# Optimización del algoritmo de robótica de enjambre *Particle Swarm Optimization* para su implementación con agentes robóticos físicos en escenarios con obstáculos en el ecosistema Robotat

Ana Luisa Barrientos Flores

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Optimización del algoritmo de robótica de enjambre *Particle Swarm Optimization* para su implementación con agentes robóticos físicos en escenarios con obstáculos en el ecosistema Robotat

Trabajo de graduación presentado por Ana Luisa Barrientos Flores para optar al grado académico de Licenciada en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____

M. Sc. Carlos Esquit

Tribunal Examinador:

(f) _____

M.Sc. Carlos Esquit

(f) _____

M. Sc. Miguel Enrique Zea Arenales

(f) _____

Ing. Kurt Emmanuel Kellner

Fecha de aprobación: Guatemala, 13 de febrero de 2025.

# Índice

# Lista de figuras

# Lista de cuadros

# Resumen

La robótica de enjambre, inspirada en el comportamiento colectivo de animales como peces o aves, permite que un conjunto de partículas trabaje de forma colaborativa para alcanzar un objetivo en común. En la Universidad del Valle de Guatemala (UVG), se han desarrollado diversas investigaciones basadas en el algoritmo *Particle Swarm Optimization* (PSO), el cual evalúa posiciones óptimas mediante funciones de costo que determinan las mejores posicones encontradas por el enjambre de robots.

En esta investigación, se presenta la optimización del algoritmo *Modified Particle Swarm Optimization* (MPSO) desarrollado en fases anteriores, para su implementación con agentes robóticos físicos en escenarios más complejos con obstáculos en el ecosistema Robotat. La optimización se llevó a cabo mediante técnicas de vectorización, paralelización y lenguaje de programación Python, buscando mejorar el tiempo de convergencia (tiempo que tardan los agentes robóticos en llegar a la meta) y las trayectorias generadas.

Las pruebas se realizaron utilizando agentes robóticos Pololu 3pi+ y las funciones de costo Sphere, Booth y Schaffer. Asimismo, se evaluaron y compararon las trayectorias generadas con las originales. El algoritmo PSO también fue implementado como planificador de trayectorias mediante el uso de campos potenciales artificiales para la evasión de obstáculos, implementado en dos entornos: uno en tiempo real en el ecosistema Robotat y otro en el simulador Webots.

Los resultados más relevantes presentan mejoras en el tiempo de convergencia y las trayectorias generadas. De igual forma, la implemetación del algoritmo PSO, como planificador de trayectorias, demostró un mejor desempeño en la evasión de obstáculos y el tiempo requerido para llegar a la meta.

Palabras clave: PSO, robótica de enjambre, evasión de obstáculos, funciones de costo.

# Abstract

Swarm robotics, inspired by the collective behavior of animals such as fish or birds, enables a set of particles to work collaboratively to achieve a common goal. At the University of Valle de Guatemala, several research projects have been developed based on the Particle Swarm Optimization (PSO) algorithm, which evaluates optimal positions using cost functions to determine the best positions found by the swarm of robots.

This research presents the optimization of the *Modified Particle Swarm Optimization* (MPSO) algorithm, developed in previous phases, for implementation with physical robotic agents in more complex scenarios, including obstacles within the Robotat ecosystem. The optimization was carried out through techniques such as vectorization, parallelization, and the use of the Python programming language, aiming to improve convergence time (the time it takes for the robotic agents to reach the target) and the generated trajectories.

Tests were conducted using Pololu 3pi+ robotic agents and the cost functions Sphere, Booth, and Schaffer. Additionally, the generated trajectories were evaluated and compared to the original ones. The PSO algorithm was also implemented as a trajectory planner using artificial potential fields for obstacle avoidance, tested in two environments: one in real-time in the Robotat ecosystem and another in the Webots simulator.

The most significant results showed improvements in convergence time and the generated trajectories. Furthermore, the implementation of the PSO algorithm as a trajectory planner demonstrated better performance in obstacle avoidance and reduced time to reach the target.

Keywords: Particle Swarm Optimization, swarm robotics, trajectory planner, obstacle avoidance, cost functions.

# CAPÍTULO 1

## Introducción

Los algoritmos de optimización son herramientas fundamentales en diversas áreas, van desde el aprendizaje reforzado hasta la ingeniería. Su principal objetivo es encontrar la mejor solución a un problema, explorando posibles soluciones, para seleccionar la que minimice o maximice la función objetivo. El algoritmo *Particle Swarm Optimization* (PSO), es una técnica iterativa de optimización inspirada en el comportamiento social de los animales como pájaros o peces. Este método ha ganado popularidad gracias a su simplicidad, flexibilidad y robustez.

En investigaciones realizadas en la Universidad del Valle de Guatemala, se ha implementado el algoritmo PSO como planificador para encontrar la mejor ruta hacia una meta. Asimismo, este algoritmo se ha utilizado para la evasión de obstáculos mediante el uso de campos potenciales artificiales.

Este documento presenta las mejoras realizadas al algoritmo *Modified Particle Swarm Optimization* (MPSO), con el objetivo de optimizar su rendimiento y eficiencia. Se implementaron mejoras en aspectos como la vectorización, la paralelización, otro lenguaje de programación y protocolos de comunicación. Además, se compararon los tiempos de convergencia de cada mejora con la versión original del algoritmo para evaluar su rendimiento.

El objetivo final fue implementar el algoritmo optimizado en escenarios más complejos, incluyendo obstáculos en el ecosistema Robotat. La evasión de obstáculos se logró mediante el uso de campos potenciales artificiales, lo que permitió incorporar obstáculos rectángulares en una serie de experimentos. También, se desarrolló un planificador de trayectorias que utilizó el algoritmo PSO junto con los campos potenciales artificiales para mejorar la eficiencia en la planificación de rutas libres de obstáculos.

CAPÍTULO 2

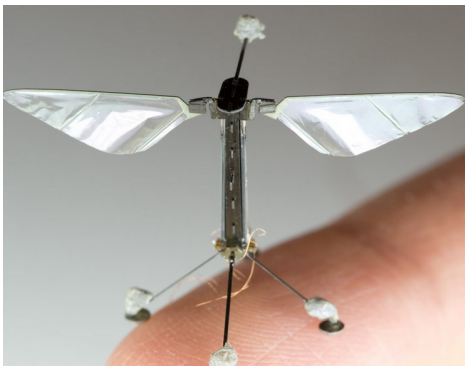
# Antecedentes

## 2.1. Robótica de enjambre

La robótica de enjambre es un campo que combina ingeniería, inteligencia artificial y biología para simular el comportamiento de enjambres naturales por medio de sistemas robóticos [1]. Estos sistemas robóticos tienen la capacidad de trabajar de forma coordinada y se comunican entre sí para lograr objetivos en común.

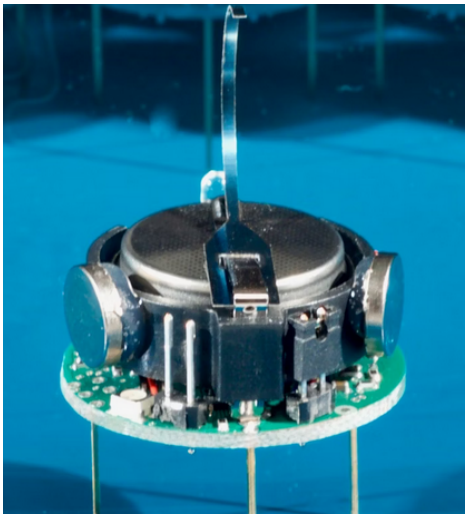
Entre las investigaciones que se han desarrollado se encuentran, los microrobots del laboratorio *Microrobotics* de Harvard [2], llamados *RoboBees* ilustrados en la Figura 1. Estos robots están inspirados en la biología de las abejas, el objetivo era desarrollar vehículos microaéreos autónomos capaces de realizar vuelos autónomos y autodirigidos y de lograr un comportamiento coordinado en grandes grupos.

**Figura 1:** *RoboBees desarrollados en el laboratorio de Microrobotics [2].*



De igual manera investigadores de *Wyss Institute* en Boston, Massachusetts [3], han desarrollado sistemas robóticos basados en robótica de enjambres llamados *Kilobots* ilustrados en la Figura 2. Dichos robots fueron creados para colaborar entre sí y llegar a una meta en específico. Estos pueden ser programados para mostrar comportamientos de enjambres complejos, como búsqueda de alimento y sincronización inspirada en luciérnagas. El algoritmo desarrollado permite el autoensamblaje programable en grandes enjambres, en donde el usuario puede solicitar la figura que realizarán todos los robots.

**Figura 2:** *Kilobots desarrollados por el Instituto de Investigaciones Wyss Institute en Boston [3].*



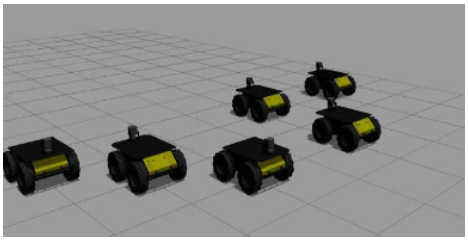
## 2.2. Aplicaciones de la robótica de enjambre

Dentro de las aplicaciones más importantes de la robótica de enjambre se encuentran búsqueda y rescate, medicina, mapeo de entornos, exploración de terrenos, entre otros.

### 2.2.1. Búsqueda y rescate

Los investigadores del Laboratorio *MIT media lab* [4] desarrollaron un método que permite encapsular misiones de robótica de enjambres cooperativas en una estructura conocida como *Merkle tree*. Este método permite que los robots puedan cooperar y realizar tareas sin tener mucho conocimiento sobre los objetivos de la misión, haciendo que los robots deban probar su integridad con los demás mediante el intercambio de pruebas criptográficas (*blueprint*). En esta investigación se utilizaron dos tipos de misiones *foraging* y formación de laberintos, en donde los resultados demuestran la factibilidad de usar el *Merkle tree* como un mecanismo de colaboración para sistemas de robótica de enjambres.

**Figura 3:** *RDPSO aplicado a búsqueda y rescate [5].*



De la misma manera, otra investigación realizada por laboratorios AMMACHI en India [5], fue la implementación de *Robotic Darwinian Particle Swarm Optimization* (RDPSO) ilustrado en la Figura 3. Este algoritmo se basa en la exploración por medio de múltiples objetivos. Esta investigación presentan simulaciones utilizando *Robot Operating System* (ROS) y Gazebo, en donde se imitaron distintos escenarios del mundo real como identificación de víctimas mediante mapeo de intensidad de voz/sonido, localización de la fuente del incendio mediante mapeo de intensidad de temperatura, entre otros.

### 2.2.2. Mapeo y exploración de terrenos

En 2023 se publicó un artículo sobre la localización y mapeo simultáneos (SLAM) en *Swarm Robots* para la fusión de mapas y la generación uniforme de mapas usando ROS [6]. Este artículo propone un sistema multi-robot en donde robots individuales en el enjambre interactúan entre sí y contribuyen a la creación de un mapa a medida que se va mapeando el entorno. Este mapa puede ser utilizado por cualquier otro robot para explorar un entorno desconocido y realizar diferentes tareas como búsqueda y rescate.

### 2.2.3. Robótica de enjambre sistema de entregas

En 2021 se publicó un artículo sobre el uson de drones utilizando una arquitectura centralizada y descentralizada para trabajar de forma colaborativa en una aplicación de sistema de entregas [7]. Se investigaron ambos enfoques, los cuales se pusieron a prueba con diferentes experimentos. Una de las conclusiones de este artículo fue, que el uso del control descentralizado para reducir la comunicación excesiva entre los drones es más fluido y se propuso una arquitectura para el sistema de entrega de piezas utilizando el enfoque centralizado y descentralizado.

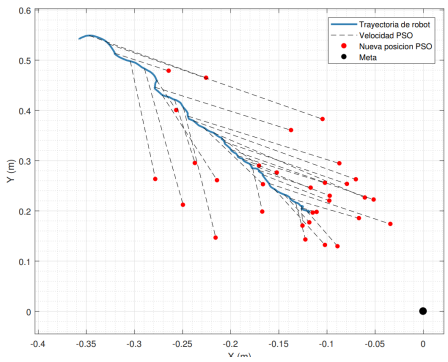
## 2.3. Megaproyecto Robotat

En la Universidad del Valle de Guatemala se ha ido desarrollando el proyecto llamado Robotat. Este proyecto consiste en un ecosistema para realizar experimentos con diferentes agentes robóticos. Su funcionamiento es por medio de una red de comunicación bi-direccional Wifi, la cual permite obtener las coordenadas y orientación de los agentes robóticos, proporcionadas por el sistema de captura Optitrack [8] y a su vez enviar información al servidor para conectarse al ecosistema y obtener la información [9].

## 2.4. Investigaciones realizadas en UVG sobre robótica de enjambre

En el trabajo de investigacón realizado por Aldo Aguilar [10] se implementó el algoritmo *Particle Swarm Optimization* (PSO) como planificador de trayectorias, ilustrado en la Figura 4. Se diseñó el algoritmo *Modified Particle Swarm Optimization* (MPSO) para encontrar la mejor trayectoria obtenida por el enjambre, la cuál posteriormente es ejecutada por robots diferenciales. El algoritmo fue adaptado para considerar las restricciones físicas y cinématicas de los robots. Además se utilizaron diferentes controladores para ejecutar las trayectoria encontradas. Como resultado se obtuvo que el controlador que generó trayectorias y velocidades más suaves fue el controlador TUC LQI.

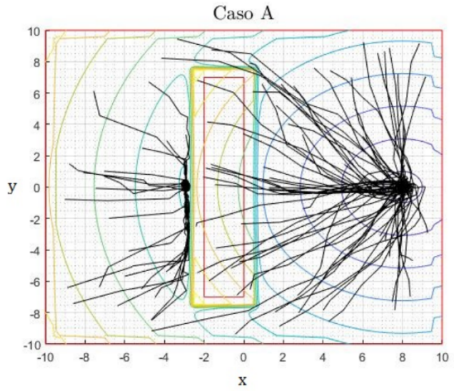
**Figura 4:** *Simulación de las trayectorias con controlador TUC-LQI [10].*




Posteriormente, en el trabajo de investigación llevado a cabo por Juan Cahueque [11] se implementó el algoritmo PSO en el contexto de búsqueda y rescate como se ilustra en la Figura 5.

Se definieron las ecuaciones de campos potenciales artificales de Choset y Kim para representar la zona de desastre mediante obstáculos. Después, se definieron los parámetros a utilizar mediante simulaciones del PSO, considerando el campo potencial total como la función de optimización.

**Figura 5:** *Simulación de búsqueda y rescate utilizando campos potenciales y el algoritmo PSO [10].*

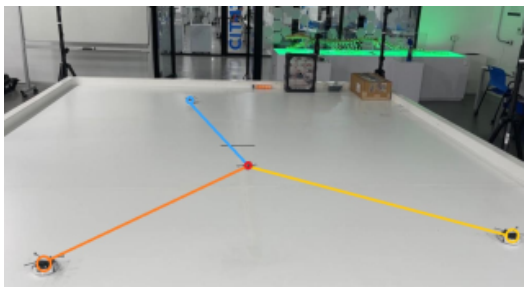



Posteriormente, en el trabajo de investigación llevado a cabo por Eduardo Santizo [12] se creó la herramienta llamada *Swarm Robotics Toolbox* la cuál permite simular el algoritmo PSO generado por [11] y los campos potenciales, utilizando cierta cantidad de partículas y robots diferenciales *e-pucks* con sus restricciones físicas.

Posteriormente, en el trabajo de investigación llevado a cabo por Alex Mass [13] se realizó la implementación de [10] con agentes robóticos físicos por medio del ordenador Raspberry Pi y el lenguaje de programación C.

Por último, en el trabajo llevado a cabo por Jonathan Menéndez [14] se implementaron los algoritmos de PSO de [10] y el algoritmo ACO de [15] en el ecosistema Robotat, utilizando las plataformas móviles Pololu 3Pi+ Figura 6. Uno de los objetivos fue migrar los algoritmos PSO y ACO para utilizarlo en plataformas móviles y poder comparar el rendimiento con los algoritmos realizados en fases anteriores. El alcance de este trabajo fue la realización de pruebas físicas con los algoritmos centralizados para recorrer las trayectorias con los agentes robóticos. El limitante de este trabajo fue que solo se contaban con diez agentes robóticos y que las pruebas sólo se pueden realizar dentro de la plataforma del Robotat.

**Figura 6:** *Implementación física del algoritmo PSO [14].*



Se migró el algoritmo MPSO de forma centralizada al lenguaje de Matlab y se ajustó la forma de adquirir las posiciones y orientaciones de los robots para que se orientaran en la misma posición. También se recrearon las pruebas simuladas de la fase anterior para evaluar el rendimiento del algoritmo PSO utilizando diferentes funciones objetivo para encontrar la solución óptima. Es importante considerar que el tiempo de ejecución aumenta conforme aumentan la cantidad de agentes. Por último, se realizaron pruebas para verificar el desempeño del algoritmo ACO y encontrar los ajustes necesarios tomando en cuenta que el algoritmo puede verse afectado por la distancia entre el nodo destino y nodo final.

Dentro de los resultados más importantes se obtuvo que al ejecutar el algoritmo PSO de forma centralizada y en simultáneo con muchos agentes se necesita mayor tiempo de ejecución para el algoritmo. Asimismo, los algoritmos ACO y MPSO demostraron ser capaces de generar trayectorias satisfactorias para la mesa de pruebas, utilizando el controlador PID con acercamiento exponencial para mantener las velocidades constantes.

# CAPÍTULO 3

## Justificación

La robótica de enjambre es un campo de estudio que se inspira en los comportamientos de animales como hormigas, peces, pájaros o abejas. Esta busca replicar como el conjunto de individuos interactúa con el entorno, se comunican y cooperan entre sí para lograr un objetivo en común. Esta permite a los individuos poder adaptarse dentro del ambiente por medio de aprendizajes propios y colectivos. La importancia de este campo es que cada agente en el enjambre es relativamente simple, pero al trabajar de forma colaborativa, se logran resultados impresionantes [16].

