiento exponencial
Cantidad de hormigas	100
Iteración máx.	150
Tasa de evaporación (ρ)	0.9
Peso de feromona (α)	1.5
Peso de costo de link (β)	1.0
Constante de regulación (Q)	2.1
% de hormigas (ϵ)	1.0

Cuadro 17: Parámetros de ACO para prueba 1.

Como se puede observar en la Figura 37, la trayectoria encontrada para llegar desde el

nodo inicial hacia el nodo destino es bastante directa, como se presentaba en los resultados anteriores en[8]. El algoritmo luego ajusta la trayectoria generada para las dimensiones de la mesa de pruebas y las características de las plataformas móviles físicas. Además, con los factores seleccionados para esta prueba el algoritmo converge antes de la iteración máxima colocada.

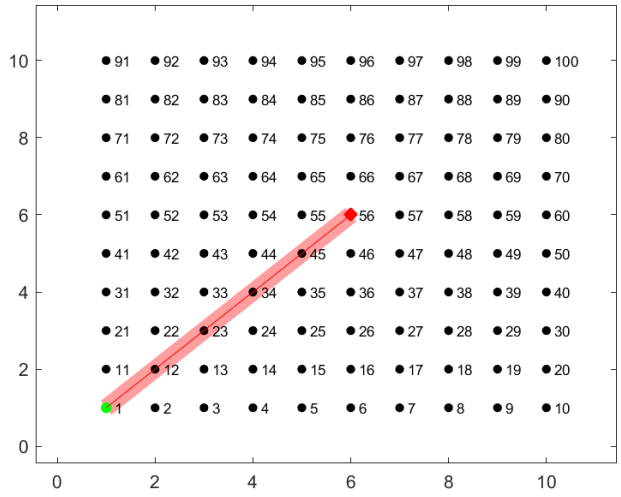
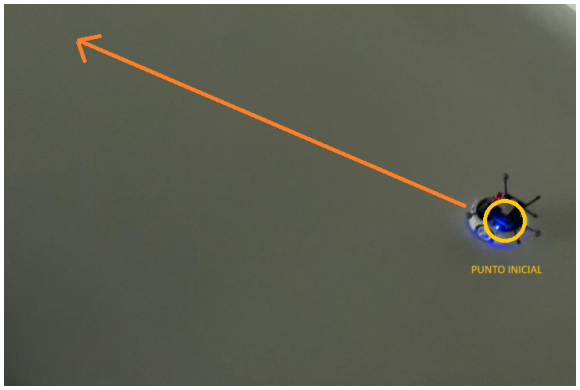
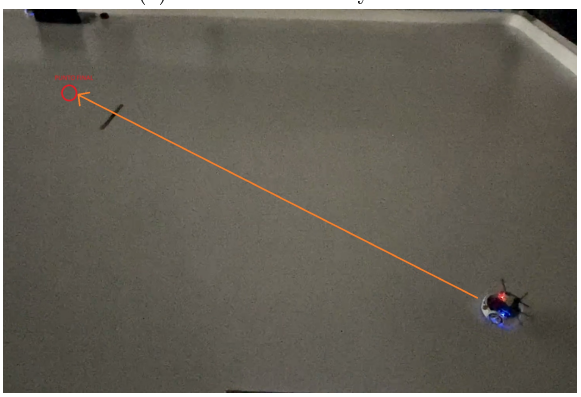


Figura 37: Trayectoria encontrada por ACO con ajustes del Cuadro 17.

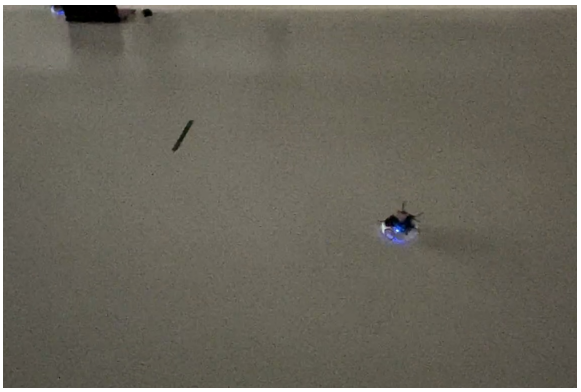
Queda claro al revisar la Figura 38 y su respectiva trayectoria generada por datos del sistema de captura de movimiento, en la Figura 39d, que el controlador fue capaz de seguir la trayectoria obtenida a partir del punto inicial, hasta llegar al destino. En la Figura 39b y la Figura 39c puede observarse que el controlador tiene un buen resultado en cuanto a las variaciones de velocidades lineales y angular, manteniendo velocidades bastante regulares a partir de la orientación inicial hacia el nodo destino. Además, se comprueba con la Figura 39a que el controlador utilizado es capaz de mantener reguladas la velocidad de los motores del robot, sin existencia de cambios bruscos ni superando la velocidad máxima seleccionada.