En la Universidad del Valle de Guatemala se han realizado distintos proyectos de robótica de enjambre durante varios años. Uno de los más recientes consistió en validar de forma física dos de los algoritmos más utilizados, como el *Particle Swarm Optimization* (PSO) y *Ant Colony Optimization* (ACO). La validación de los algoritmos se realizó en el ecosistema Robotat utilizando agentes robóticos Pololu 3Pi+. Las limitantes de este proyecto fueron que no se contaban con suficientes agentes robóticos para realizar los experimentos. Asimismo, al utilizar muchos agentes robóticos, se notó que el tiempo de ejecución de los algoritmos era bastante lento.

En consecuencia, el presente trabajo buscó continuar con la línea de investigación de robótica de enjambre mediante la implementación del algoritmo PSO desarrollado en la fase anterior. Se buscó mejorar el rendimiento y tiempo de convergencia de los agentes robóticos mediante métodos de vectorización, uso de otro lenguaje de programación y paralelización. Finalmente, se desarrolló un planificador para evasión de obstáculos con el algoritmo PSO mejorado mediante el uso de campos potenciales artificiales.

# CAPÍTULO 4

# Objetivos

### 4.0.1. Objetivo general

Optimizar la implementación del algoritmo *Particle Swarm Optimization* desarrollado en la fase anterior para agentes móviles y validarlo en más escenarios y con obstáculos, en el ecosistema Robotat.

### 4.0.2. Objetivos específicos

- Evaluar la implementación del algoritmo PSO desarrollado anteriormente e identificar ineficiencias y puntos de mejora en el código, lenguaje de programación y métodos de comunicación.
- Evaluar la posibilidad de implementar paralelización para la optimización del algoritmo.
- Implementar las mejoras identificadas y evaluar el rendimiento del algoritmo PSO mejorado en escenarios similares a los probados en fases previas.
- Validar el algoritmo mejorado en escenarios más complejos, incluyendo obstáculos, en el ecosistema Robotat.

# CAPÍTULO 5

# Alcance

El alcance de este trabajo de graduación se centró en la optimización de la implementación del algoritmo *Modified Particle Swarm Optimization* (MPSO) desarrollado en fases anteriores, mediante la implementación de técnicas como la vectorización, paralelización y la migración del algoritmo al lenguaje de programación Python.

Se llevaron a cabo experimentos tanto en entornos físicos como en simulación, utilizando las funciones de costo Sphere, Booth y Schaffer. En estos experimentos, se evaluaron los tiempos de convergencia promedio para cada mejora implementada, con el objetivo de verificar si hubo una mejora en el rendimiento del algoritmo MPSO. Asimismo, se migró el algoritmo MPSO al lenguaje Python para comparar los tiempos de convergencia con la versión original en el simulador Webots.

Se desarrolló un algoritmo basado en el PSO y campos potenciales artificiales para la evasión de obstáculos. Los primeros experimentos se realizaron en tiempo real, durante los cuales se probaron y ajustaron los parámetros para los campos potenciales. Finalmente, se llevó a cabo un planificador de trayectorias basado en el algoritmo PSO y los campos potenciales artificiales.

Entre las limitantes que se presentaron durante el desarrollo de este trabajo, destacan la alta demanda de los agentes robóticos Pololu 3pi+ y las restricciones de espacio en la mesa del Robotat. Además, no se logró implementar un nuevo protocolo de comunicación debido a la falta de infraestructura necesaria.

# CAPÍTULO 6

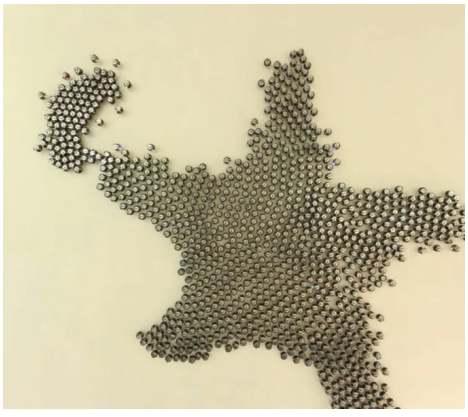
# Marco teórico

## 6.1. Robótica de enjambre

Los enjambres son definidos como una gran cantidad de cosas animadas (animales, personas) o inanimadas (vehículos, robots) que van en conjunto y usualmente están en movimiento.

La robótica de enjambre está inspirada en el comportamiento biológico de distintos enjambres como bandadas de pájaros, enjambres de hormigas o colmenas de abejas, en donde el enjambre trabaja en conjunto, como un único sistema cohesivo, para lograr un objetivo Figura 7. En el caso de la robótica se puede considerar a un enjambre como conjuntos de robots, drones, vehículos terrestres, que trabajan y se comunican entre sí para realizar una tarea en específico [17].

**Figura 7:** *Enjambre robótico [3].*



### 6.1.1. Comportamientos colectivos

Los comportamientos colectivos son un conjunto de comportamientos básicos de enjambres de robots que pueden ser combinados para crear comportamientos colectivos complejos. Entre ellos se encuentran:

- **Comportamientos de organización espacial:** este tipo de comportamientos se centra en como los robots se distribuyen y organizan en el espacio [18].
  - **Agregación:** tiene como objetivo agrupar a los robots en una región especifica del entorno, y permite que los robots se reúnan para interactuar entre sí.
  - **Formación de patrones:** la formación de patrones es un comportamiento que tiene como objetivo posicionar a los robots en el espacio utilizando un patrón definido.
  - **Formación de cadenas:** los robots de forma individual dentro del entorno crean cadenas que conectan dos posiciones. Después esta cadena es utilizada por otros robots como ayuda para la navegación.
  - **Autoensamblaje y morfogénesis:** es el proceso por el cual los robots se conectan físicamente entre ellos. Cuando los robots conectados forman un patrón o forma particular, se conoce como morfogénesis. Esta es utilizada cuando una estructura en particular permite al enjambre ejecutar una tarea específica.
  - **Agrupación y ensamblaje de objetos:** son los comportamientos en donde los robots crean agregados de objetos. La diferencia entre agrupación y ensamblaje de objetos es en, el primero los agregados son grupos de objetos desconectados, en cambio, en el ensamblaje, los objetos deben estar conectados por algún tipo de vínculo físico.
- **Comportamientos de navegación:** este tipo de comportamientos se centra en la coordinación de los movimientos del enjambre de robots [18].
  - **Exploración colectiva:** la colaboración colectiva tiene como objetivo explorar el entorno o algunas partes de él. En ocasione,s el enjambre se basa en robots

**Figura 8:** *Ejemplo de agrupación y ensamblaje de objetos de organización espacial [19].*

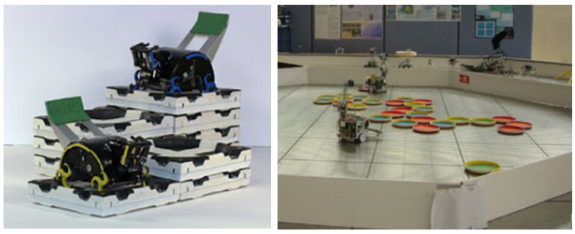
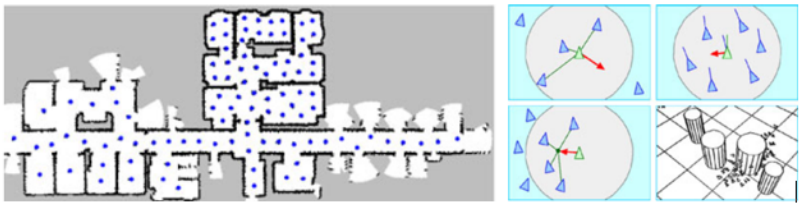


**Figura 9:** *Ejemplo de comportamientos de navegación [19].*



estáticos que actúan como puntos de referencia para guiar la navegación de los movimientos de los otros.

- **Movimiento coordinado:** también conocido como *flocking*, los robots se mueven con cierta formación a través del entorno, similar a las bandadas de pájaros o bancos de peces.
- **Transporte colectivo:** se refiere a un conjunto de comportamientos en donde el objetivo del enjambre es mover objetos de forma cooperativa desde una posición a otra.

- **Toma de decisiones colectiva:** la toma de decisiones colectiva se centra en como un enjambre de robots toma ciertas decisiones. Existen dos categorías de situaciones que pueden requerir la toma de decisión dentro de un enjambre. La primera de ellas es llamada posibles alternativas, esta situación es cuando los robots tienen que llegar a un consenso sobre una única opción entre un conjunto de alternativas posibles. La segunda es llamada asignación de tareas, esta situación es cuando los robots tienen que distribuirse entre un conjunto de tareas posibles y operar de forma paralela esas tareas para maximizar el rendimiento del sistema.

### 6.1.2. Aplicaciones de la robótica de enjambre

- **Recuperación de desastres:** el uso de la robótica de enjambres en el área de búsqueda y rescate ante desastres naturales ha demostrado la habilidad que tienen los enjambres utilizando *Unmanned Aircraft Systems* (UAS) para localizar más del 90 porciento de sobrevivientes dentro de un escenario inundado urbano de 2×2 kilómetros en menos de una hora. De la misma manera, los enjambres pueden ser utilizados para mapear y localizar de forma rápida los restos de un desastre natural, explorar y mapear edificios dañados de forma eficiente [17].
- **Defensa:** pruebas recientes han demostrado la habilidad que tienen los enjambres de drones pequeños, por medio de visión por computador y algoritmos de aprendizaje automático (*machine learning*), poder detectar y discriminar objetivos. Así mismo, la robótica de enjambre aplicada a la defensa se puede utilizar para búsqueda y rescate

**Figura 10:** *Aplicación de un sistema Swarm utilizando UAS [17].*

de combate, adquisición de ubicación de objetos inteligentes, enjambre de municiones, entre otros.

- **Reconocimiento, inspección y mapeo:** los UAS son utilizados individualmente para tener una cobertura más detallada de un área grande en poco tiempo. Los enjambres pueden mejorar la exactitud de mapas al cubrir el área múltiples veces. Los pequeños UAS podrían navegar entre edificios, mapeo de paredes interiores y exteriores, entre otros obstáculos.

## 6.2. *Particle Swarm Optimization* (PSO)

El PSO es un algoritmo propuesto por Kennedy y Eberhart, el cuál simula el comportamiento social de animales, tales como insectos, pájaros y peces. Estos enjambres conforman una forma cooperativa de encontrar alimento, en donde cada uno de los miembros del enjambre sigue cambiando el patrón de búsqueda de acuerdo con las experiencias de aprendizaje propias y la de otros miembros [20].

En el PSO, las partículas navegan a través de un espacio de posibles soluciones para encontrar la mejor solución. Cada partícula va ajustando su posición en función de la mejor encontrada de forma individual (*local best*) y la mejor solución encontrada por todo el enjambre (*global best*) [20].

### 6.2.1. Estructura del algoritmo PSO

Se inicializa el número de partículas $n_s$ y la población del enjambre definida. Cada partícula tiene posicion $(x, y)$.

$$P_i = [x_1, x_2, ..., x_{n_s}]$$

Cada partícula es evaluada con la función de costo, también llamada *fitness function*. Dicha función es utilizada para encontrar la solución óptima al problema.

$$f(P) = f(x_1, x_2, ..., x_{n_s})$$

Posteriormente, cada partícula guarda la posición con menor costo que ha encontrado hasta el momento, conocida como *local best* (mejor posición encontrada por cada partícula). Así mismo, cada partícula transmite su mejor posición y se selecciona cual de todas las posiciones es la mejor dentro de todo el enjambre, conocida como *global best*.

Después se procede a calcular los dos primeros factores del PSO:

Componente cognitivo: diferencia entre la posición actual y el *local best*. Me da cierta información individual de la partícula.

$$P_{local} - P_i$$

Componente social: diferencia entre la posición actual y el *global best*. Me da cierta información intercambiada por las partículas.

$$P_{global} - P_i$$

Utilizando ambos factores se obtiene la ecuación para la siguiente iteración de la velocidad de las partículas:

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

Otros parámetros que son implementados para el algoritmo PSO se encuentran los coeficientes de aceleración $C_1$ y $C_2$, estos representan el peso de cuanta importancia se da a la experiencia de forma individual ($C_1$) o el aprendizaje de global del enjambre ($C_2$). Así mismo, se utilizan los factores de uniformidad $r_1$ y $r_2$ para introducir un elemento estocástico al algoritmo, dichos factores se encuentran en el rango de 0 a 1.

Por otra parte, el factor de constricción $\varphi$ restringe el control de la habilidad de búsqueda global de la partícula [21]. Es una manera de asegurar la convergencia a un punto sin necesidad de colocar rango límite a la velocidad. Este parámetro se calcula mediante la ecuación:

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

El factor de inercia $w$ controla las habilidades de exploración (con valores altos) y explotación (con valores pequeños) del enjambre [21]. Este factor hace que el movimiento las partículas sean en la misma dirección y velocidad. Según Eberhart y Shi [22] y [23], existen varias ecuaciones para calcular este parámetro:

Inercia constante:

$$0.8 < w < 1.2$$

Inercia Lineal Decreciente:

$$w \;=\; w_{max} - (w_{max} - \; w_{min})\frac{iter}{MAX_{iter}}$$

Inercia Caótica:

$$cZ_i \;\in\; [0, ...1]$$
$$Z_{i+1} = 4Z_i(1 - Z_i)$$
$$w \;=\; w_{max} - (w_{max} - \; w_{min})\frac{MAX_{iter} - iter}{MAX_{iter}} w_{mn}\; Z_{i+1}$$

Inercia Aleatoria:

$$rand() \;\in\; [0, ...1]$$
$$w \;=\; 0.5 \;+\; \frac{rand()}{2}$$

Inercia Exponencial:

$$w \;=\; w_{min} \;+ (w_{max} - w_{min})\; e^{\frac{M\_A-t}{10}}$$

La ecuación del PSO utilizando los parámetros mencionados anteriormente se muestra a continuación:

$$V_{i+1} = \varphi[wV_i + C_1 * r_1(P_{local} - P_i) + C_2 * r_2(P_{global} - P_i)] \tag{2}$$

donde $V_i$ es la velocidad actual de la partícula, $V_{i+1}$ es la nueva velocidad obtenida para cada partícula. Finalmente, después de obtener la nueva velocidad se procede a obtener la

nueva posición mediante:

$$X_{i+1} = X_i + V_{i+1}\Delta t \tag{3}$$

donde $X_i$ es la posición actual de la partícula, $X_{i+1}$ es la nueva posición obtenida para cada partícula. El término $\Delta t$ es el tiempo que toma el algoritmo en realizar cada iteración.

## 6.3. Funciones de costo

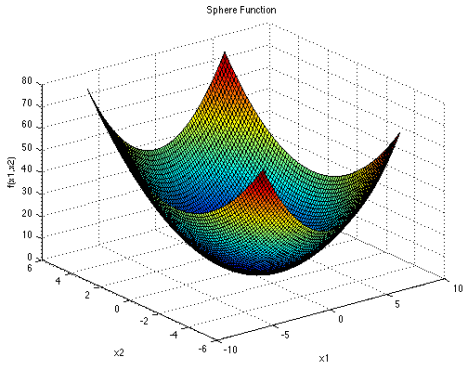
El algoritmo PSO determina la mejor posición colectiva encontrada por el enjambre. Para evaluar la efectividad de esta posición, se utilizan funciones de costo como *benchmark functions*. Estas funciones pueden presentar un único mínimo o varios mínimos y máximos. A continuación, se definen cada una de ellas:

### 6.3.1. Función sphere

La función Sphere Ecuación 4 tiene un único mínimo en el origen, como se ilustra en la Figura 11. Esta función de costo permite evaluar la velocidad de convergencia del enjambre.

$$f(x) \ = \sum_{i \ = \ 1}^{d} x_i^2 \tag{4}$$

**Figura 11:** *Función de costo Sphere.*

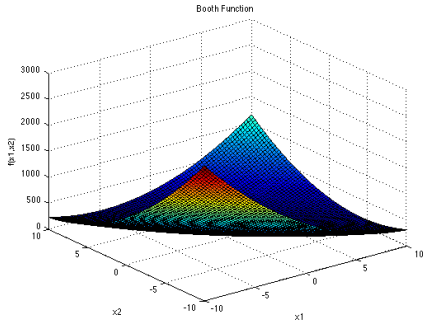


### 6.3.2. Función booth

La función Booth Ecuación 5 posee dos mínimos globales: en $(0, 0)$ y en $(1, 3)$. Esta se muestra en la Figura 12.

$$f(x) \;=\; (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \tag{5}$$

**Figura 12:** *Función de costo Booth.*

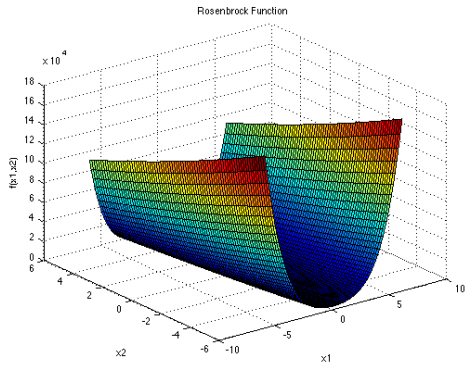



### 6.3.3. Función rosenbrook

La función Rosenbrook es también conocida como la función valle Ecuación 6, el mínimo global se encuentra en $(1, 1)$. Esta se muestra en la Figura 13.

$$f(x) \;=\; \sum_{i\,=\,1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right] \tag{6}$$

**Figura 13:** *Función de costo Rosenbrook.*

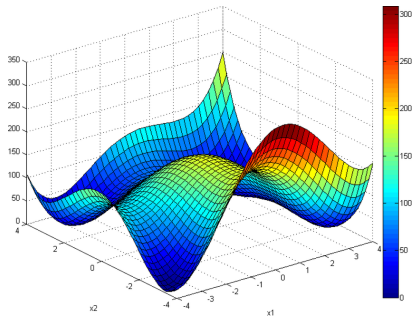


### 6.3.4. Función himmelblau

La función Himmelblau Ecuación 7 posee múltiples mínimos globales como: $(3, 2)$ y $(-2.8051, 3.2831)$. La función se muestra en la Figura 14.