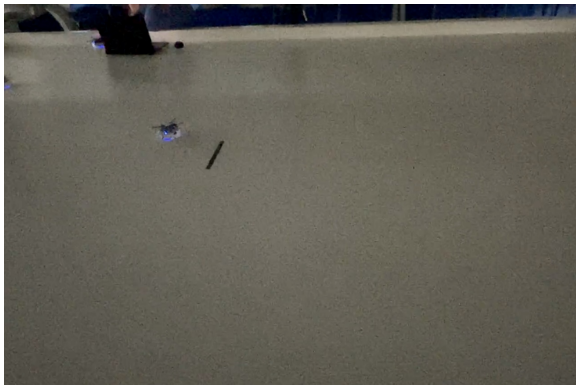
(a) Posición 1 de trayectoria



(b) Posición 2 de trayectoria

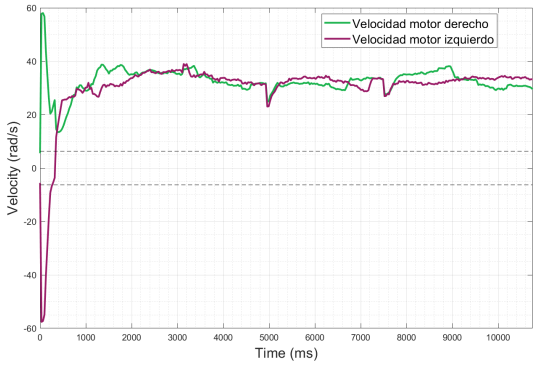


(c) Posición 3 de trayectoria

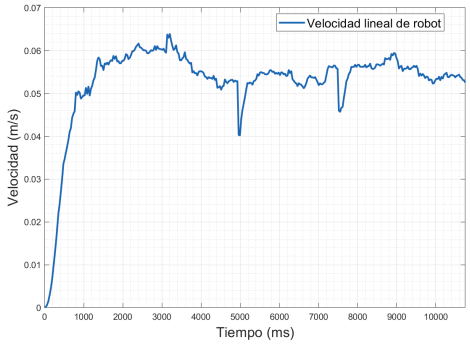


(d) Posición 4 de trayectoria

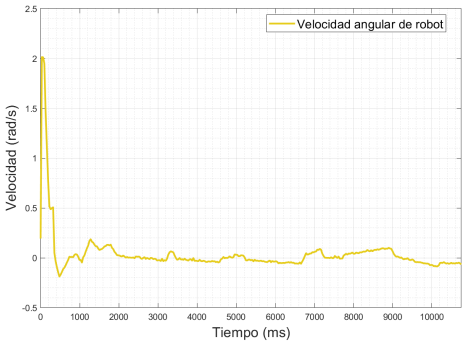
Figura 38: Trayectoria generada en físico para prueba con parámetros del Cuadro 17.



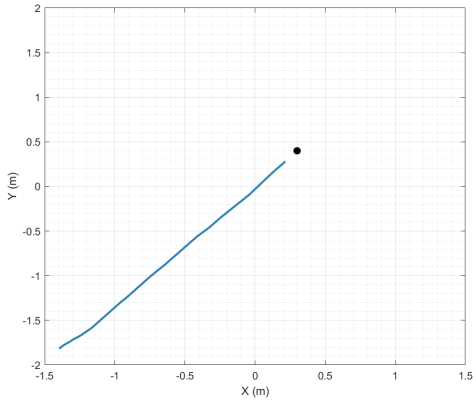
(a) Velocidad de motores del robot.



(b) Velocidad lineal del robot.



(c) Velocidad angular del robot.



(d) Trayectoria realizada por el robot en físico

Figura 39: Trayectoria generada en físico para prueba con parámetros del Cuadro 17.

En la segunda prueba, el nodo final se mantuvo en el 56, pero el nodo inicial se cambió al 41. El objetivo era observar cómo afecta el punto de partida a la generación de la trayectoria. Además, se realizaron los siguientes cambios en los parámetros del algoritmo para comprobar su robustez:

Descripción	**Ajuste seleccionado**
Controlador	PID con acercamiento exponencial
Cantidad de hormigas	100
Iteración máx.	250
Tasa de evaporación (ρ)	0.9
Peso de feromona (α)	1.5
Peso de costo de link (β)	0.8
Constante de regulación (Q)	1.5
% de hormigas (ϵ)	0.9

Cuadro 18: Parámetros de ACO para prueba 2.

Como puede observarse en la Figura 40, la trayectoria encontrada para esta prueba es más compleja que la anterior, pese a que el nodo inicial no tiene una mayor distancia hacia el nodo destino. Se añadieron más iteraciones al algoritmo para facilitar la generación de

una trayectoria satisfactoria, además de que se incremento el número de hormigas.

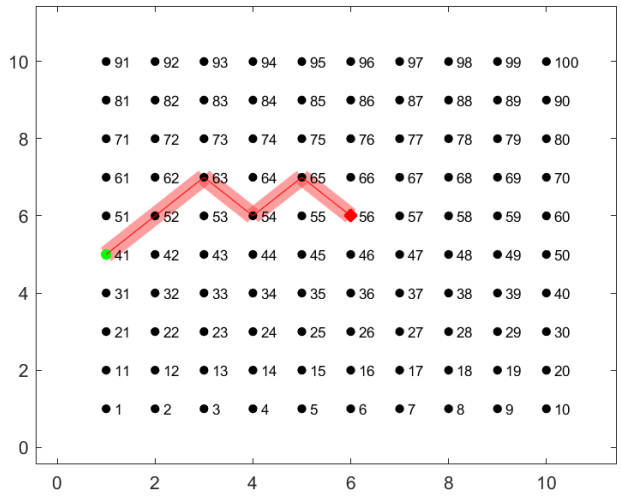
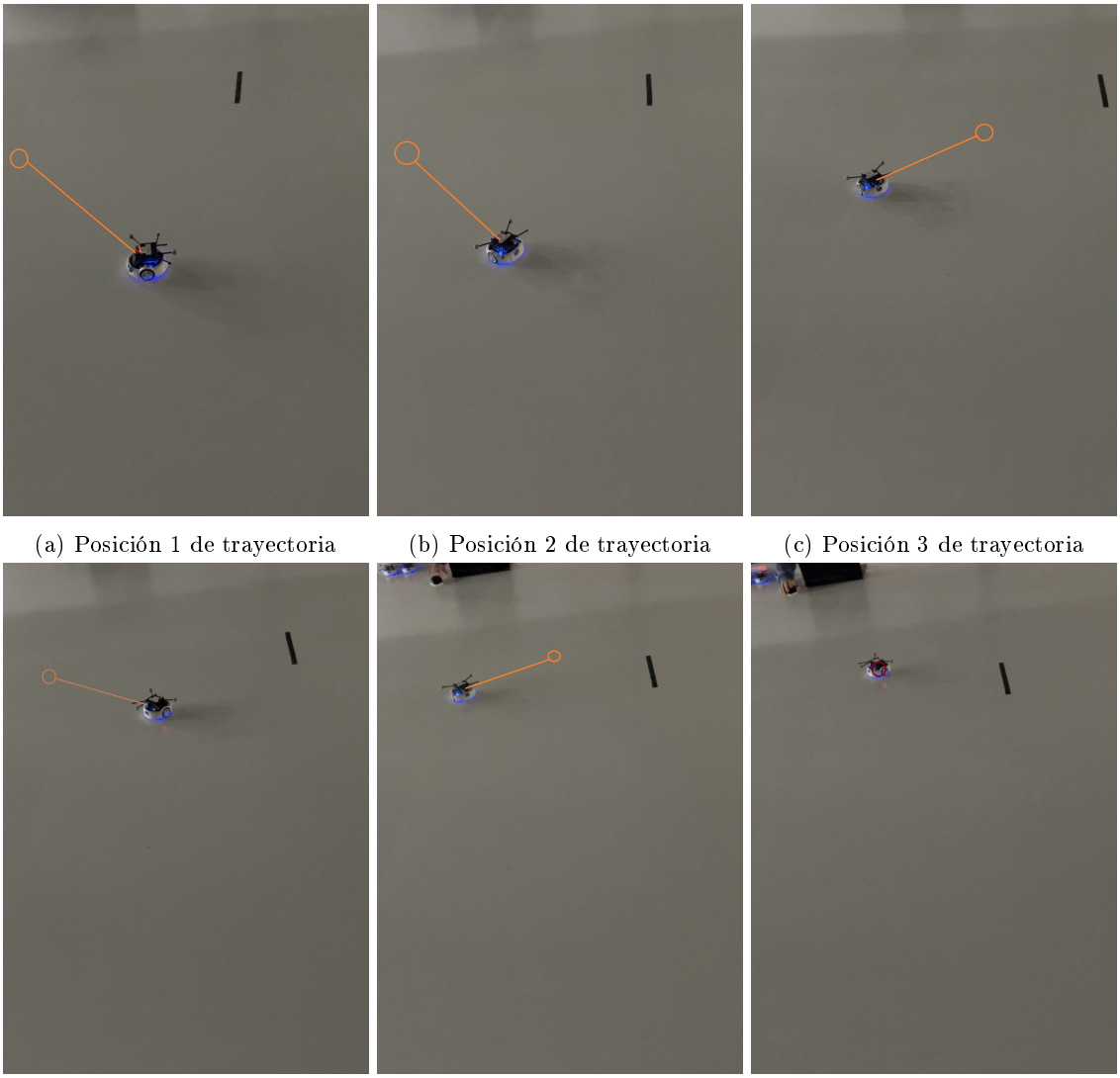