$$f(x) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2 \tag{7}$$

**Figura 14:** *Función de costo Himmelblau.*

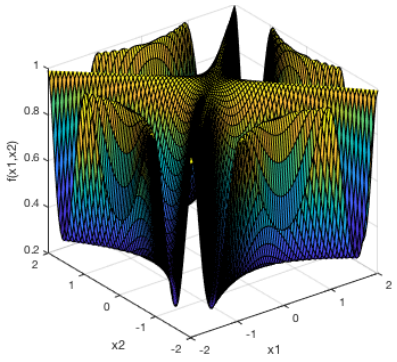



### 6.3.5. Función schaffer no. 4

La función Schaffer no. 4 Ecuación 8 posse múltiples mínimos alejados del origen. Esta se muestra en la Figura 15.

$$f(x) = 0.5 + \frac{cos^2\left(sin\left(\left|x_1^2 - x_2^2\right|\right)\right) - 0.5}{\left[1 + 0.001\left(x_1^2 + x_2^2\right)\right]^2} \tag{8}$$

**Figura 15:** *Función de costo Schaffer no. 4.*

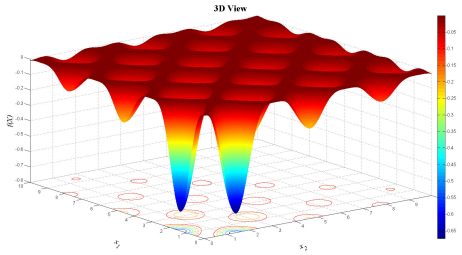


### 6.3.6. Función keane

La función Keane Ecuación 9 posee múltiples mínimos globales alineados al eje $x$ y al eje $y$. Esta se muestra en la Figura 16.

$$f(x) = -\frac{sin^2(x_1 - x_2)\ sin^2(x_1 + x_2)}{\sqrt{x_1^2 + x_2^2}} \tag{9}$$

**Figura 16:** *Función de costo Keane.*



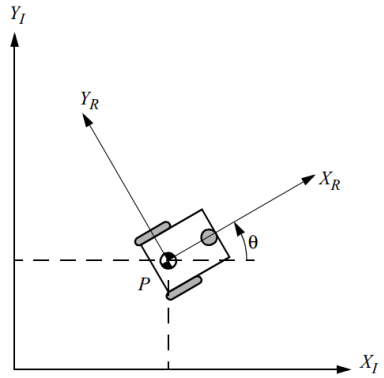

## 6.4. Robótica móvil

Los robots móviles son un diseño vehicular conformado por sistemas eléctricos y mecánicos. El diseño de este robot consiste en una plataforma o chasis sobre ruedas u orugas. Algunas aplicaciones básicas de estos robots son en manufactura, moviendo materias primas, partes o herramientas [24].

### 6.4.1. Modelo cinemático de los robots móviles

Cada rueda del robot contribuye al movimiento del robot y al mismo tiempo, impone ciertas restricciones al robot. Las ruedas van unidas a la base o chasis del robot, las fuerzas de cada rueda se deben expresar con respecto de un marco de referencia [25].

**Figura 17:** *Marco de referencia global y marco del robot [25].*




Para el modelo cinemático se considera al robot como un cuerpo rígido sobre ruedas, moviéndose sobre un plano horizontal. Para especificar la posición del robot se debe tomar en consideración el marco de referencia global inercial y el marco de referencia del robot. La pose del robot están dadas por la posición $x, y$ y el ángulo entre el marco inercial y el marco del robot Figura 17.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{10}$$

Para describir el movimiento del robot en términos de las componentes de movimiento, es necesario mapear las coordenadas del marco inercial a las coordenadas del marco del robot. Para hace este mapeo se utiliza la matriz ortogonal de rotación,

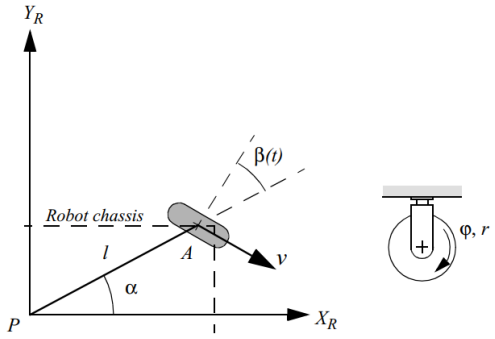
$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La transformación entre el marco inercial y el marco del robot está dada por,

$$\dot{\xi_I} = R(\theta)^{-1}\dot{\xi_R} \tag{11}$$

Para una rueda estándar direccional los parámetros para realizar el modelo cinématico son las coordenadas de punto de instalación de las ruedas en el robot, es decir, a que distancia $l$ y ángulo $\alpha$ se instalan las ruedas en el robot. Asimismo, se toma en consideración el ángulo de instalación de la rueda $\beta$, la velocidad $\varphi$ y el radio $r$ de las ruedas Figura 18.

**Figura 18:** *Parámetros de una rueda estándar [25].*




El movimiento del robot móvil está definido por restricciones de rodamiento y no deslizamiento. Para la restricción de rodamiento se considera la siguiente ecuación para una rueda estándar direccional,

$$[sin(\alpha+\beta) - cos(\alpha+\beta)\ (-l)\ cos(\beta)]R(\theta)\dot{\xi}_I - J_2 r\dot{\varphi} = 0] \tag{12}$$

Luego sustituyendo el primer término de la Ecuación 12 con un jacobiano $J_{1i}$ se obtiene la expresión,

$$J_1 R(\theta)\dot{\xi}_I \ = \ -J_2 r\dot{\varphi} \tag{13}$$

Asimismo, se presenta la ecuación de restricción de no deslizamiento,

$$[cos(\alpha+\beta)\ sin(\alpha+\beta)\ l\ sin(\beta)]\ R(\theta)\ \dot{\xi}_I = 0] \tag{14}$$

De igual manera, al sustituir el primer término de la Ecuación 14 por un jacobiano $C_{1i}$ se obtiene la expresión,

$$C_1 R(\theta)\ \dot{\xi}_I = 0 \tag{15}$$

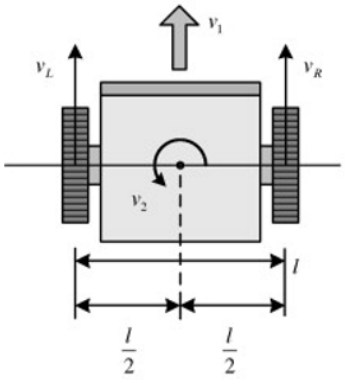
Finalmente, el modelo final a utilizar para el robot móvil con $(N)$ ruedas es, en donde $A^+$ es la pseudo-inversa de los jacobianos $J_1$ y $C_1$, $B$ es la matriz formada por matrices de ajuste $J_2$ y $C_2$, $v$ es la velocidad de las ruedas y $\dot{\xi}_I$ es la velocidad del robot desde el marco inercial.

$$\dot{\xi}_I = R(\theta)\ A^+ Bv \tag{16}$$

### 6.4.2. Robot diferencial

El robot diferencial está conformado por dos ruedas, con diámetro $r$. Ambas ruedas están a una distancia $l$ del punto $p$ [25]. Con los parámetros anteriores y la velocidad de giro por cada rueda, $\dot{\varphi_R}$ y $\dot{\varphi_L}$ Figura 19.

**Figura 19:** *Robot diferencial [26].*




La velocidad angular de las ruedas se obtienen a partir de

$$w_R = \frac{r\dot{\varphi}_R}{2l}$$

$$w_L = \frac{-r\dot{\varphi}_L}{2l}$$

Combinando la velocidad de las ruedas y la Ecuación 16, se obtiene el modelo cinemático de un robot diferencial:

$$\dot{\xi}_I = \frac{r}{2} \begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \dot{\varphi_R} \\ \dot{\varphi_L} \end{bmatrix} \tag{17}$$

### 6.4.3. Modelo uniciclo

El modelo uniciclo es un vehículo capaz de moverse hacia delante y atras $(x, y)$ y cambiar su dirección $(\theta)$. El modelo cinemático del uniciclo se define como lo siguiente:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta \\ sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w \tag{18}$$

El modelo uniciclo representa el funcionamiento del vehículo tomando en consideración una velocidad lineal ($v_{ref}$) y angular ($w_{ref}$), luego, a partir de la Ecuación 17 obtener las velocidades de las ruedas [27].

## 6.5. Controladores de velocidad y posición de robots diferenciales

Para que el uniciclo pueda converger a la meta es necesario aplicar control para asegurar su convergencia. Los dos tipos de controladores más utilizados son el PID de posición y orientación, así como el controlador PID con acercamiento exponencial.

### 6.5.1. Controlador PID de posición y orientación

Este controlador está formado por un PID para la posición y orientación de la meta deseada. Primeramente, se obtiene el error de posición mediante la resta entre la posición deseada y la posición del robot [28].

$$e_p = \left\| \begin{matrix} x_g - x \\ y_g - y \end{matrix} \right\|_2$$

En el caso del error de orientación, se debe de utilizar una ecuación específica la cual se muestra a continuación:

$$e_o = \arctan\left(\frac{sin(\theta_g - \theta)}{cos(\theta_g - \theta)}\right)$$

$$\theta_g = \arctan\left(\frac{y_g - y}{x_g - x}\right)$$

La implementación del control PID de posición y orientación en el dominio tiempo con $k_p$, $k_i$, $k_d$ como constantes.

$$v \;=\; k_{P_p} e_p + k_{I_p} \int_0^t e_p(\tau)\; d\tau \;+\; k_{D_p} \dot{e_p} \tag{19}$$

$$w \;=\; k_{Po}e_o + k_{I_o}\int_0^t e_o(\tau)\;d\tau \;+\; k_{D_o}\dot{e_o} \tag{20}$$

Por lo tanto, el controlador se resume como:

$$v \;=\; PID(e_p) \tag{21}$$

$$w \;=\; PID(e_o) \tag{22}$$

### 6.5.2. Controlador PID con acercamiento exponencial

Para corregir los problemas de convergencia en espirales que presenta el controlador PID convencional, se agrega un coeficiente de ajuste $\alpha$ que hace que la velocidad del robot disminuya cuando esté cerca de la meta [28]. El controlador de orientación se mantiene igual que en la Ecuación (20). Dicho controlador se presenta a continuación:

$$v \;=\; -k_p e_p = -v_0\left(1 - e^{-\alpha e_p^2}\right) \tag{23}$$

En donde $v_0$ es la velocidad lineal nominal o máxima.

## 6.6. Campos potenciales artificiales

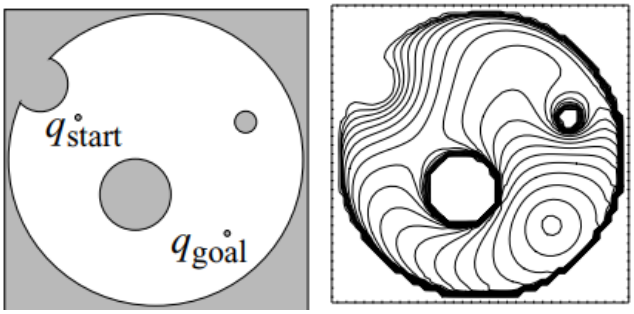
Los campos potenciales artificiales son una técnica utilizada en robótica para la planificación de trayectorias. Esta técnica genera un campo de fuerzas o gradientes en el mapa, que atraén al robot hacia la meta evitando los obstaculos [29]. En este enfoque el objetivo se visualiza como un mínimo en el espacio y los obstáculos se visualizan como montañas como se ilustra en la Figura 20.

### 6.6.1. Campo atractivo

El campo atractivo está conformado por una función potencial cónica y una función potencial cuadrática. En la Ecuación 24 se define el campo atractivo, en donde $k_{rep}$ es la ganancia de la atracción, $d$ es la distancia del robot hacia la meta y $d^*_{goal}$ es el umbral de distancia hacia la meta donde cambia entre el potencial cónico y cuadrático [29].

$$U_{att} = \begin{cases} \frac{1}{2}\;k_{att}\;d^2(q, q_{goal})\;, & d(q, q_{goal}) \leq d^*_{goal} \\ d^*_{goal}\;k_{att}d(q, q_{goal}) \;-\; \frac{1}{2}k_{att}\left(d^*_{goal}\right)^2, & d(q, q_{goal}) \geq d^*_{goal} \end{cases} \tag{24}$$

**Figura 20:** *Campo potencial artificial [29].*




**(a)** *Campo potencial definición de punto de inicio y meta.*

**(b)** *Generación campo potencial.*

### 6.6.2. Campo repulsivo

El campo repulsivo genera una fuerza que debe ser lo suficientemente fuerte cuando el robot está cerca del obstáculo, pero no debe influir en su movimiento cuando el robot se encuentre lejos de él [30]. En la Ecuación 25 se define el campo repulsivo, en donde $k_{rep}$ es la ganancia de repulsión, $rho(q)$ es la distancia del robot al obstaculo y $rho_0$ es la distancia de fluencia del obstáculo.

$$U_{att} = \begin{cases} \frac{1}{2}\, k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 , & \rho(q) \leq \rho_0 \\ 0 & \rho(q) \geq \rho_0 \end{cases} \tag{25}$$

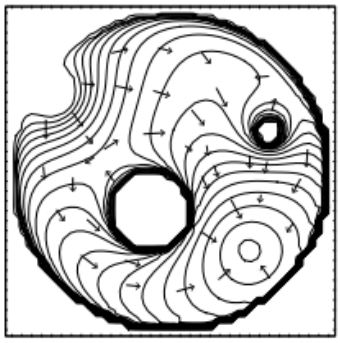
La suma del campo repulsivo y atractivo generan un campo total artificial potencial:

$$U_{total} = U_{att} + U_{rep} \tag{26}$$

### 6.6.3. Descenso del gradiente

La idea del *gradient descent* es tomar pequeños pasos en la dirección opuesta a el mismo. El gradiente del campo potencial total dirige al robot para llegar a la meta, evitando colisionar con los obstáculos, como se ilustra en la Figura 21.

**Figura 21:** *Gradiente del campo total potencial [26].*



## 6.7. Protocolos de red

El protocolo de red es un estándar de comunicación, este contiene reglas e información sobre como las computadoras intercambian datos entre sí. Los protocolos llevan a cabo las siguientes tareas:

- Establecer una comunicación confiable entre los equipos implicados en la comunicación.
- Dirigir los paquetes de datos enviados al destinatario correcto.
- Si los paquetes no llegan al destinatario, el protocolo asegura que se reenvíen.
- Transmisión sin errores de los paquetes de información.

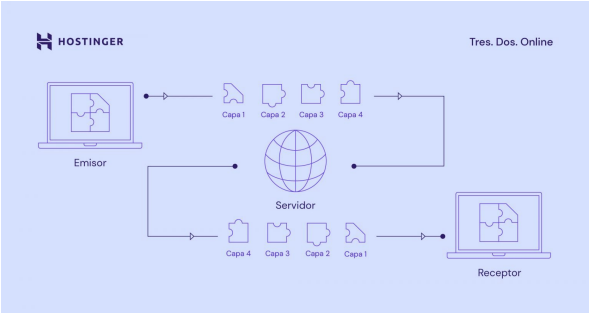
### 6.7.1. Protocolo TCP

El protocolo de control de transmmisión (TCP) es utilizado para transmitir datos a través de redes. Este protocolo descompone los datos en paquetes y los reenvía a la capa del protocolo de internet (IP) para garantizar que cada mensaje llegue a su ordenador de destino. Los paquetes pueden viajar por varias rutas si la ruta actual está congestionada o no está disponible.

El protocolo IP divide las tareas de comunicación en varias capas. Todos los paquetes pasan por al menos cuatro capas antes de llegar a su destino como se muestra en la Figura 22. Este protocolo está compuesto por las siguientes capas:

- **Capa de acceso a la red:** también conocida como la capa de enlace de datos, gestiona la infraestructura física que permite a los ordenadores comunicarse entre sí por internet. Esto abarca, entre otros elementos, cables de Ethernet, redes inalámbricas, tarjetas de interfaz de red, etc.

**Figura 22:** *Estructura del protocolo TCP.*




- **Capa de internet:** también llamada la capa de red, controla el flujo y el enrutamiento de tráfico para garantizar que los datos se envían de forma rápida y correcta. Esta capa también es responsable de volver a armar el paquete de datos en el destino.
- **Capa de transporte:** proporciona una conexión de datos fiable entre dos dispositivos de comunicación. Esta capa divide los datos en paquetes, confirma los paquetes que ha recibido del remitente y se asegura que el destinatario confirme los paquetes recibidos.
- **Capa de aplicaciones:** es el grupo de aplicaciones que permite al usuario acceder a la red. Por ejemplo, el correo electrónico, aplicaciones de mensajeria, etc.

## 6.8. Matlab

Matlab es una plataforma de programación y cálculo numérico, una característica importante de este lenguaje es que implementa el uso de matrices y vectores para realizar cálculos de algebra lineal. También se puede llevar a cabo análisis de datos, gráficas, creación de apps y cálculo paralelo.

Algunas aplicaciones que utilizan Matlab son sistemas de control, aprendizaje reforzado, procesamiento de señales, robótica, entre otros [31].

### 6.8.1. Herramientas para mejorar el rendimiento de programas en Matlab

A continuación, se presentan diferentes herramientas para mejorar el rendimiento y tiempo de ejecución de los scripts en matlab [32].

- ***Code Analyzer y The profiler***: el analizador de código (*Code Analyzer*) es utilizado para revisar tu código mientras lo escribes, esta herramienta ayuda a identificar problemas y recomienda ciertas modificaciones para maximizar el rendimiento del script.

Por otro lado, el perfilador (*profiler*) muestra en que parte del código se está perdiendo tiempo, brinda un listado del número de veces que se ejecuta cada función y el tiempo total que tarda cada una. Una vez que se tiene esta información, se puede optar por diferentes formas para mejorar el rendimiento en las diferentes secciones del código.