Figura 40: Trayectoria encontrada por ACO con ajustes del Cuadro 18.

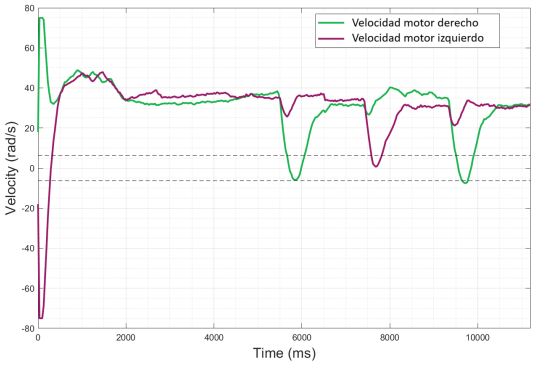
En la Figura 41 se observan la trayectoria general y los cambios de dirección en cada ubicación necesarios para llegar a la ubicación del nodo final. El robot fue capaz de seguir la trayectoria generada sin ningún problema y de forma casi exacta a la generada por el algoritmo, como puede observarse en la Subfigura 42d. En el resto de la Figura 42 podemos notar que al igual que en el caso anterior, las velocidades de los motores y del robot en general se mantienen estables y con valores que permiten un movimiento fluido y controlado.



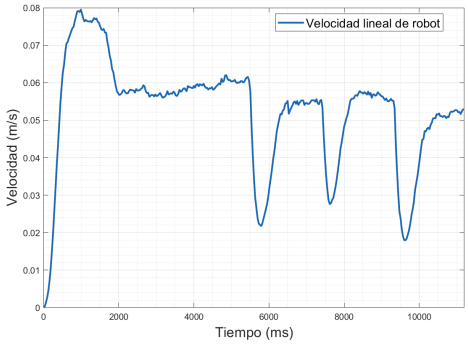
(a) Posición 1 de trayectoria (b) Posición 2 de trayectoria (c) Posición 3 de trayectoria

(d) Posición 4 de trayectoria (e) Posición 5 de trayectoria (f) Posición 6 de trayectoria

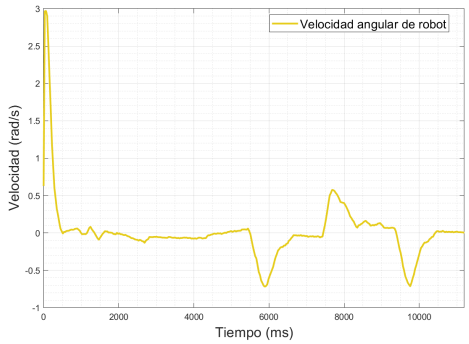
Figura 41: Trayectoria generada en físico para prueba con parámetros del Cuadro 18.



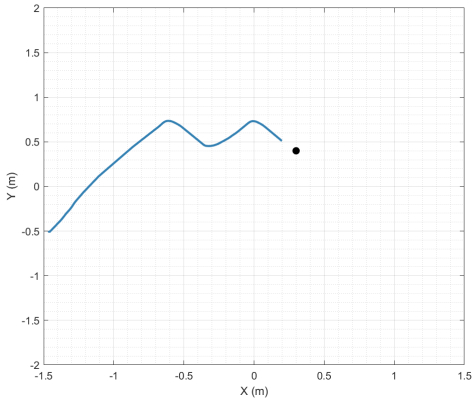
(a) Velocidad de motores del robot.



(b) Velocidad lineal del robot.



(c) Velocidad angular del robot.



(d) Trayectoria realizada por el robot en físico

Figura 42: Trayectoria generada en físico para prueba con parámetros del Cuadro 18.

Se deseaba probar la solución encontrada por el algoritmo ante mayores distancias entre el nodo inicial y el final. Por esta razón se realizó una selección del nodo 11 como inició y el nodo 48 como destino de la prueba con los parámetros mostrados en el Cuadro 19. Se puede observar como el algoritmo tiende a buscar la forma de colocarse primero en un nodo que este completamente en una diagonal que pase por el nodo destino. Luego de lograr colocarse en diagonal al objetivo, el algoritmo es capaz de crear una trayectoria directa hacia el nodo 48.