- **Programación en paralelo (*Parallel computing*):** esta técnica permite ejecutar ciertas tareas de forma paralela, es decir, se ejecutan al mismo tiempo. Matlab utiliza la librería llamada *Parallel Computing toolbox*, esta librería utiliza los procesadores multinúcleo, GPU y clusters de proceso para obtener la cantidad de *workers* (núcleos) se pueden utilizar para la paralelización. Las dos técnicas más utilizadas son preasignación (*preallocation*) y vectorización [33].
- **Preasignación:** la preasignación se basa en inicializar los vectores con el tamaño requerido. Dado que las matrices en matlab se mantienen en bloques de memoria, cambiar el tamaño de las matrices repetidamente requiere que matlab dedique tiempo para buscar los bloques de memoria más grandes y luego mover la matriz a esos bloques.
- **Vectorización:** la vectorización es el proceso de convertir código con bucles a código utilizando operaciones de matrices y vectores. Asimismo, en caso de no poder vectorizar un ciclo for, se puede utilizar la paralelización de matlab por ejemplo *parfor*.
- **Ejecutables *MEX-files*:** este tipo de archivos permite reemplazar partes del código de matlab utilizando otro tipo de lenguaje de programación como lenguaje C/C++.

## 6.9. Python

Python es un lenguaje de programación de código abierto y gratuito. Utiliza programación orientada a objetos en lugar de utilizar funciones, es de alto nivel y tiene aplicaciones en distintos campos: ciencia de datos, análisis de datos, aprendizaje automático, desarrollo web y desarrollo de software.

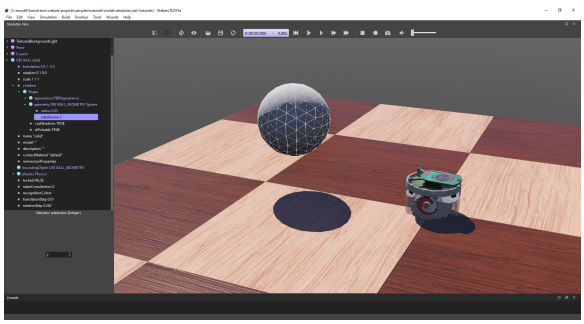
### 6.9.1. Paralelización

Uno de los métodos para implementar paralelización en Python es mediante el uso de hilos (*threads*). Un hilo representa una secuencia de instrucciones que el procesador (CPU) ejecuta dentro de un programa o aplicación. Cuantos más hilos pueda manejar el CPU, más tareas podrán se podrán ejecutar. Esta técnica mejora el rendimiento al permitir que múltiples hilos se ejecuten simultáneamente en uno o varios núcleos (*cores*) del procesador, lo que aumenta la velocidad y eficiencia del sistema [34].

## 6.10. Webots

Webots es un entorno profesional de simulación ampliamente utilizado en robótica, como se ilustra en la Figura 23. Este software permite crear distintos entornos mediante mundos virtuales que incluyen propiedades físicas, tales como masa, uniones, coeficiente de fricción, entre otros. Además, facilita la simulación de distintos robots, como el robot móvil E-puck, equipado con sus propios sensores de distancia, motores, cámara, emisores y receptores, entre otros.

**Figura 23:** *Entorno de simulación Webots [26].*

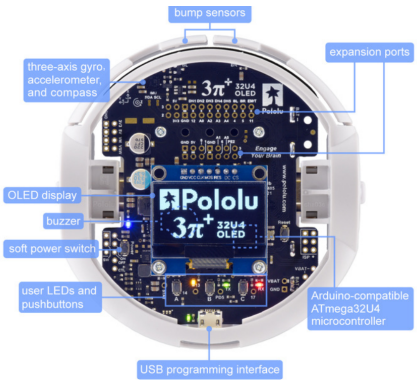


## 6.11. Pololu 3pi+ 32U4

El Pololu 3pi+ es una plataforma robótica móvil versátil, de alto rendimiento y es programable por el usuario. Este robot está equipado por un microcontrolador *ATmega32U4 AVR* de la compañía *microchip*, tiene un puerto USB externo y tiene compatibilidad con Arduino. Asimismo, contiene puertos de expansión para personalizar o mejorar al robot, está equipado con dos drivers puente H y una variedad de sensores, incluyendo un par de codificadores de cuadratura para control de motor de circuito cerrado, un sensor de medida inercial, entre otros.

También posee una pantalla OLED de 128 x 64, un buzzer o zumbador y tres LEDs que se utilizan como indicadores. Los 3pi+ funcionan con cuatro baterías AAA, estas pueden ser alcalinas o recargables de NiMH [35].

**Figura 24:** *Estructura de un Pololu 3pi+ [35].*




## 6.12. OptiTrack

El ecosistema Robotat utiliza seis cámaras $Prime^x41$ de OptiTrack. Tienen una resolución de 4.1 megapíxeles y una velocidad de 180 cuadros o *frames* por segundo. Dichas cámaras se utilizan como un sistema de captura de movimiento óptico altamente, preciso [8].

Este sistema de captura utiliza un seguimiento de marcadores pasivos y activos, estos se colocan en los objetos o sistemas robóticos para obtener la posición y orientación del *marker*. El programa que se utiliza para obtener dichas coordenadas se conoce como *Motive*. Este software permite configurar y calibrar las cámaras utilizadas en el sistema de captura de OptiTrack.

**Figura 25:** *Cámaras Prime x 41 [8].*

# CAPÍTULO 7

## Algoritmo *particle swarm optimization*

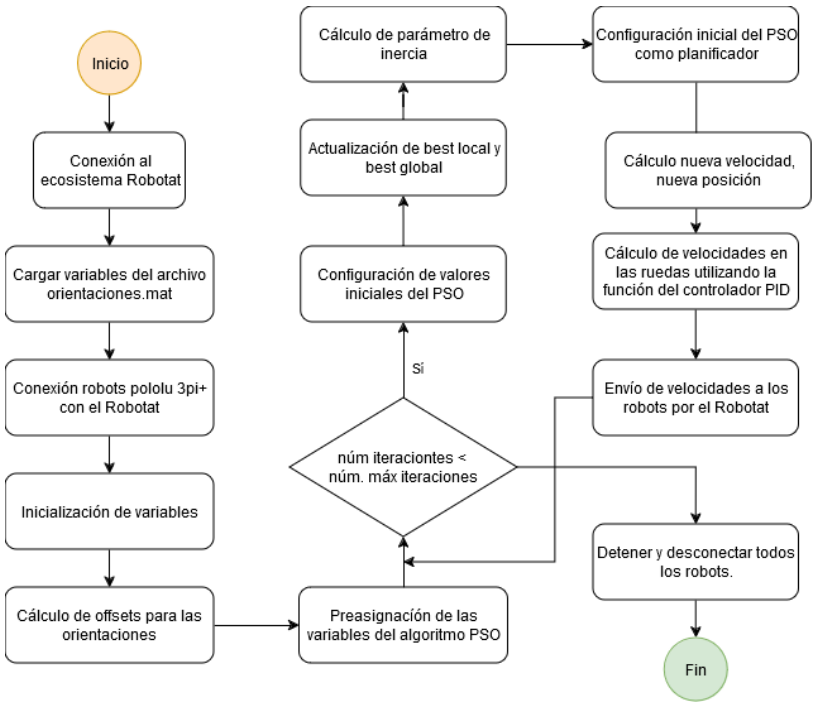
El algoritmo de robótica de enjambre *Particle Swarm Optimization* (PSO) simula el comportamiento colectivo de animales, permitiendo los individuos se trasladen en conjunto hacia una meta en común. En este algoritmo, un conjunto de partículas, que presentan posibles soluciones al problema de optimización, navegan a través del espacio de soluciones en busca de la mejor solución posible.

A continuación, se describe la estructura del algoritmo *Modified Particle Swarm Optimization* (MPSO) desarrollado en la fase anterior [14]. Y se presentan los resultados de las pruebas iniciales con el algoritmo en el entorno de simulación Webots.

## 7.1. Estructura del algoritmo MPSO

El algoritmo MPSO, desarrollado en la fase anterior, se compone de tres partes principales. La primera consiste en establecer la conexión con el ecosistema robotat y los agentes robóticos Pololu 3pi+. La segunda parte se centra en el algoritmo PSO, que se encarga de calcular las posiciones de los agentes hasta alcanzar la meta deseada. Por último, se incluye el controlador PID con acercamiento exponencial, que se encarga de calcular las velocidades de las ruedas para cada uno de los agentes. La Figura 26 muestra el esquema general de la estructura del algoritmo MPSO. A continuación, se describe detalladamente cada una de las partes del algoritmo.

**Figura 26:** *Estructura general del algoritmo MPSO.*




Nota. Elaboración propia.

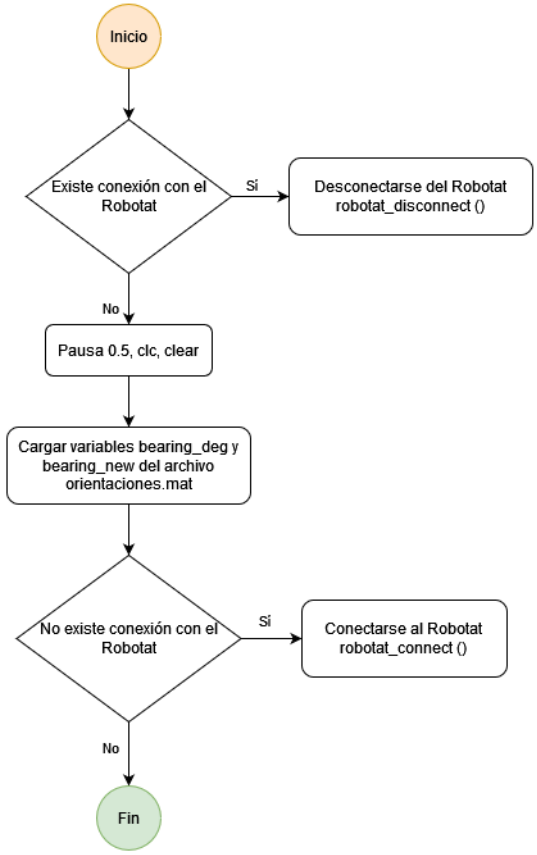
### 7.1.1. Comunicación con ecosistema Robotat

La comunicación con el ecosistema Robotat es un elemento importante dentro del algoritmo MPSO. El servidor permite obtener las posiciones y orientaciones de los agentes robóticos por medio de las funciones:

- ***robotat_connect.m***: establece la conexión con el servidor. Esta función crea un objetivo TCP, con una dirección IP y un número de puerto asignado para obtener la información necesaria.
- ***robotat_disconnect.m***: realiza la desconexión con el servidor.
- ***robotat_get_pose.m***: obtiene las posiciones y orientaciones de los marcadores de interés. Esta función recibe como argumentos el objeto TCP creado, número de marcadores a utilizar y el tipo de representación para la orientación. La función regresa la información en una matriz ordenada con las posiciones $(x,y,z)$ y orientaciones dependiendo si es secuencia de ángulos de Euler $(xyz,\ zyx)$ o cuaterniones $(\omega,x,y,z)$.

La conexión con el servidor en el algoritmo MPSO se hace por medio de las funciones *robotat_ connect.m* y *robotat_ disconnect.m*. Primero, verifica si ya existe una conexión con el Robotat, después verifica si todavía no existe la conexión y se conecta al servidor. En la Figura 27 se presenta el proceso para conectarse con el servidor Robotat.

**Figura 27:** *Diagrama de flujo para la conexión con el servidor.*




Nota. Elaboración propia.

### 7.1.2. Control pololu 3pi+

Los agentes robóticos utilizados para las distintas pruebas fueron los Pololu 3pi+. Estos están equipados con un ESP32, el cual se encarga de conectarse a la red del Robotat para recibir las velocidades de las ruedas y enviarlas a los encoder de los robots mediante el protocolo TCP. Las funciones para controlar a los robots se describen a continuación:

- ***robotat_3pi_connect.m***: esta función establece la conexión con el agente robótico deseado, creando un ID asociado a una dirección IP (del 1 al 9). La salida de la función es un objeto TCP que contiene la IP (número del robot) y el puerto a utilizar. La función recibe como argumento el número de robot que se desea conectar.
- ***robotat_3pi_disconnect.m***: desconecta el agente robótico deseado de la red. Esta función recibe como argumento el objeto TCP del robot.
- ***robotat_3pi_force_stop.m***: envía ceros a las velocidades de las ruedas para detener al robot. Esta función recibe como argumento el objeto TCP del robot (ID del robot).
- ***robotat_3pi_set_wheel_velocities.m***: actualiza las velocidades en rpm de las ruedas. Limita las velocidades entre 850 y -850 rpm. Esta función recibe como argumento el objeto TCP del robot (ID del robot) así como las velocidades izquierda y derecha de las ruedas en rpm.

La conexión con los Pololu 3pi+ en el MPSO se realiza mediante la función *robotat_3pi_connect.m*. Esta función define los números del primer al último agente a utilizar y calcula la cantidad total de agentes robóticos. Luego, mediante un ciclo for y la función eval, conecta a todos los agentes y almacena en diferentes variables de tipo struct el ID del agente, IP y puerto del servidor. En el Cuadro 1, se presentan los pasos para la conexión con los agentes robóticos.

**Cuadro 1:** *Pasos para la conexión con los agentes Pololu 3pi+.*

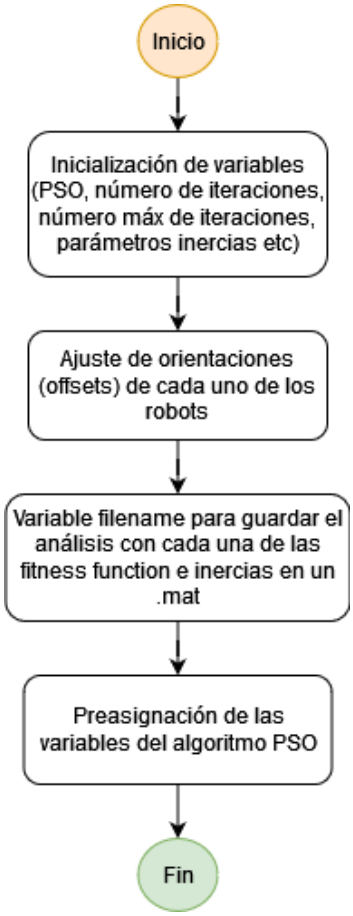
| Paso | Descripción |
|---|---|
| 1 | Define el número del primer agente robótico |
| 2 | Define el número del último agente robótico |
| 3 | Calcula la cantidad total de agentes robóticos |
| 4 | Para cada robot desde el primer agente hasta el último agente: |
| 5 | Realiza una pausa de 0.1 segundos |
| 6 | Conecta al robot con la función **robotat_3pi_connect()** |
| 7 | Fin del bucle |
| 8 | Retorna una estructura **struct** para el primer agente hasta el último agente |

Nota. Elaboración propia.

### 7.1.3. Funcionamiento del algoritmo MPSO

El algoritmo MPSO se compone de partes principales: inicialización de variables y cálculo de *offsets* para los marcadores; configuración inicial del *local best* y *global best*; y la actualización de estos valores junto con el cálculo de las nuevas velocidades y posiciones de las

**Figura 28:** *Diagrama de flujo de la primera parte del algoritmo MPSO.*




Nota. Elaboración propia.

partículas. En la Figura 28 se muestra la estructura del algoritmo para la inicialización de las variables, número máximo de iteraciones, parámetros del factor de inercia, peso cognitivo, social y factor de constricción.

El ciclo principal del algoritmo MPSO está divido por tres partes: la configuración inicial de los valores *local best* y *global best*; la actualización de estos valores con las mejores posiciones encontradas hasta el momento; y el cálculo de las posiciones y velocidades del algoritmo.

Para la configuración inicial Cuadro 2, la variable *state* debe establecerse en 0, lo que permite inicializar las posiciones actuales de las partículas utilizando las posiciones de los robots con la función robotat_get_pose (). Después,se inicializa el valor del *local best* con las posiciones actuales de las partículas y se obtiene el valor mínimo de las posiciones para inicializar el *global best*.

Después de inicializar los valores del *local best* y *global best*, se inicializan los valores de fitness local y fitness global por medio de la función *fitness*. La función *fitness* recibe como argumentos la posición actual del robot $(x, y)$ y el tipo de *benchmark* a utilizar. Por último, se actualiza el *global fitness* con el valor mínimo encontrado anteriormente y se establece la variable *state* en 1 para que se ejecute la configuración una sola vez.

**Cuadro 2:** *Configuración inicial del global best y local best.*

| **Paso** | **Descripción** |
|---|---|
| 1 | Mientras el número de iteraciones sea menor al máximo permitido: |
| 2 | Incrementa el número de iteración |
| 3 | Si la variable **state** es igual a cero: |
| 4 | Obtiene las posiciones actuales de los Pololu |
| 5 | Actualiza la posición actual con las coordenadas $x$ y $y$ |
| 6 | Inicializa el *local best* con la posición actual |
| 7 | Obtiene la norma del vector posición actual |
| 8 | Obtiene el valor mínimo de la norma y guarda la posición (fila, columna) |
| 9 | Actualiza el *global best* con ese valor mínimo |
| 10 | Para $i$ desde 1 hasta la cantidad total de agentes: |
| 11 | Calcula el *local fitness* y el *global fitness* con la función **fitness** |
| 12 | Fin del bucle |
| 13 | Actualiza el *global fitness* con el valor mínimo |
| 14 | Cambia el valor de **state** a 1 |
| 15 | Fin del condicional |
| 16 | Fin del bucle **while** |

Nota. Elaboración propia.

Para la actualización del *local best*, se calcula el fitness actual para cada agente robótico. Si el *fitness* actual es menor al *local fitness*, se actualiza el best local con la posición actual del robot y se actualiza el *local fitness* con el valor del *actual fitness*

Para la actulización del *global best*, se verifica si algún valor del *local fitness* anterior es menor al *global fitness*. Si se cumple esta condición, se actualiza el *global best* con el valor mínimo encontrado del *local best* y se actualiza el *global fitness* con el valor del *local fitness*.