Descripción	**Ajuste seleccionado**
Controlador	PID con acercamiento exponencial
Cantidad de hormigas	50
Iteración máx.	250
Tasa de evaporación (ρ)	0.9
Peso de feromona (α)	1.1
Peso de costo de link (β)	0.8
Constante de regulación (Q)	1.5
% de hormigas (ϵ)	0.9

Cuadro 19: Parámetros de ACO para prueba 3.

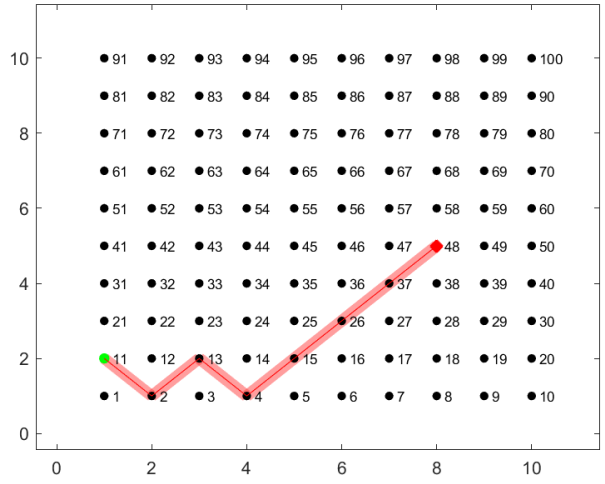
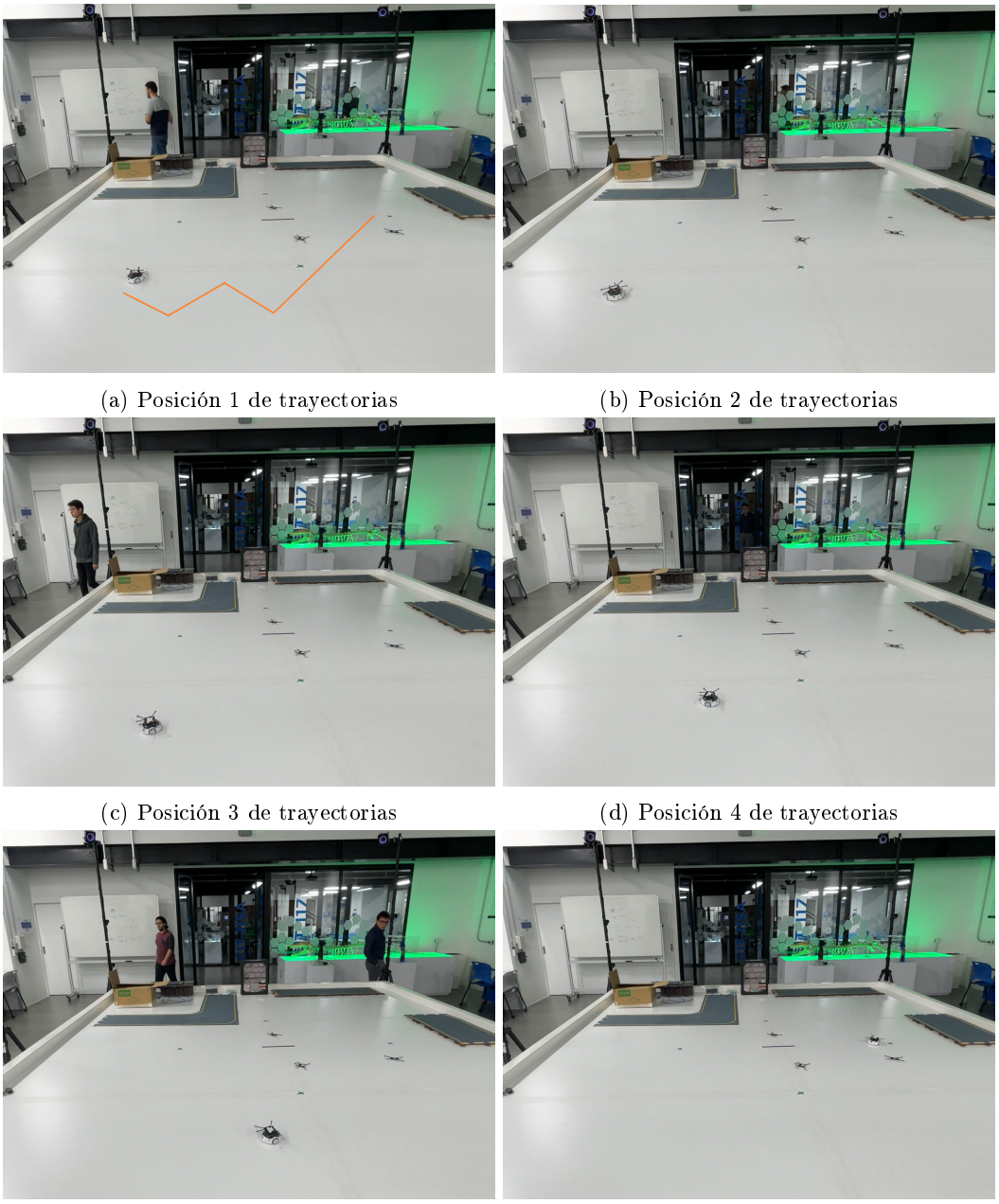


Figura 43: Trayectoria encontrada por ACO con ajustes del Cuadro 19.

En la Figura 44, podemos verificar que tanto el algoritmo como el controlador lograron que el agente llegara a la ubicación final establecida por la trayectoria generada. Es importante destacar que se realizaron pruebas al alejar al robot del punto inicial y orientarlo de manera diferente a la ruta ideal, con el propósito de verificar su capacidad para tender hacia la ruta indicada y lograr alcanzar la meta.

Aunque el agente llegó a su destino, se puede observar en la Subfigura 45d que la trayectoria real fue menos constante en seguir la ruta de manera óptima en comparación con las pruebas anteriores. Esto puede deberse tanto a la distancia del robot con respecto al nodo inicial como a la distancia entre el nodo inicial y el final. Asimismo, en las subfiguras 45a, 45b y 45c se hace notar que el controlador fue capaz de ajustar la velocidad angular y lineal para el control de los motores del robot, permitiendo así que se alcanzará el destino.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

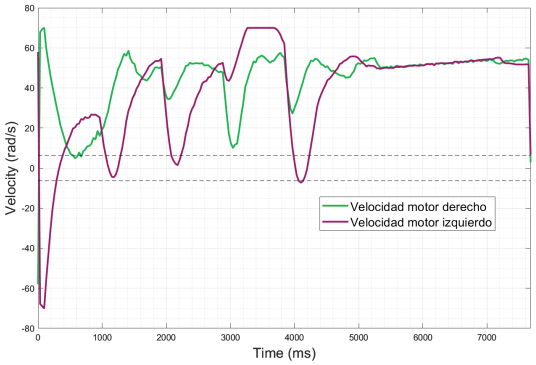
(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

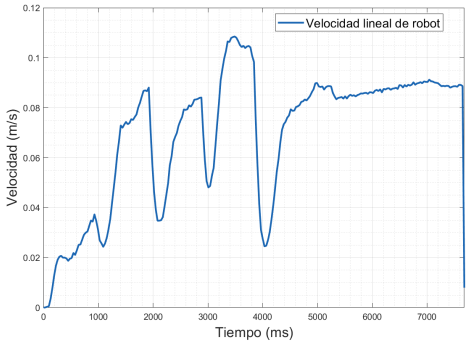
(e) Posición 3 de trayectorias

(f) Posición 4 de trayectorias

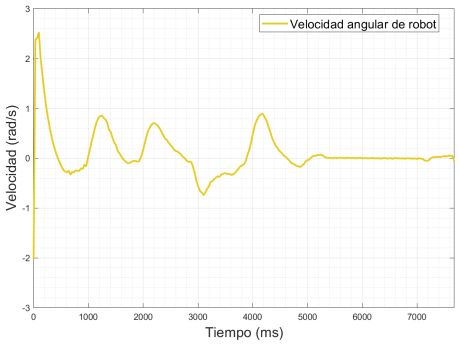
Figura 44: Trayectoria generada en físico para prueba con parámetros del Cuadro 19.



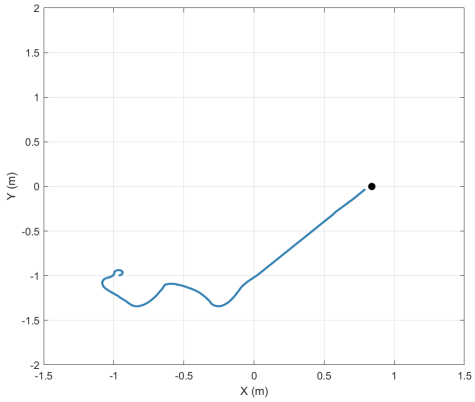
(a) Velocidad de motores del robot.



(b) Velocidad lineal del robot.



(c) Velocidad angular del robot.



(d) Trayectoria realizada por el robot en físico

Figura 45: Trayectoria generada en físico para prueba con parámetros del Cuadro 19.

Por último, para evaluar la capacidad del algoritmo de generar rutas completamente verticales, se seleccionaron los nodos 15 y 65 como puntos de inicio y fin. Los parámetros utilizados se encuentran en el Cuadro 20. Como se puede verificar en la Figura 46, el algoritmo tiende a generar rutas por medio de diagonales en lugar de por las líneas verticales y horizontales de la cuadrícula.

Descripción	**Ajuste seleccionado**
Controlador	PID con acercamiento exponencial
Cantidad de hormigas	75
Iteración máx.	150
Tasa de evaporación (ρ)	0.85
Peso de feromona (α)	1.3
Peso de costo de link (β)	0.7
Constante de regulación (Q)	1.5
% de hormigas (ϵ)	0.9

Cuadro 20: Parámetros de ACO para prueba 3.

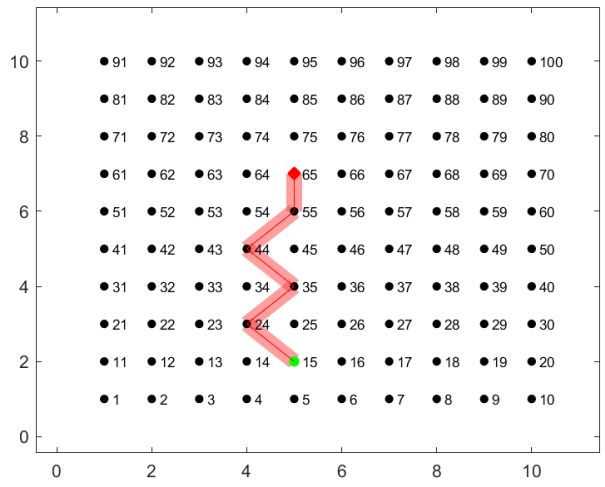
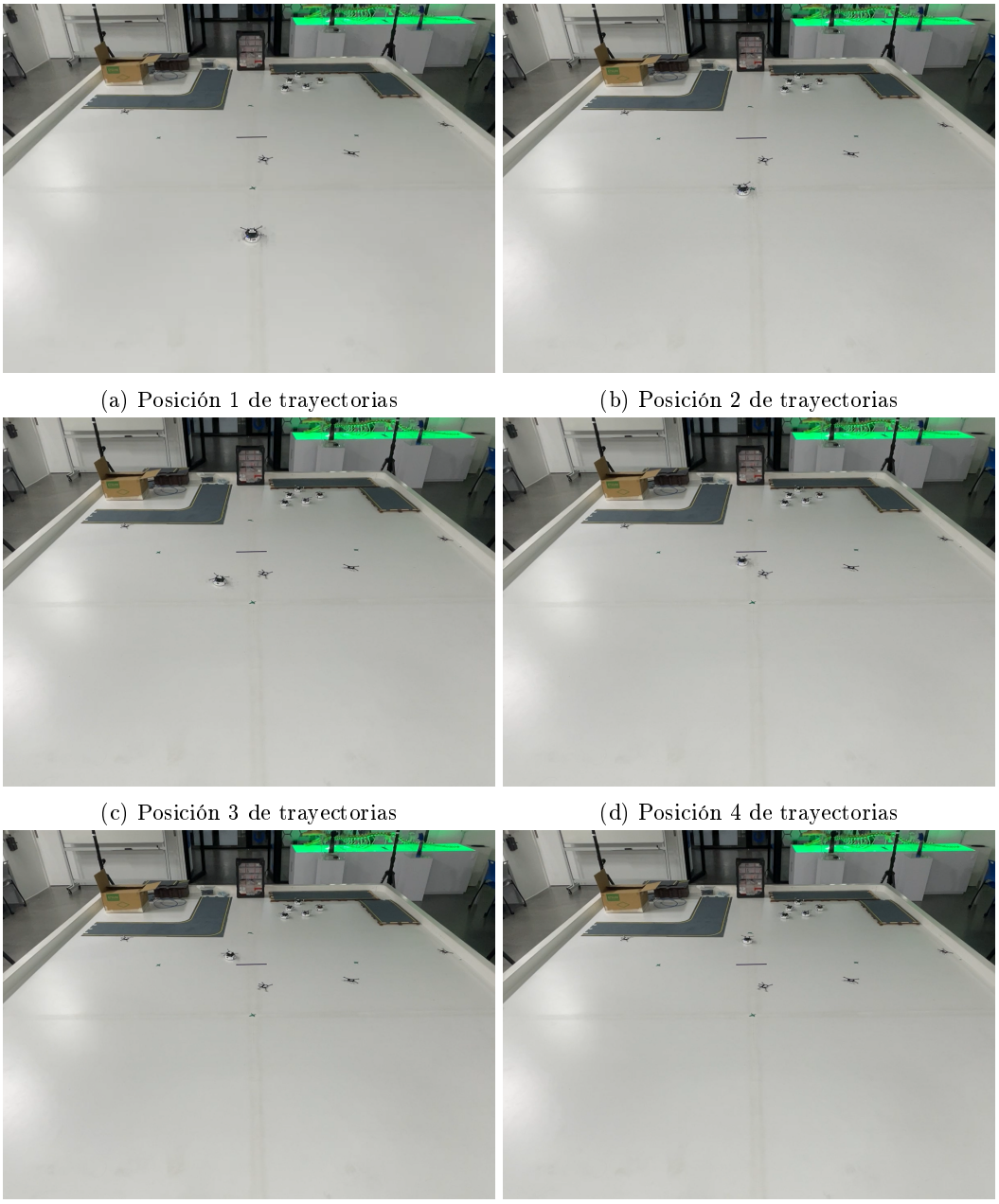