La última parte, implica el cálculo de la nueva posición de cada agente robótico Cuadro 3. Para ello, se actualiza la velocidad anterior con la nueva velocidad y se calcula la nueva velocidad utilizando la Ecuación 2. Finalmente, se determinan las nuevas posiciones de los agentes empleando la Ecuación 3.

**Cuadro 3:** *Cálculo de la nueva posición para el algoritmo MPSO.*

| Paso | Descripción |
|---|---|
| 1 | Mientras el número de iteraciones sea menor al número máximo de iteraciones, hacer: |
| 2 | Calcular valores aleatorios de $\rho_1$ y $\rho_2$ |
| 3 | Calcular el parámetro de inercia |
| 4 | Actualizar velocidad anterior con la nueva velocidad |
| 5 | Calcular la nueva velocidad $(x, y)$ |
| 6 | Calcular la nueva posición $(x, y)$ |

Nota. Elaboración propia.

### 7.1.4. Control PID con acercamiento exponencial

Una vez obtenidas las nuevas posiciones con el algoritmo PSO, el siguiente paso es controlar a los agentes robóticos utilizando un controlador PID con acercamiento exponencial Cuadro 4. El MPSO emplea la función PID_controller1.m, que toma como argumentos la posición del agente robótico, el *offset* de la orientación y la nueva posición calcula por el PSO. Esta función transforma las velocidades lineales y angulares en las velocidades de las ruedas izquierda y derecha.

**Cuadro 4:** *Controlador PID.*

| Paso | Descripción |
|---|---|
| 1 | Mientras el número de iteraciones sea menor al número máximo de iteraciones, hacer: |
| 2 | Obtener las posiciones actuales de los agentes |
| 3 | Para cada agente, desde el primero hasta el último, hacer: |
| 4 | Calcular las velocidades de las ruedas |
| 5 | Guardar la velocidad izquierda y derecha en cada iteración |
| 6 | Guardar las trayectorias de los agentes en cada iteración |
| 7 | Guardar la velocidad lineal y angular en cada iteración |
| 8 | Fin del bucle |
| 9 | Fin del bucle **while** |

Nota. Elaboración propia.

El controlador está compuesto por el PID de posición, utiliza únicamente la constante proporcional con los parámetros v0 y $\alpha$ y el PID de orientación. La forma de implementarlo es por medio del cálculo del error de posición para obtener la velocidad lineal $v$ y el error de orientación para obtener la velocidad angular $\omega$.

El controlador se compone de un PID de posición que utiliza únicamente la constante proporcional, junto con los parámetors $v_0$ y $\alpha$, así como un PID de orientación convencional. El controlador obtiene las velocidades lineal y angular ($v$, $\omega$) para calcular las velocidades de las ruedas con la Ecuación 27 y la Ecuación 28. En donde *distance from center* es la distancia medida del centro de las ruedas y *wheel radius* es el radio de las ruedas.

$$phi\_L = \frac{(v - w \times distance\_from\_center)}{wheel\_radius} \tag{27}$$

$$phi\_R = \frac{(v + w \times distance\_from\_center)}{wheel\_radius} \tag{28}$$

## 7.2. Pruebas iniciales en simulador webots

Antes de iniciar las pruebas físicas con el algoritmo MPSO en el ecosistema Robotat, se desarrolló una simulación del MPSO en el entoron de simulación Webots, ya que no se contaba con una en la fase anterior. Esta simulación se realizó con el propósito de comprender mejor el funcionamiento del PSO. A continuación, se presenta la lógica empleada para replicar el algoritmo MPSO en Webots, así como las pruebas realizadas en la fase anterior.

### 7.2.1. Simulación del algoritmo MPSO

El entorno de simulación Webots permite crear mundos con diversos tipos de robots móviles y controladores para gestionar el comportamiento de los robots en varios lenguajes de programación (Matlab, Python, C++).

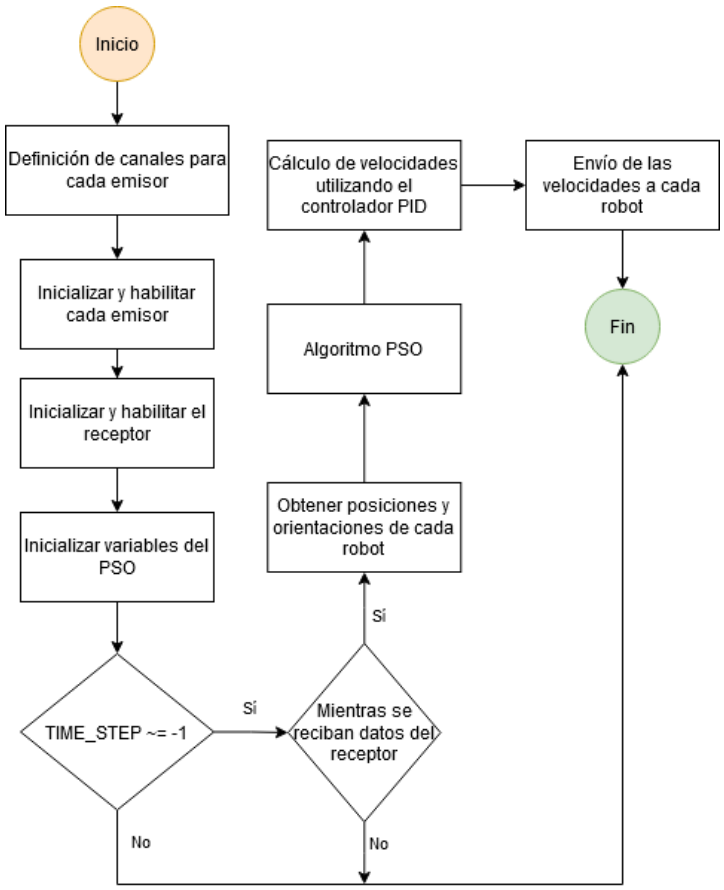
El mundo implementado para la simulación fue desarrollado por el catedrático Miguel Zea, el cual contenía un código ejemplo con la implementación de los *handles* de los motores, GPS y brújula para obtener las coordenadas de los agentes. El mundo ya incluía un modelo 3D del agente Pololu 3pi+ y la mesa con las dimensiones del Robotat, lo cual hizo que la simulación fuera lo más cercano a las pruebas físicas.

#### Modificaciones del mundo de Webots

El objetivo de la simulación era replicar las pruebas lo más exacto posible a las realizadas en el ecosistema Robotat. Para ello, se implementó un esquema similar al ecosistema Robotat, en el que se estableció un robot supervisor que simula la computadora central (el servidor).

Este robot supervisor se encarga de recibir las posiciones de los robots, ejecutar el algoritmo MPSO para calcular las nuevas posiciones y convertirlas en velocidades de las ruedas mediante un controlador PID. Finalmente, envía las velocidades de las ruedas a cada agente robótico, como se ilustra en la Figura 29.

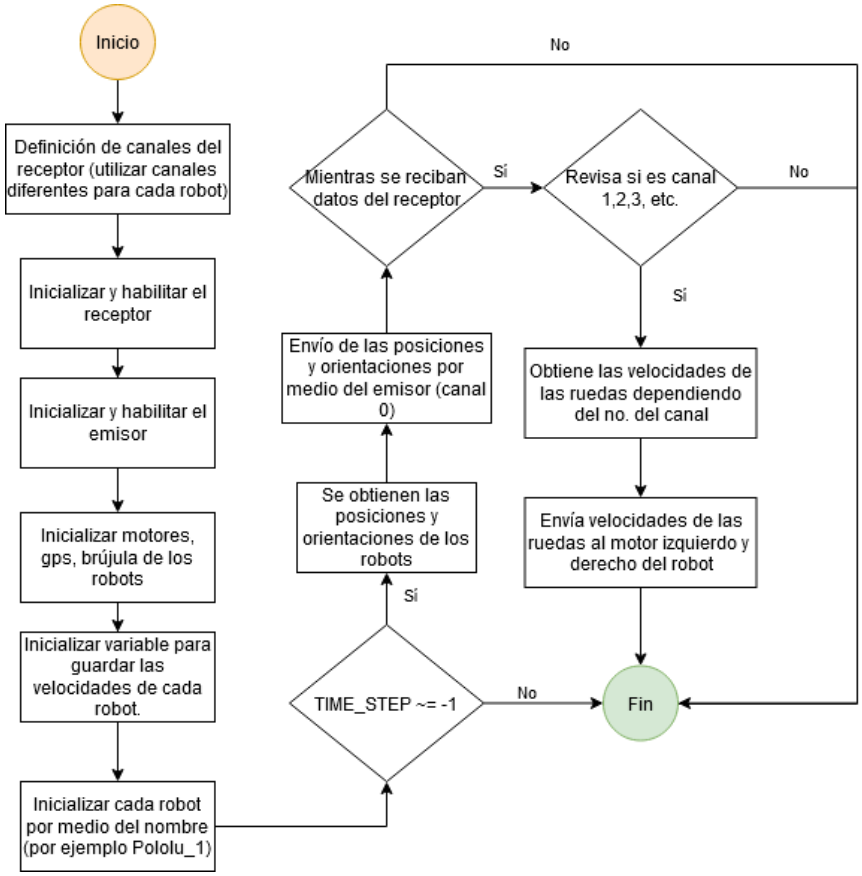
**Figura 29:** *Estructura del controlador principal.*




Nota. Elaboración propia.

Cada agente robótico está equipado con emisores, receptores, brújula, gps y motores. Estos elementos permiten simular a los agentes robóticos Pololu 3pi+, donde cada robot ejecuta su propio código para enviar, mediante el emisor, las posiciones y orientaciones, además de recibir las velocidades de las ruedas desde el supervisor (el servidor), como se muestra en la Figura 30.

**Figura 30:** *Estructura del controlador para agentes Pololu 3pi+ en Webots.*

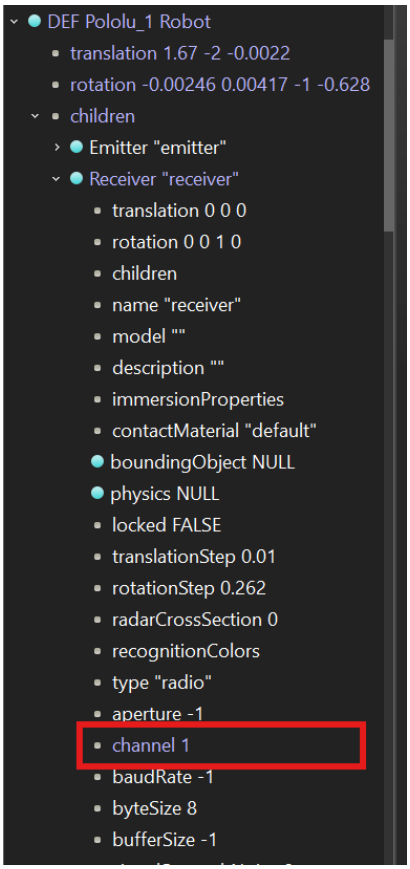



Nota. Elaboración propia.

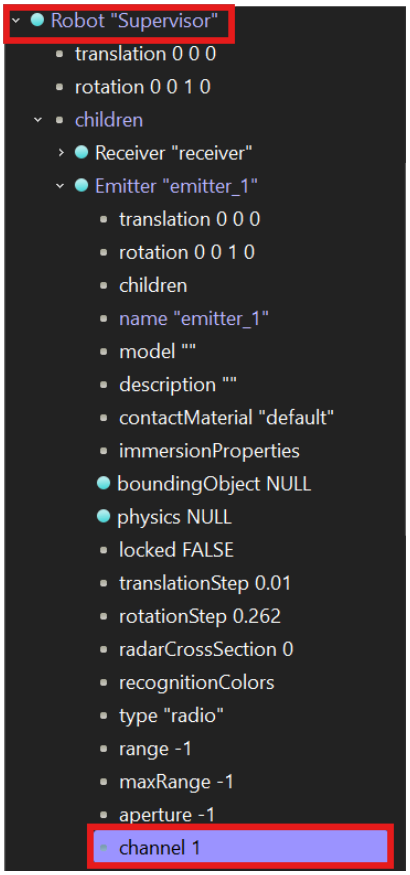
El modulo supervisor permite implementar las funciones *wb_supervisor_nodo_get_position*() y *wb_supervisor_nodo_get_orientation*() para obtener las posiciones y orientaciones de los robots. Para utilizar estas funciones, es necesario definir un nodo para cada robot mediante la función *wb_supervisor_node_get_from_det*(), que permite crear un nodo a partir de un DEF, es decir, un nombre asignado a cada robot.

La recepción de datos se realiza por medio de la función de receptor usando *wb_receiver_get_data*(). Esta función se debe de definir dentro de un ciclo while mientras se reciben los datos por medio de *wb_receiver_get_queue_length*() y recibe como argumento el canal de recepción para cada robot.

**Figura 31:** *Configuración del emisor y receptor en Webots.*




**(a)** *Configuración de la recepción agente robótico 1.*




**(b)** *Configuración del emisor agente robótico 1.*

Nota. Elaboración propia.

Es importante mencionar, que los canales de recepción y emisión de los agentes al robot supervisor deben los mismos, como se muestra en la Figura 31.

El envío de los datos se realiza por medio de la función de emisor usando *wb_ emitter_ send*(). Esta función recibe como argumento el canal del emisor establecido (canal 0), que debe establecerse para cada robot.

### 7.2.2. Pruebas con las distintas *fitness functions*

El funcionamiento de la simulación se verificó replicando las pruebas con las funciones de costo Sphere, Rosenbrock, Booth, Himmelblau, Schaffer no.4 y Keane con los parámetros utilizados en la fase anterior.

Es importante mencionar que estos experimentos se realizaron en una computadora Lenovo Legion 5 16IRX9, las caracteristicas se presentan en el Cuadro 5.

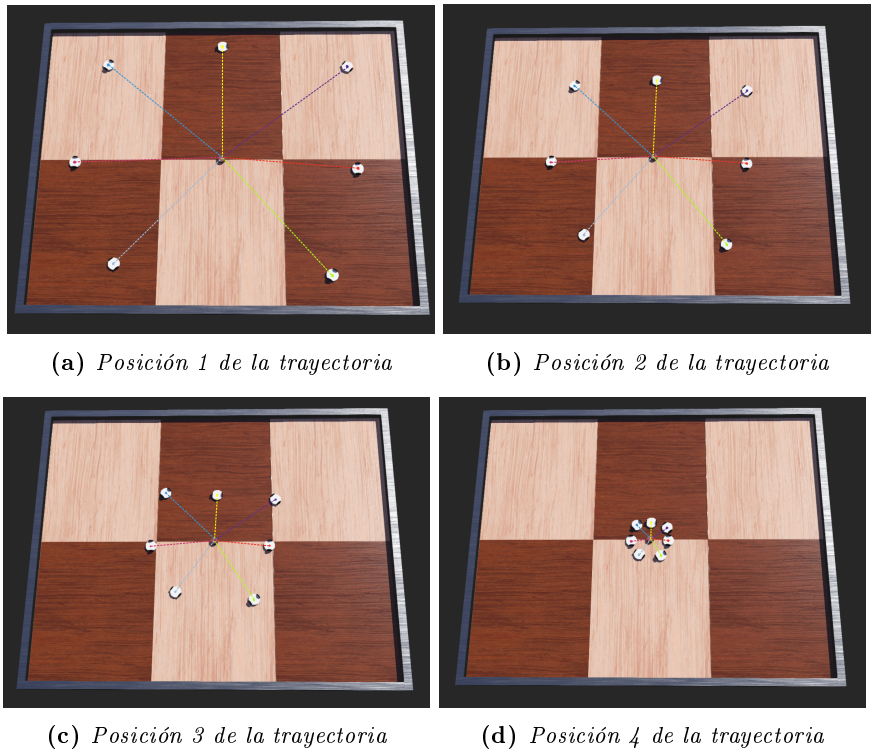
**Cuadro 5:** *Especificaciones de computadora Lenovo Legion 5 16IRX9.*

| | |
|---|---|
| **Procesador** | Intel® Core™ i7-14650HX@1.6GHz |
| **Gráficos** | NVIDIA® GeForce RTX™ 4060 8GB |
| **Memoria** | 16GB RAM DDR5-5600MHz |
| **Disco** | 512GB SSD PCIe® NVMe™ M.2 |

Nota. Elaboración propia.

El primer experimento realizado fue con la función de costo Sphere, esta función tiene un óptimo global en las coordenadas $(0, 0)$, como se muestra en la Figura 32. Se repetió el experimento cinco veces. Los tiempos de convergencia obtenidos se muestran en el Cuadro 6.

**Figura 32:** *Trayectorias generadas con la función Sphere en Webots.*



**(a)** *Posición 1 de la trayectoria* **(b)** *Posición 2 de la trayectoria*

**(c)** *Posición 3 de la trayectoria* **(d)** *Posición 4 de la trayectoria*

Nota. Elaboración propia.

**Cuadro 6:** *Tiempos de convergencia preliminares con función Sphere en Webots.*

| No. | Tiempos de convergencia (s) |
|---|---|
| 1 | 17.79 |
| 2 | 17.78 |
| 3 | 17.79 |
| 4 | 17.79 |
| 5 | 17.79 |

Nota. Elaboración propia.

El segundo experimento realizado fue con la función Booth, esta función tiene los mínimos globales descentrados del origen. Los tiempos de convergencia se muestran en el Cuadro 7.

**Cuadro 7:** *Tiempos de convergencia preliminares con función Booth en Webots.*

| No. | Tiempos de convergencia (s) |
|---|---|
| 1 | 24.16 |
| 2 | 24.16 |
| 3 | 24.16 |
| 4 | 24.16 |
| 5 | 24.16 |

Nota. Elaboración propia.

El tercer experimento realizado fue con la función Schaffer, esta función tiene los mínimos globales alejados del origen. Los tiempos de convergencia se muestran en el Cuadro 8.