Figura 46: Trayectoria encontrada por ACO con ajustes del Cuadro 20.

Así como en los casos anteriores, el agente fue capaz de seguir la trayectoria establecida por el algoritmo sin ningún problema. En la Figura 47 se puede observar la ruta real que siguió el agente robótico y como esta se asemeja a la encontrada por el ACO. Nuevamente se probó a colocar el robot en un punto no tan cercano al inició de la trayectoria, pero en este caso no generó ningún tipo de complicación como se hace notar en la Subfigura 48d. Esto puede deberse a que el robot estaba orientado de mejor forma para llegar al inicio de la ruta, como se ve en la Subfigura 47a.

De igual forma podemos comprobar en las subfiguras 48a , 48b y 48c que las velocidades, tanto las calculadas por el controlador como las enviadas a los motores, se mantienen regulares, lo que nos indica un funcionamiento adecuado del controlado y explica el correcto seguimiento de la trayectoria requerida.



(a) Posición 1 de trayectorias

(b) Posición 2 de trayectorias

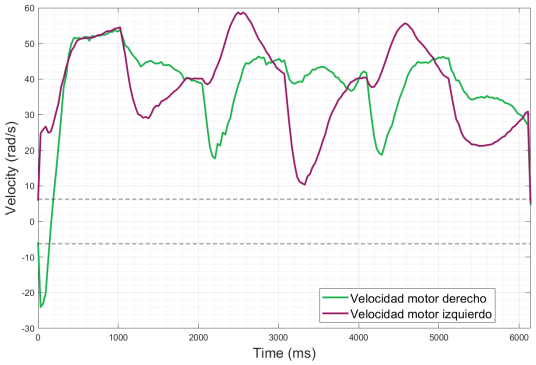
(c) Posición 3 de trayectorias

(d) Posición 4 de trayectorias

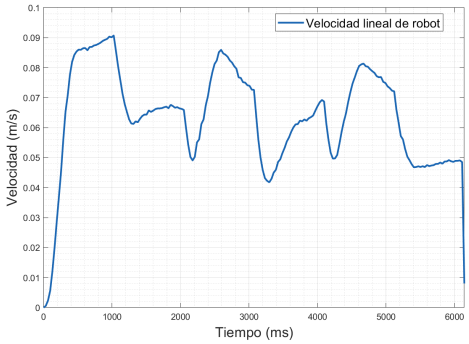
(e) Posición 3 de trayectorias

(f) Posición 4 de trayectorias

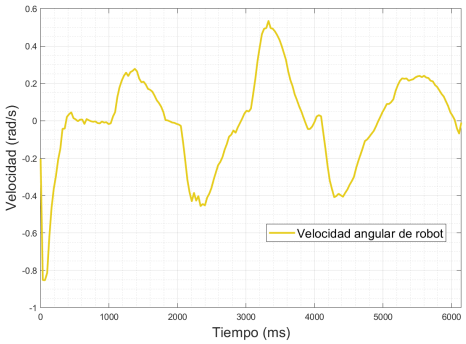
Figura 47: Trayectoria generada en físico para prueba con parámetros del Cuadro 20.



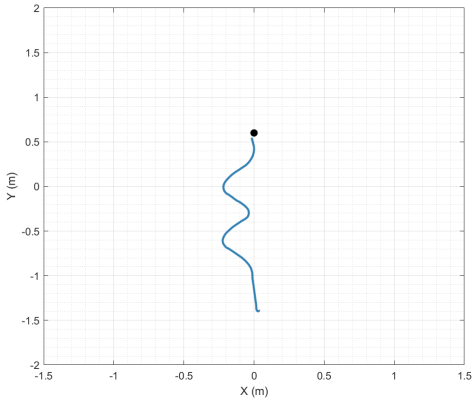
(a) Velocidad de motores del robot.



(b) Velocidad lineal del robot.



(c) Velocidad angular del robot.



(d) Trayectoria realizada por el robot en físico

Figura 48: Trayectoria generada en físico para prueba con parámetros del Cuadro 20.

CAPÍTULO 10

Conclusiones

- El sistema OptiTrack permite obtener de forma exitosa las posiciones y orientaciones necesarias para sustituir los sensores simulados en ambos algoritmos.
- Se ha demostrado que es posible reajustar el algoritmo de MPSO para que se ejecute de forma centralizada y en simultáneo para configuraciones multi-agente, tomando en cuenta que se necesita un mayor tiempo para cada ejecución del algoritmo.
- La paralelización del código en la sección de control de los agentes para el algoritmo de PSO posibilitó la validación con un mayor número de robots en comparación con el uso de ciclos *for* convencionales o la ejecución individual de cada controlador dentro del mismo archivo en MATLAB.
- El algoritmo de ACO demostró ser capaz de generar trayectorias satisfactorias para la mesa de pruebas del Robotat, ajustándose a los límites físicos de la mesa para evitar colisiones con los bordes y perdida de detección por parte de las cámaras de OptiTrack.
- El algoritmo de ACO demuestra una mayor eficacia en corregir discrepancias entre el nodo de inicio y la posición inicial del robot cuando este se encuentra orientado hacia dicho punto de inicio. Además, muestra la capacidad para manejar variaciones en la magnitud de la distancia entre estos puntos, sin que afecte significativamente su desempeño.
- El controlador PID con acercamiento exponencial demuestra ser capaz de ejecutar los movimientos correspondientes a las trayectorias generadas por los algoritmos de PSO y ACO. Este controlador logra mantener velocidades relativamente constantes y asegura la llegada exitosa al destino, incluso en trayectorias que no son totalmente directas.

CAPÍTULO 11

Recomendaciones

- En fases futuras, se sugiere evaluar casos en que los algoritmos sean capaces de tomar en cuenta la ubicación de obstáculos y poder evitarlos para llegar a su destino. Esto mejoraría la seguridad y la confiabilidad de los algoritmos, especialmente en entornos complejos.
- Se propone explorar la posibilidad de integrar los algoritmos de control con el sistema de captura de movimiento, de manera que puedan detectar cuando un agente se encuentra fuera de la zona de lectura. En tal situación, se sugiere implementar una lógica que permita al agente realizar un giro hacia la dirección necesaria para regresar a una posición válida dentro del área de detección.
- En fases futuras, se recomienda explorar la posibilidad de mantener el algoritmo de MPSO descentralizado, como en las simulaciones en [5], para mejorar la autonomía y velocidad de la ejecución.
- En las siguientes fases de este proyecto, se aconseja evaluar la viabilidad de llevar a cabo las iteraciones para encontrar la trayectoria en el algoritmo ACO utilizando los robots Pololu 3pi+ disponibles.
- Se sugiere verificar si es posible optimizar las trayectorias generadas por el algoritmo de ACO en [8] para permitir rutas más directas entre el nodo inicial y el nodo final que deberá seguir un agente físico.

CAPÍTULO 12

Bibliografía

[1] M. Dorigo, G. Theraulaz y V. Trianni, «Swarm Robotics: Past, Present, and Future [Point of View],» *Proceedings of the IEEE*, vol. 109, n.º 7, págs. 1152-1165, 2021. DOI: 10.1109/JPROC.2021.3072740.

[2] D. Pickem, P. Glotfelter, L. Wang et al., «The Robotarium: A remotely accessible swarm robotics research testbed,» en *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, págs. 1699-1706. DOI: 10.1109/ICRA.2017.7989200.

[3] Y. Tan y Z.-y. Zheng, «Research Advance in Swarm Robotics,» *Defence Technology*, vol. 239, mar. de 2013. DOI: 10.1016/j.dt.2013.03.001.

[4] R. A. Lima, «Implementación y validación de algoritmos de robótica de enjambre en plataformas móviles en la nueva mesa de pruebas del laboratorio de robótica de la UVG,» Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.

[5] A. S. Aguilar, «Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO),» Tesis de licenciatura, Universidad Del Valle de Guatemala, 2019.

[6] A. Aguilar, M. Zea y L. Rivera, «PSO Trajectory Planner Using Kinematic Controllers that Ensure Smooth Differential Robot Velocities,» dic. de 2020, págs. 481-488. DOI: 10.1109/SSCI47803.2020.9308498.

[7] A. D. Maas, «Implementación y Validación del Algoritmo de Robótica de Enjambre Particle Swarm Optimization en Sistemas Físicos,» Tesis de licenciatura, Universidad Del Valle de Guatemala, 2021.

[8] G. Iriarte, «Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica,» Tesis de licenciatura, Universidad Del Valle de Guatemala, 2021.

[9] W. A. Sierra, «Implementación y Validación del Algoritmo de Robótica de Enjambre Ant Colony Optimization en Sistemas Físicos,» Tesis de licenciatura, Universidad Del Valle de Guatemala, 2021.

[10] M. M. Shahzad, Z. Saeed, A. Akhtar et al., «A Review of Swarm Robotics in a NutShell,» *Drones*, vol. 7, n.º 4, 2023, ISSN: 2504-446X. dirección: https://www.mdpi.com/2504-446X/7/4/269.

[11] M. Schranz, M. Umlauft, M. Sende y W. Elmenreich, «Swarm Robotic Behaviors and Current Applications,» *Frontiers in Robotics and AI*, vol. 7, 2020, ISSN: 2296-9144. DOI: 10.3389/frobt.2020.00036. dirección: https://www.frontiersin.org/articles/10.3389/frobt.2020.00036.

[12] M. Brambilla, E. Ferrante, M. Birattari y M. Dorigo, «Swarm Robotics: A Review from the Swarm Engineering Perspective,» *Swarm Intelligence*, vol. 7, págs. 1-41, mar. de 2013. DOI: 10.1007/s11721-012-0075-2.

[13] R. Eberhart y J. Kennedy, «A new optimizer using particle swarm theory,» en *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, págs. 39-43. DOI: 10.1109/MHS.1995.494215.

[14] M. N. Ab Wahab, S. Nefti-Meziani y A. Atyabi, «A Comprehensive Review of Swarm Optimization Algorithms,» *PLOS ONE*, vol. 10, n.º 5, págs. 1-36, mayo de 2015. DOI: 10.1371/journal.pone.0122827. dirección: https://doi.org/10.1371/journal.pone.0122827.

[15] R. Poli, J. Kennedy y T. Blackwell, «Particle Swarm Optimization: An Overview,» *Swarm Intelligence*, vol. 1, oct. de 2007. DOI: 10.1007/s11721-007-0002-0.