La función de costo que presentó el menor tiempo de convergencia fue la función Sphere, mientras que la que requirió más tiempo para alcanza la meta fue la función Schaffer.

**Cuadro 8:** *Tiempos de convergencia preliminares con función Schaffer en Webots.*

| No. | Tiempos de convergencia (s) |
|---|---|
| 1 | 28.57 |
| 2 | 28.57 |
| 3 | 28.57 |
| 4 | 28.57 |
| 5 | 28.57 |

Nota. Elaboración propia.

# CAPÍTULO 8

# Implementación física del algoritmo MPSO

En este capítulo se describen los ajustes realizados al algoritmo MPSO para su funcionamiento en el ecosistema Robotat. También, se presentan las pruebas iniciales del algoritmo con los robots Pololu 3pi+ y otras modificaciones realizadas para facilitar la implementación.

## 8.1. Actualización del algoritmo MPSO

Al implementar el algoritmo por primera vez, los agentes róboticos no llegaban a la meta deseada. Por esta razón, se realizaron ajustes al algoritmo MPSO para asegurar su correcto funcionamiento. Las actualizaciones realizadas fueron en el controlador PID y las funciones que envían las velocidades a los robots.

### 8.1.1. Ajustes del controlador PID

En los robots diferenciales, la distancia entre ruedas se mide desde el punto de instalación de las mismas. En la fase anterior, se asumió que la distancia entre ruedas era de 96 mm; sin embargo, este valor no consideraba la ubicación de la instalación de las ruedas. El ajuste realizado consistió en considerar la distancia desde el punto de instalación de las ruedas, restando el ancho de ambas ruedas, como se muestra en la Ecuación 29 y Figura 33. La distancia entre ruedas fue de 96 $mm$ y el ancho de las ruedas fue de 6.8 $mm$.

$$\text{Distancia centros de las ruedas} = \frac{\text{Distancia ruedas} - 2 \times \text{Ancho ruedas}}{100} \tag{29}$$

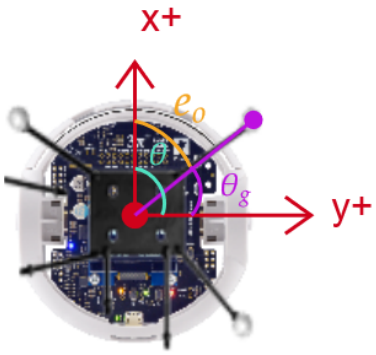
**Figura 33:** *Distancia entre ruedas del Pololu 3pi+.*


Nota. Elaboración propia.

Otro ajuste realizado consistió en modificar el cálculo del ángulo de orientación del robot. En el controlador original, el ángulo $\theta$ se obtenía sumando un *offset* al ángulo $z$ del marcador obtenido por el OptiTrack. La modificación implementada consistió en utilizar la Ecuación 24, la cual permite que la orientación en $z$ del marcador sea igual a cero, haciendo que coincida con el eje $x$ del robot como se muestra en la Figura 34.

$$e_o = \arctan 2 \left( \frac{\sin(\theta_g - \theta)}{cos(\theta_g - \theta)} \right) \tag{30}$$

**Figura 34:** *Alineación del ángulo del robot con el ángulo del marker.*




Nota. Elaboración propia.

Es importante mencionar que los ángulos se manejan en grados y no en radianes, ya que en las funciones utilizadas para obtener las coordenadas de orientación y posición del robot devuelven las orientaciones en grados.

### 8.1.2. Ajustes del envío de velocidades a los pololu 3pi+

Se crearon las funciones *pololu_3pi_set_velocities.m* y *pololu_3pi_force_stop.m*. Ambas funciones son similares a *robotat_3pi_set_wheel_velocities.m* y *robotat_3pi_force_stop.m*, con la única diferencia en la forma de acceder al *struct* que contiene el ID y puerto del robot.

Estas nuevas funciones permiten conectarse al robot por medio del número asginado, por medio de la función *evalin* de Matlab. Esta función permite acceder a las variables dentro del *workspace*, esto facilita el acceso al puerto y ID del robot.

## 8.2. Pruebas preliminares con el MPSO

Las primeras pruebas físicas realizadas con el MPSO y el ecosistema Robotat consistieron en replicar algunas de las pruebas de la fase anterior, utilizando los mismos parámetros. En estas pruebas se evaluaron el tiempo de convergencia y el *global best* encontrado, con un número máximo de 600 iteraciones.

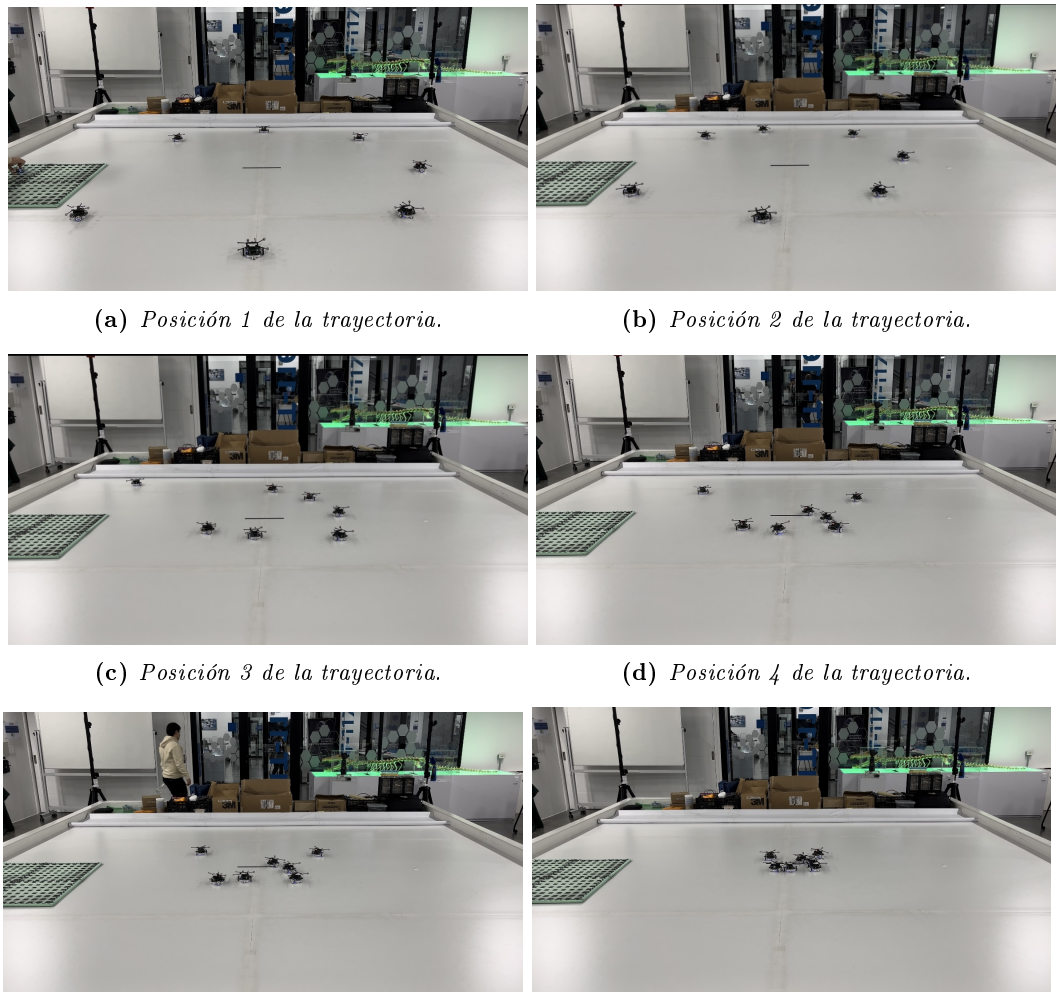
La primera *fitness function* utilizada fue la función Sphere, la cual tiene un máximo global en las coordenadas $(0, 0)$. Los parámetros utilizados se muestran en el Cuadro 9. En la Figura 35 se puede observar que todos los agentes robóticos logran llegar al origen del Robotat.

**Cuadro 9:** *Parámetros de prueba con función Sphere.*

| **Descripción** | **Ajuste seleccionado** |
|---|---|
| *Fitness function* | Sphere |
| Tipo de inercia | Exponencial |
| Factor de constricción | 0.2087 |
| Peso cognitivo | 2 |
| Peso social | 7 |
| Escalador de velocidad | 0.4 |

Nota. Elaboración propia.

**Figura 35:** *Trayectorias generadas por los robots Pololu 3pi+ durante la evaluación preliminar utilizando la función Sphere.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

**(e)** *Posición 5 de la trayectoria.* **(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

Las posiciones iniciales de cada robot se presentan en el Cuadro 10, mientras que los tiempos de convergencia y valores del *local best* para esta función se detallan en el Cuadro 11.

**Cuadro 10:** *Posiciones iniciales de los agentes Pololu 3pi+ durante la evaluación preliminar con función Sphere.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (0.7774, -0.8424) |
| 3 | (1.0987, -0.0592) |
| 4 | (-0.0382, 1.0809) |
| 5 | (-0.0061, -1.2660) |
| 6 | (-0.7309, 0.7860) |
| 7 | (-0.9044, -0.8706) |
| 8 | (0.8653, 0.7321) |

Nota. Elaboración propia.

**Cuadro 11:** *Pruebas preliminares con la función Sphere en el ecosistema Robotat.*

| No. | *Fitness function* | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Sphere | 59.0 | (-0.0423, -0.0410) |
| 2 | Sphere | 60.7 | (-0.0269, 0.0375) |
| 3 | Sphere | 65.0 | (0.0194, -0.0293) |
| 4 | Sphere | 59.0 | (0.0347, -0.0901) |
| 5 | Sphere | 60.5 | (0.0030, -0.1098) |

Nota. Elaboración propia.

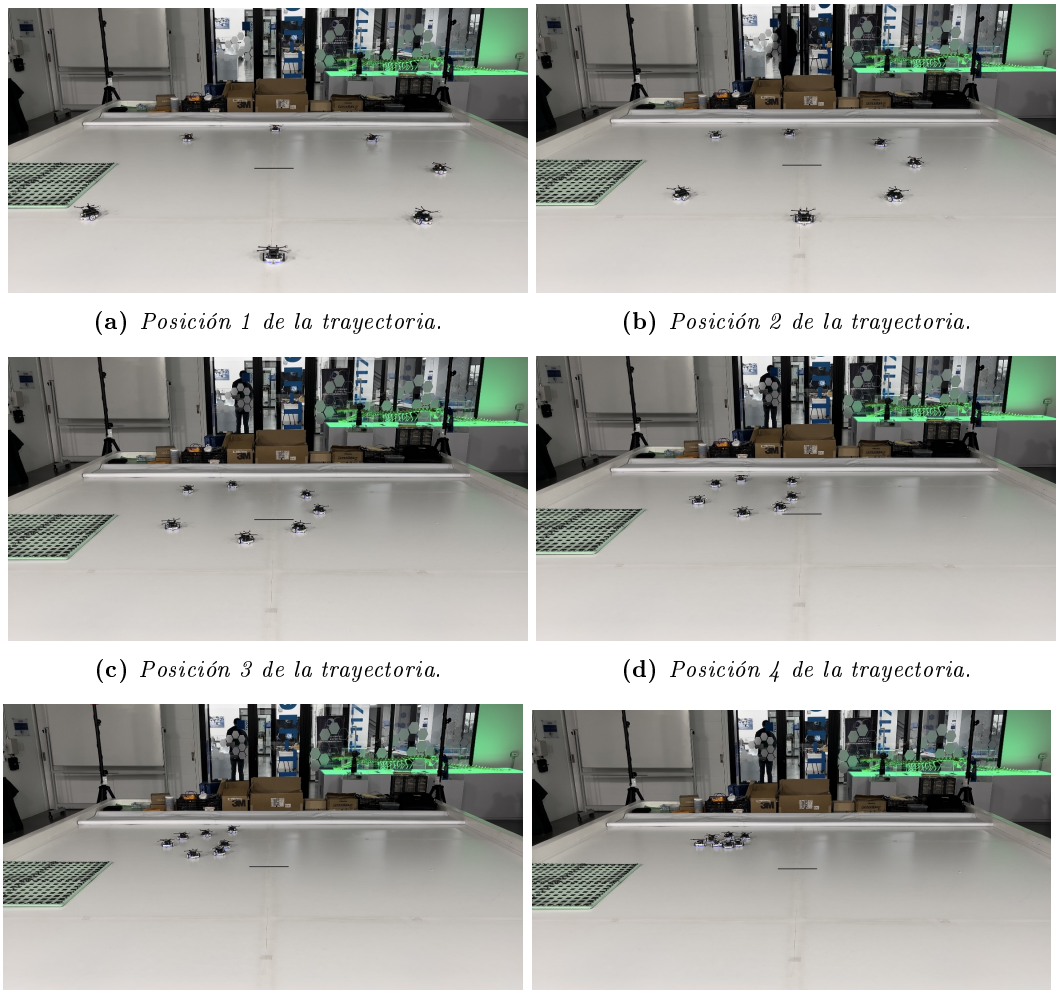
La segunda *fitness function* replicada fue la función Booth, que tiene un mínimo absoluto alejado del origen. Esta función fue evaluada con los parámetros presentados en el Cuadro 12. En la Figura 36 se observa que todos los agentes robóticos convergen al mínimo global, ubicado en la esquina superior izquierda del Robotat.

**Cuadro 12:** *Parámetros de prueba con función Booth.*

| Descripción | Ajuste seleccionado |
|---|---|
| *Fitness function* | Booth |
| Tipo de inercia | Constante |
| Factor de constricción | 0.2087 |
| Peso cognitivo | 1 |
| Peso social | 6 |
| Escalador de velocidad | 0.5 |

Nota. Elaboración propia.

**Figura 36:** *Trayectorias generadas por los robots Pololu 3pi+ durante la evaluación preliminar utilizando la función Booth.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

**(e)** *Posición 5 de la trayectoria.* **(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

Los tiempos de convergencia y *global best* obtenidos se presentan en el Cuadro 13. Las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 14.

**Cuadro 13:** *Pruebas preliminares con la función Booth en el ecosistema Robotat.*

| No. | *Fitness function* | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Booth | 36.80 | (0.9086, 0.7815) |
| 2 | Booth | 45.60 | (-0.7236, 0.7850) |
| 3 | Booth | 48.00 | (1.0923, -0.0504) |
| 4 | Booth | 41.04 | (0.7504, 0.9094) |
| 5 | Booth | 42.02 | (0.8737, 0.7693) |

Nota. Elaboración propia.

**Cuadro 14:** *Posiciones iniciales de los agentes Pololu 3pi+ durante la evaluación preliminar con función Booth.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (0.7610, -0.8598) |
| 3 | (-0.8812, -0.8247) |
| 4 | (-0.0087, 1.1001) |
| 5 | (-0.0299, -1.2454) |
| 6 | (0.8761, 0.7769) |
| 7 | (-0.7201, 0.7922) |
| 8 | (1.1097, -0.1238) |

Nota. Elaboración propia.

La tercera *fitness function* replicada fue la función Schaffer que presenta múltiples mínimos en forma de anillo alejados del origen. Esta función fue evaluada con los parámetros mostrados en el Cuadro 15.

**Cuadro 15:** *Parámetros de prueba con función Schaffer.*

| Descripción | Ajuste seleccionado |
|---|---|
| *Fitness function* | Schaffer |
| Tipo de inercia | Aleatoria |
| Factor de constricción | 0.2087 |
| Peso cognitivo | 2 |
| Peso social | 5 |
| Escalador de velocidad | 0.6 |

Nota. Elaboración propia.

Los tiempos de convergencia y los valores del *global best* obtenidos para esta función se presentan en el Cuadro 16. Adicionalmente, se presentan las posiciones iniciales de los agentes robóticos en el Cuadro 17.

**Cuadro 16:** *Pruebas preliminares con la función Schaffer en el ecosistema Robotat.*

| **No.** | ***Fitness function*** | **Tiempo (s)** | **Global best (m)** |
|---|---|---|---|
| 1 | Schaffer | 40.61 | (1.1343, -0.0896) |
| 2 | Schaffer | 47.2 | (1.1233, -0.0889) |
| 3 | Schaffer | 59.6 | (1.1283, -0.0879) |
| 4 | Schaffer | 57.03 | (1.1381, -0.1087) |
| 5 | Schaffer | 49.03 | (-1.3828, -0.2717) |

Nota. Elaboración propia.

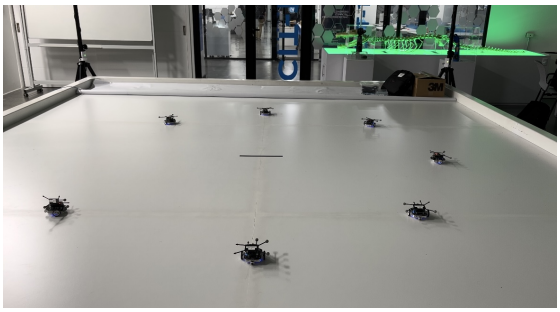
**Cuadro 17:** *Posiciones iniciales de los agentes Pololu 3pi+ durante la evaluación preliminar con función Schaffer.*

| **Robot** | **Posición inicial (m)** |
|---|---|
| 2 | (0.7774, -0.8424) |
| 3 | (1.0987, -0.0592) |
| 4 | (-0.0382, 1.0809) |
| 5 | (-0.0061, -1.2660) |
| 6 | (-0.9044, -0.8706) |
| 7 | (0.8653, 0.7321) |

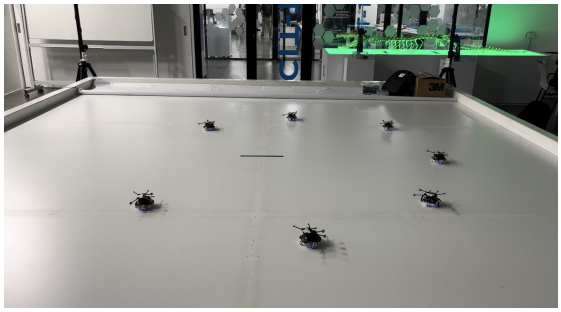
Nota. Elaboración propia.