[16] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon y A. Abraham, «Inertia Weight strategies in Particle Swarm Optimization,» en *2011 Third World Congress on Nature and Biologically Inspired Computing*, 2011, págs. 633-640. DOI: 10.1109/NaBIC.2011.6089659.

[17] B. Obayyanahatti e Y. Shi, «Comparing inertial weights and Constriction factor in particle swarm optimization,» vol. 1, feb. de 2000, 84-88 vol.1, ISBN: 0-7803-6375-2. DOI: 10.1109/CEC.2000.870279.

[18] M. Dorigo, M. Birattari y T. Stutzle, «Ant colony optimization,» *IEEE Computational Intelligence Magazine*, vol. 1, n.º 4, págs. 28-39, 2006. DOI: 10.1109/MCI.2006.329691.

[19] J.-L. Deneubourg, S. Aron, S. Goss y J. Pasteels, «The Self-Organizing Exploratory Pattern of the Argentine Ant,» *J. Insect Behav.*, vol. 3, pág. 159, mar. de 1990. DOI: 10.1007/BF01417909.

[20] P. Corke, *Robotics, Vision and Control - Fundamental Algorithms in MATLAB®* (Springer Tracts in Advanced Robotics). Springer, 2011, vol. 73, págs. 1-495, ISBN: 978-3-642-20143-1. dirección: http://dblp.uni-trier.de/db/series/star/index.html#Corke11.

[21] R. Siegwart, I. Nourbakhsh y D. Scaramuzza, *Introduction to Autonomous Mobile Robots, second edition* (Intelligent Robotics and Autonomous Agents series). MIT Press, 2011, ISBN: 9780262015356. dirección: https://mitpress.mit.edu/9780262015356/introduction-to-autonomous-mobile-robots/.

[22] K. Lynch y F. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017, ISBN: 9781316609842. dirección: https://hades.mech.northwestern.edu/images/4/4e/MR-tablet.pdf.

[23] *Pololu 3pi+ 32U4 User's Guide*, English, Pololu Corporation, 2022, 86 págs. dirección: https://www.pololu.com/docs/pdf/0J83/3pi_plus_32u4.pdf, 2022.

[24] I. NaturalPoint, *PrimeX 41*, 2023. dirección: https://optitrack.com/cameras/primex-41/.

[25] F. Al-Dhief, N. Sabri, N. M. Abdul Latiff et al., «Performance comparison between TCP and UDP protocols in different simulation scenarios,» *International Journal of Engineering & Technology*, vol. 7, n.º 4, págs. 172-176, 2018. DOI: 10.14419/ijet.v7i4.12832.

[26] G. Cai, B. M. Chen y T. H. Lee, *Unmanned Rotorcraft Systems*. Springer London, 2011. DOI: 10.1007/978-0-85729-635-1. dirección: https://doi.org/10.1007%2F978-0-85729-635-1.

[27] Cyberbotics. «Cyberbotics - Webots Robot Simulator.» Accessed on August 18, 2023. (2023), dirección: https://cyberbotics.com.

[28] Cyberbotics. «How to adapt your world or PROTO to Webots R2022a.» Accessed on August 18, 2023. (2023), dirección: https://github.com/cyberbotics/webots/wiki/How-to-adapt-your-world-or-PROTO-to-Webots-R2022a#automatic-conversion-to-enu-flu.

[29] T. M. Inc., *MATLAB*, 2023. dirección: https://www.mathworks.com/products/matlab.html.

[30] The MathWorks Inc., *Vectorization*, 2023. dirección: https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html.

[31] The MathWorks Inc., *Parallel Computing*, 2023. dirección: https://www.mathworks.com/products/parallel-computing.html.

[32] C. Perafán, «Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica,» Tesis de licenciatura, Universidad Del Valle de Guatemala, 2021.

CAPÍTULO 13

Anexos

13.1. Código y proceso de ejecución de los algoritmos

Para descargar los código para los algoritmos asociados a esta investigación, puede ingresar al siguiente link: https://github.com/men18023/Jonathan-Menendez-Swarm-Robotics.

Aquí encontrará las carpetas **MPSO_3pi** y **ACO_3pi** que contienen los controladores y archivos necesarios para realizar las pruebas en la mesa del Robotat. A continuación se da el proceso a seguir para ejecutar cada algoritmo:

13.1.1. PSO

- Descargar la carpeta **MPSO_3pi** que contiene los archivos **MPSO_3pi.m** y las funciones de controladores para los robots y comandos para comunicación con el sistema OptiTrack
- Asegurarse que el servidor del Robotat y el sistema de OptiTrack estén en funcionamiento
- Elegir los robots Pololu 3pi+ que se deseen utilizar. Se debe elegir una secuencia continua entre los robots disponibles del 1 al 10
- Confirmar que los microcontroladores ESP32 cuente con la configuración para comunicarse con el servidor.
- Colocar los marcadores del sistema OptiTrack en cada uno de los robots. Verificar que el número entre el robot, ESP32 y marcador coincidan
- Encender y colocar los agentes en las posiciones iniciales deseadas
- Abrir en MATLAB la carpeta descargada en el primer paso

- Correr el código del archivo MPSO_pololu, donde debe ajustar algunas variables para indicar que robots fueron seleccionados y los parámetros del algoritmo que se deseen probar

13.1.2. ACO

- Descargar la carpeta **ACO_3pi** que contiene los archivos **ACO_3pi.m** y las funciones de controladores para los robots y comandos para comunicación con el sistema OptiTrack
- Asegurarse que el servidor del Robotat y el sistema de OptiTrack estén en funcionamiento
- Elegir el robot Pololu 3pi+ que se desee utilizar.
- Confirmar que el microcontrolador ESP32 cuente con la configuración para comunicarse con el servidor.
- Colocar el marcador del sistema OptiTrack en el robot. Verificar que el número entre el robot, ESP32 y marcador coincidan
- Abrir en MATLAB la carpeta descargada en el primer paso
- Correr el archivo ACO.m, donde se debe elegir los nodos de inicio y final para encontrar la trayectoria por medio del algoritmo
- Iniciar el agente en una posición cercana al nodo inicial seleccionado en el paso previo, idealmente
- Correr el código del archivo ACO_pololu, donde debe configurar los parámetros que se deseen probar