En la Figura 37 se observa que todos los agentes robóticos convergen hacia el mínimo global en el eje $x$ positivo del Robotat.

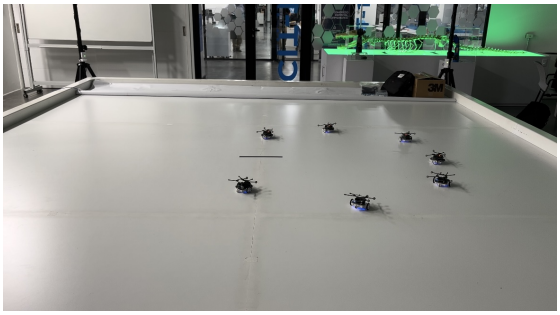
**Figura 37:** *Trayectorias generadas por los robots Pololu 3pi+ durante la evaluación preliminar utilizando la función Schaffer.*



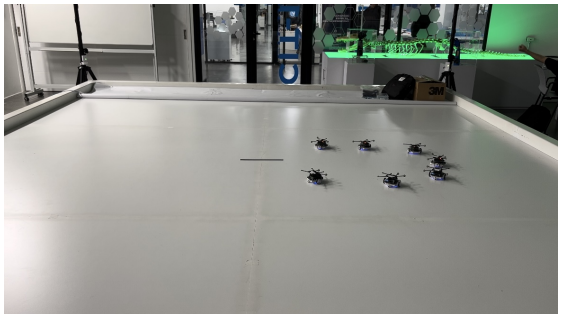
**(a)** *Posición 1 de la trayectoria.*



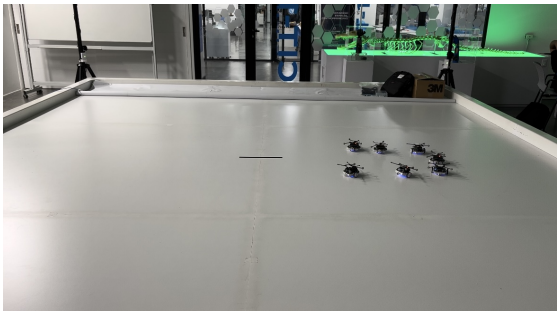
**(b)** *Posición 2 de la trayectoria.*



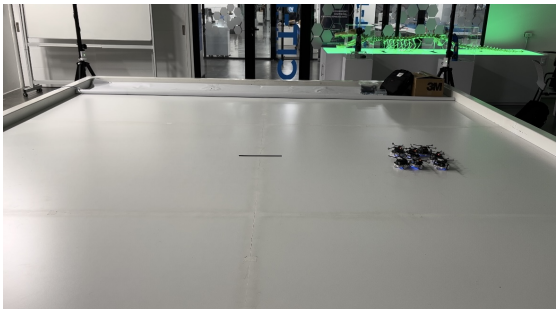
**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

Se compararon los tiempos promedio de convergencia de las funciones, estos se muestran en el Cuadro 18, junto con las mejores posiciones encontradas por todo el enjambre. La función con mayor tiempo de convergencia fue Sphere, ya que requirió más tiempo para explorar y encontrar una posible solución. La función con menor tiempo fue Booth.

**Cuadro 18:** *Tiempo promedio para las fitness functions.*

| ***Fitness function*** | **Tiempo promedio (s)** |
|---|---|
| Sphere | 60.84 |
| Booth | 42.692 |
| Schaffer | 50.698 |

Nota. Elaboración propia.

La mejor posición encontrada con la función Sphere se muestra en el Cuadro 19. De igual forma, para la función Booth se identificaron tres posibles soluciones que cumplen los mínimos locales de la función.

**Cuadro 19:** *Global best encontrados con la función Booth y Sphere.*

| ***Fitness function*** | **Global best promedio (m)** |
|---|---|
| Sphere | (0.0024, 0.0465) |
| Booth | (0.8442, 0.8201) |
| Booth | (-0.7236, 0.785) |
| Booth | (1.092, -0.0504) |

Nota. Elaboración propia.

Con la función Schaffer se encontraron dos posibles soluciones que cumplen con los mínimos locales de la función, estos se presentan en el Cuadro 20.

**Cuadro 20:** *Global best encontrados con la función Schaffer.*

| ***Fitness function*** | **Global best promedio (m)** |
|---|---|
| Schaffer | (1.131, -0.094) |
| Schaffer | (-1.3828, -0.2717) |

Nota. Elaboración propia.

### 8.2.1. Otras modificaciones del algoritmo MPSO

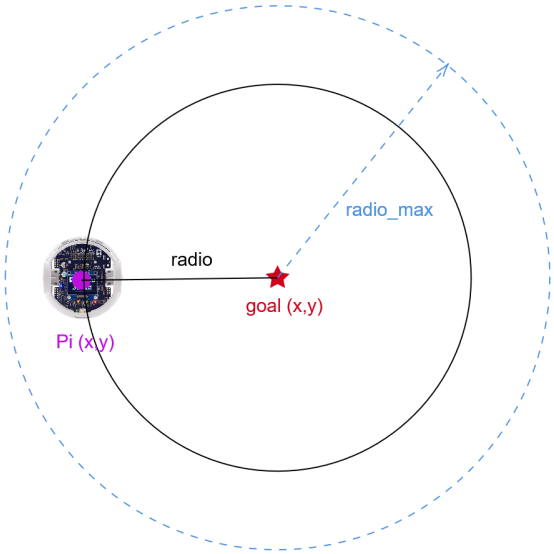
Al replicar las pruebas de la fase anterior con el algoritmo MPSO, se observó que no existía una forma concreta de determinar cuando los agentes llegaban a la meta. Por esta razón, se implementó un radio de convergencia para proporcionar una medida más clara de si los agentes robóticos lograban acercarse a un área cercana a la meta.

Esta modificación se implementó utilizando la ecuación de la circunferecia mostrada en la Ecuación 31. En donde $x$ y $y$ son las coordenadas del punto, $h, k$ son las coordenadas del centro de la circunferencia y $r$ es su radio.

$$r = \sqrt{(x-h)^2 + (y-k)^2} \tag{31}$$

La implementación de la Ecuación 31 en el algoritmo MPSO se realizó mediante un radio máximo de convergencia de 0.25 m. Las coordenadas $(x, y)$ representan la posición actual del robot, mientras que las coordenadas $h, k$ corresponden a la coordenada del best global. Luego, se calcula el radio: si $r$ es mayor al radio máximo, el robot no ha llegado al área de convergencia cerca de la meta. En cambio, si $r$ es menor al radio máximo, el robot ha llegado al área de convergencia y se detiene, esto se ilustra en la Figura 38.

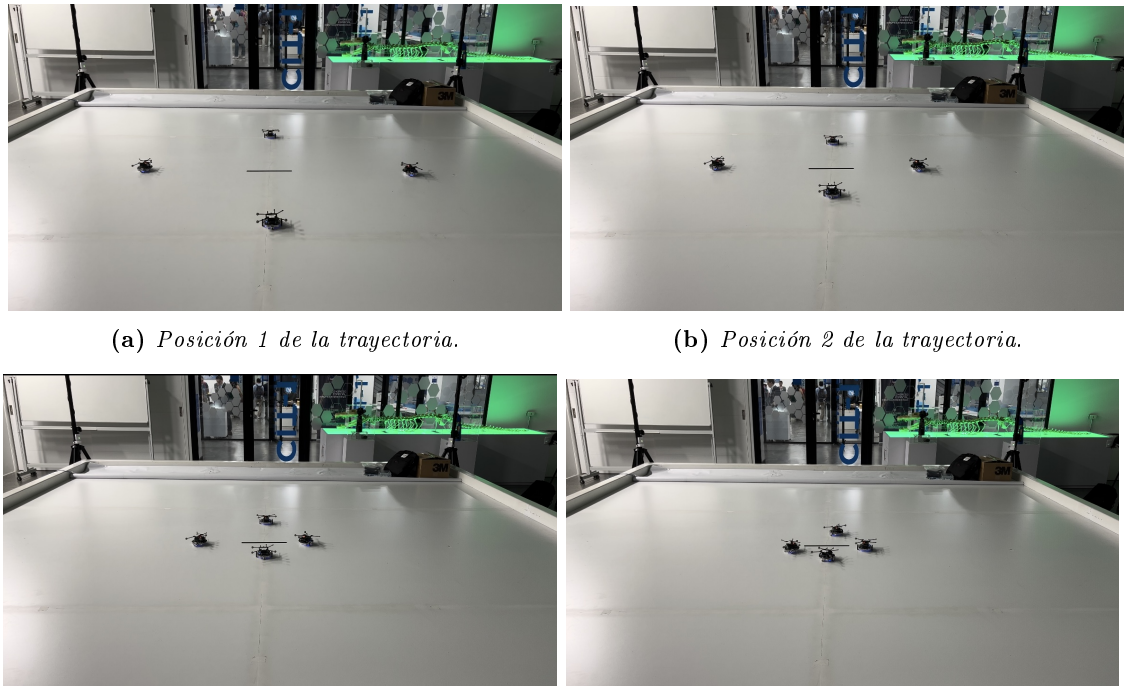
**Figura 38:** *Radio de convergencia.*




Nota. Elaboración propia.

La Figura 39 ilustra el funcionamiento del radio de convergencia utilizando la función Sphere y cuatro agentes Pololu 3pi+.

**Figura 39:** *Implementación del radio de convergencia con la función Sphere y cuatro agentes Pololu 3pi+.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

Con esta modificación, se replicaron las pruebas preliminares con las *fitness function*: Sphere, Booth y Schaffer, omitiendo el uso del ciclo for paralelizado para el cálculo del controlador PID. Asimismo, se midieron los tiempos de convergencia y el *global best* de cada función. Es importante mencionar que el *global best* encontrado puede no corresponder a un mínimo o máximo global, ya que solo se verificó que los agentes llegaron a un área cercana al punto de convergencia.

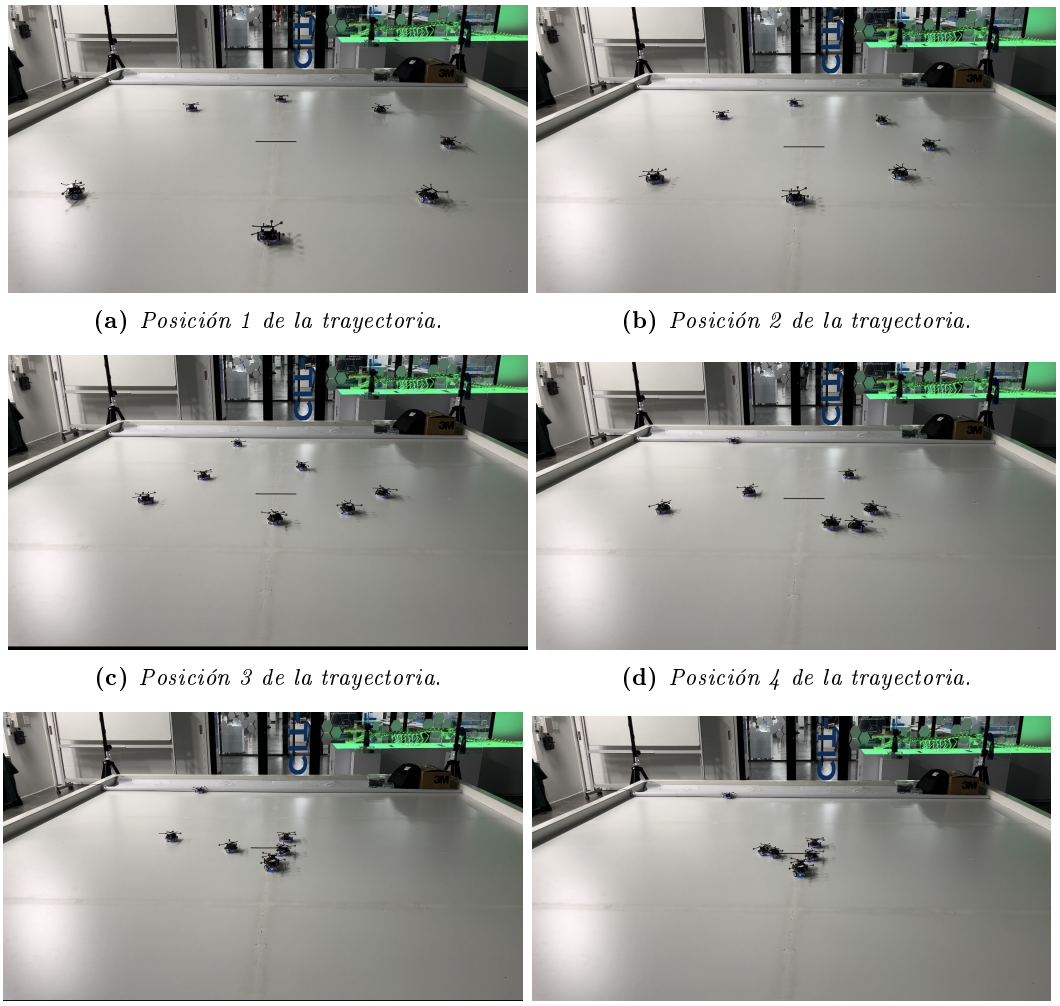
Al implementar la función Sphere, se observa que los *global best* encontrados son cercanos al máximo global, como se muestra en el Cuadro 21.

**Cuadro 21:** *Pruebas preliminares con el algoritmo MPSO modificado utilizando la función Sphere en el ecosistema Robotat.*

| No. | *Fitness function* | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Sphere | 51.02 | (0.1130, -0.0846) |
| 2 | Sphere | 53.06 | (0.1412, 0.2173) |
| 3 | Sphere | 59.00 | (0.1083, -0.0268) |
| 4 | Sphere | 55.02 | (-0.1041, 0.0527) |
| 5 | Sphere | 58.54 | (-0.0620, -0.1324) |

Nota. Elaboración propia.

**Figura 40:** *Trayectoria generada con la función Sphere y el radio de convergencia en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

**(e)** *Posición 5 de la trayectoria.* **(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

En la Figura 40, se puede observar que casi todos los agentes logran llegar a un área de convergencia específica, solo un agente no alcanzó la meta encontrada. Las posiciones iniciales de los agentes robóticos se muestran en el Cuadro 22.

**Cuadro 22:** *Posiciones iniciales de los agentes Pololu 3pi+ con el algoritmo MPSO modificado con función Sphere en el ecosistema Robotat.*

| **Robot** | **Posición inicial (m)** |
|---|---|
| 2 | (0.01666, 1.0945) |
| 3 | (1.0922, -0.0747) |
| 4 | (0.8055, 0.7508) |
| 5 | (-0.0033, -1.2533) |
| 6 | (-0.9334, -0.8298) |
| 7 | (-0.6974, 0.7956) |

Nota. Elaboración propia.

Para la función Booth, los *global best* encontrados están cercanos a un mínimo global de la función, como se muestra en el Cuadro 23. En la Figura 41, se observa que todos los agentes robóticos logran llegar al área de convergencia, cerca del mínimo global encontrado. Las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 24.

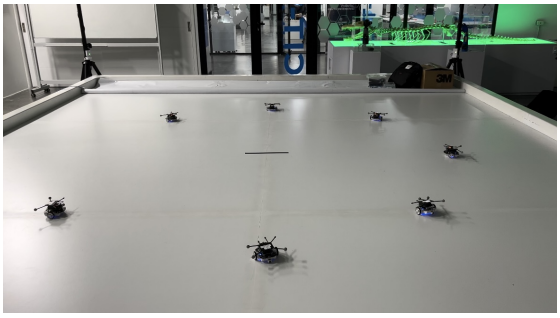
**Cuadro 23:** *Pruebas oreliminares con el algoritmo MPSO modificado utilizando la función Booth en el ecosistema Robotat.*

| **No.** | ***Fitness function*** | **Tiempo (s)** | **Global best (m)** |
|---|---|---|---|
| 1 | Booth | 51.05 | (-0.7261, 0.7976) |
| 2 | Booth | 40.02 | (0.8203, 0.7923) |
| 3 | Booth | 55.51 | (0.8200, 0.7919) |
| 4 | Booth | 53.91 | (-0.7430, 0.8200) |
| 5 | Booth | 58.25 | (-0.7414, 0.8179) |

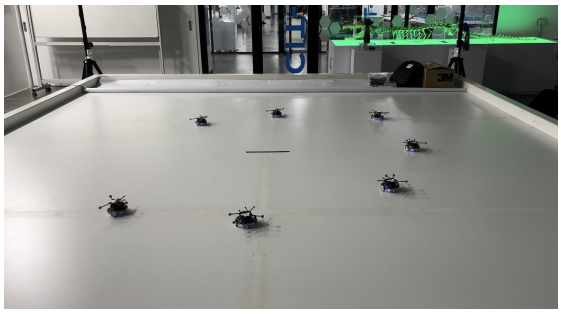
Nota. Elaboración propia.

Para la función Schaffer, se observa que todos los *global best* están cerca de un mínimo global, como se muestra en el Cuadro 25. En la Figura 42, se puede ver que casi todos los agentes logran converger al área esperada, solo un agente no no alcanzó el área de convergencia.

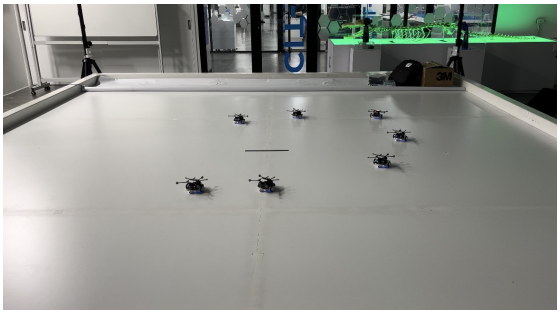
**Figura 41:** *Trayectoria generada con la función Booth y el radio de convergencia en el ecosistema Robotat.*



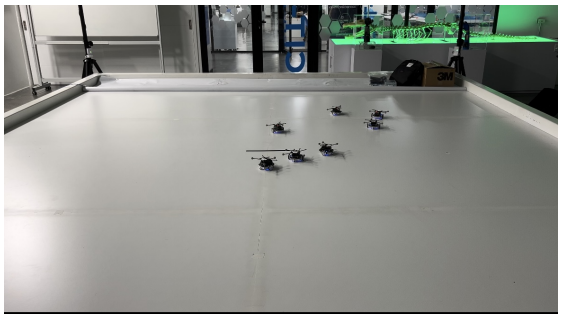
**(a)** *Posición 1 de la trayectoria.*



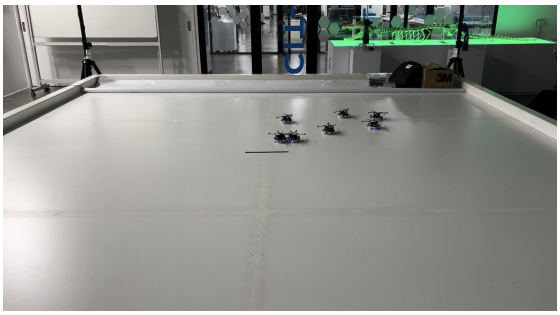
**(b)** *Posición 2 de la trayectoria.*



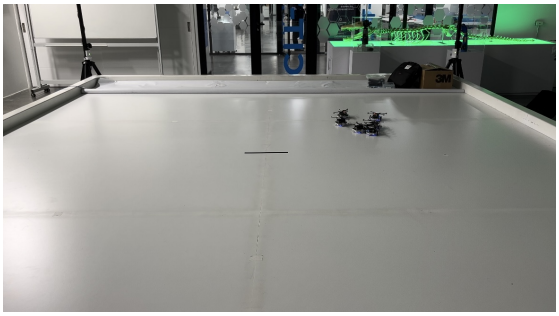
**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Cuadro 24:** *Posiciones iniciales de los agentes Pololu 3pi+ con el algoritmo MPSO modificado utilizando función Booth en el ecosistema Robotat.*

| **Robot** | **Posición inicial (m)** |
|---|---|
| 2 | (0.0053, 1.1181) |
| 3 | (1.0981, -0.0530) |
| 4 | (-0.8702, -0.8585) |
| 5 | (0.8918, 0.7803) |
| 6 | (-0.7261, 0.7976) |
| 7 | (0.0107, -1.2501) |
| 8 | (0.7375, -0.8659) |

Nota. Elaboración propia.

**Cuadro 25:** *Pruebas preliminares con el algoritmo MPSO modificado utilizando la función Schaffer en el ecosistema Robotat.*

| **No.** | ***Fitness function*** | **Tiempo (s)** | **Global best (x,y)** |
|---|---|---|---|
| 1 | Schaffer | 40.23 | (1.183, -0.0871) |
| 2 | Schaffer | 41.11 | (1.1162, -0.0612) |
| 3 | Schaffer | 50.44 | (-0.0062, 1.094) |
| 4 | Schaffer | 43.32 | (1.319, 0.2701) |
| 5 | Schaffer | 38.20 | (1.1241, -0.0996) |

Nota. Elaboración propia.

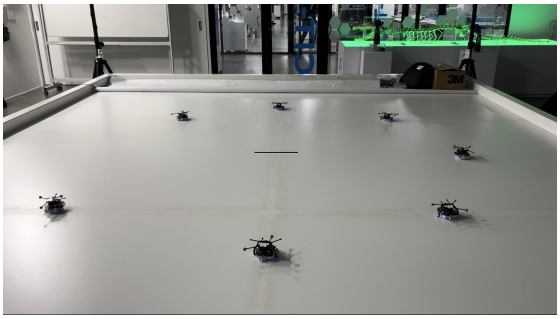
Al comparar el tiempo promedio de cada una de las pruebas Cuadro 26, se observa que la función con menor tiempo de convergencia fue la Schaffer, ya que para los agentes fue más sencillo llegar al área de convergencia del mínimo global.

**Cuadro 26:** *Tiempo promedio de las tres pruebas con el radio de convergencia.*

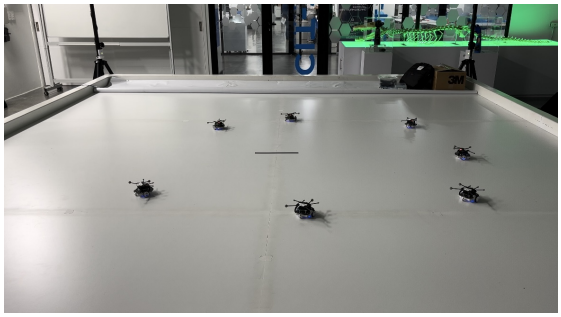
| ***Fitness function*** | **Tiempo promedio (s)** |
|---|---|
| Sphere | 55.328 |
| Booth | 51.748 |
| Schaffer | 42.66 |

Nota. Elaboración propia.

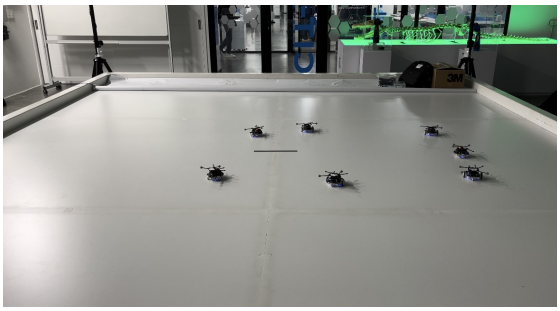
**Figura 42:** *Trayectoria generada con la función Schaffer y el radio de convergencia.*



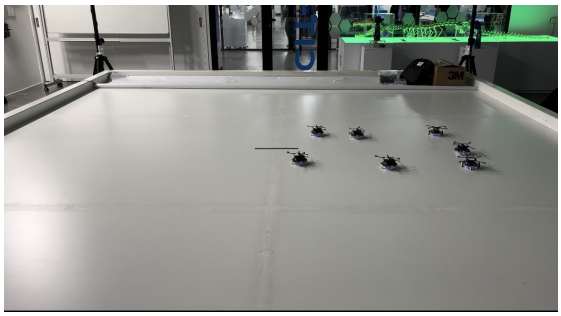
**(a)** *Posición 1 de la trayectoria.*



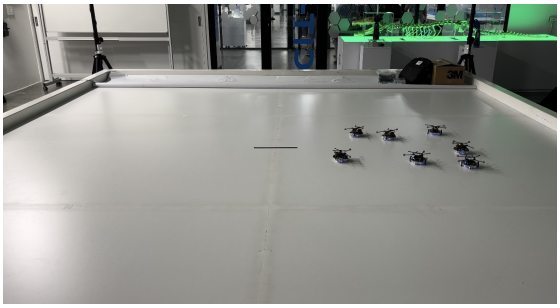
**(b)** *Posición 2 de la trayectoria.*



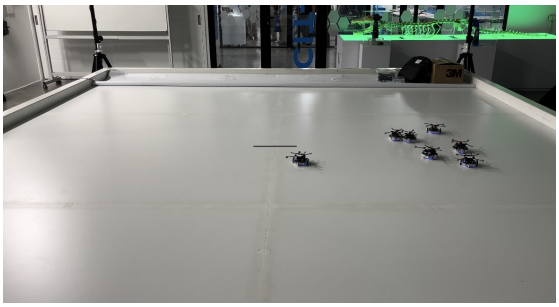
**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

# CAPÍTULO 9

## Optimización de la implementación del algoritmo MPSO

El primer objetivo de este trabajo fue evaluar el algoritmo MPSO e identificar posibles puntos de mejora en el código, lenguaje de programación y métodos de comunicación con el ecosistema Robotat. Con el fin de optimizar el rendimiento, reduciendo el tiempo de convergencia y tiempo de ejecución del algoritmo MPSO. En este capítulo se presentan las mejoras implementadas, así como las diversas pruebas realizadas, tanto simuladas como físicas.

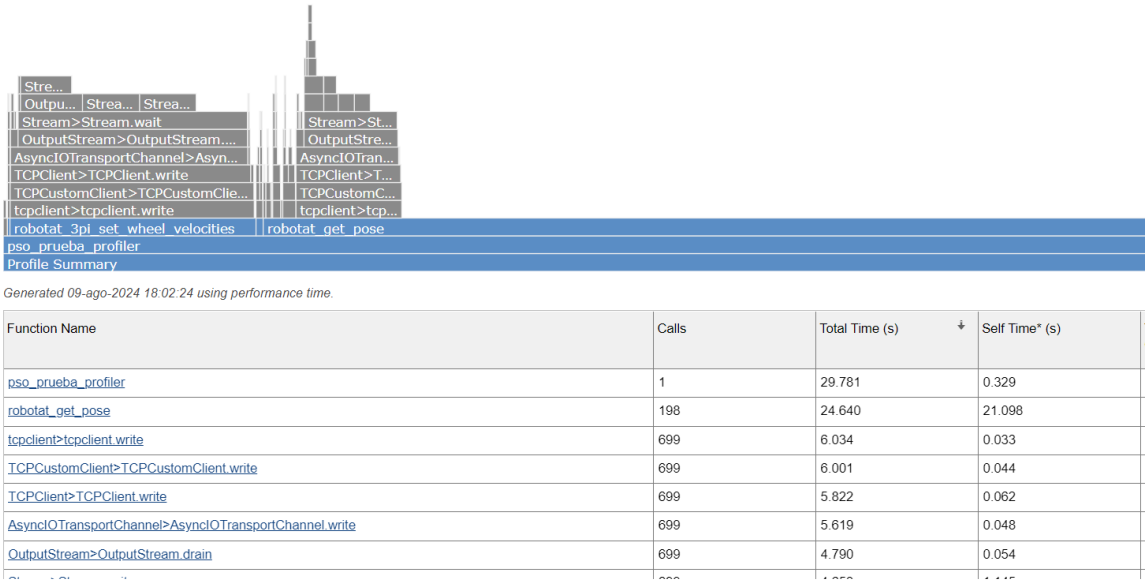
## 9.1. *Matlab profiler*

La herramienta *profiler* de matlab permite medir el tiempo de ejecución de un *script*. Identifica las funciones que requieren más tiempo de ejecución y evalúa posibles mejoras para aumentar el rendimiento del código.

Se utilizó la herramienta *profiler* para evaluar el rendimiento del algoritmo MPSO. En la Figura 43 se presenta los resultados obteidos mediante un gráfico llamado (*flame graph*). Este gráfico muestra las funciones funciones que requirieron más tiempo en ejecutarse:

- ***pso_prueba_profiler.m*** con 29.781 segundos.
- ***robotat_get_pose.m*** con 24.640 segundos.

**Figura 43:** *Gráfico de llamas de la herramienta the profiler de Matlab.*



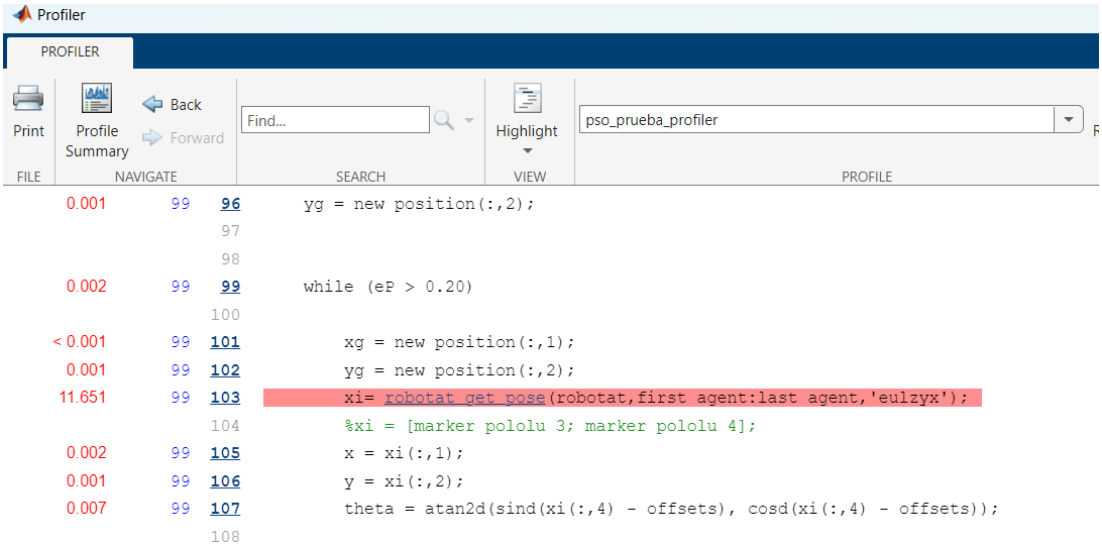

*Generated 09-ago-2024 18:02:24 using performance time.*

| Function Name | Calls | Total Time (s) | Self Time* (s) |
|---|---|---|---|
| pso_prueba_profiler | 1 | 29.781 | 0.329 |
| robotat_get_pose | 198 | 24.640 | 21.098 |
| tcpclient>tcpclient.write | 699 | 6.034 | 0.033 |
| TCPCustomClient>TCPCustomClient.write | 699 | 6.001 | 0.044 |
| TCPClient>TCPClient.write | 699 | 5.822 | 0.062 |
| AsyncIOTransportChannel>AsyncIOTransportChannel.write | 699 | 5.619 | 0.048 |
| OutputStream>OutputStream.drain | 699 | 4.790 | 0.054 |

Nota. Elaboración propia.

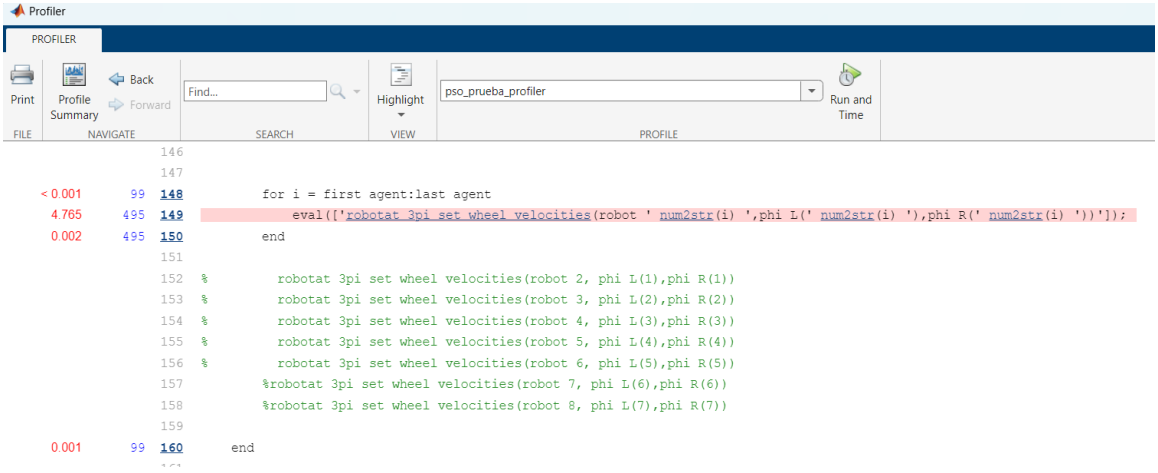
Al ejecutar la herramienta, se observó que la función *pso_prueba_profiler.m* presenta un mayor tiempo de ejecución, debido a las funciones para obtener las posiciones y envío de velocidades a las ruedas de los robots. Los tiempos obtenidos se muestran en la Figuras 44 y 45.

**Figura 44:** *Función robotat_get_pose() evaluada con la herramienta Profiler.*




Nota. Elaboración propia.

**Figura 45:** *Función robotat_3pi_set_wheel_velocities() evaluada con la herramienta Profiler.*




Nota. Elaboración propia.

## 9.2. Protocolos de comunicación

Actualmente, el protocolo TCP es utilizado para establecer la conexión con el servidor del Robotat y obtener las posiciones y orientaciones de los marcadores que se encuentran sobre los agentes robóticos Pololu 3pi+. A continuación, se detallan los pasos para realizar una conexión entre el servidor Robotat y el cliente (Matlab):

- **Establecimiento de conexión al servidor:** para establecer la conexión se implementa el método *Three-way Handshake*, este permite al cliente el envío de un mensaje al servidor del Robotat por medio de una dirección IP y un puerto específico para establecer la conexión. Si el servidor está disponible, responde con un mensaje confirmando que está listo para comunicarse. Por último, el cliente (Matlab) responde para completar la conexión.
- **Transferencia de datos:** la transmisión de datos se realiza mediante la función *robotat_get_pose*(), esta envía los números de los agentes y recibe las posiciones y orientaciones de cada robot por medio de la conexión TCP. El servidor Robotat envía los datos en formato *JSON*, luego la función decodifica y procesa, convirtiendo los datos en la matriz de posición y orientación de los agentes robóticos.

Para el envío de las velocidades a los agentes robóticos, se establece una conexión con el protocolo TCP mediante un número de identificación de cada robot. A continuación, se detallan los pasos para realizar una conexión entre el servidor y los agentes robóticos Pololu 3pi+:

- **Establecimiento de conexión con los agentes robóticos:** la conexión a los agentes robóticos Pololu 3pi+ se realiza por medio de la función *robotat_ 3pi_ connect*(), esta mediante un número de identificación permite conectarse a una dirección IP y un puerto específico, para luego crear un objeto de conexión TCP.
- **Transferencia de datos entre el servidor y los robots:** la transmisión de datos se realiza mediante la función *robotat_ 3pi_ set_ wheel_ velocities*(), esta permite enviar las velocidades de las ruedas a cada uno de los robots por medio de una conexión TCP a los ESP32 equipados en cada agente robótico Pololu 3pi+. El mensaje se envía al ESP32 en formato binario.

### 9.2.1. Problemas y limitantes con el protocolo TCP

Durante la implementación de la infraestructura del ecosistema Robotat, se han observado diversos problemas y limitantes con el protocolo TCP, algunos se detallan a continuación:

- **Latencia:** este protocolo garantiza la entrega ordenada de datos mediante el uso de retransmisiones y confirmaciones. Este mecanismo genera retrasos en la obtención repetitiva de las posiciones y orientaciones de múltiples agentes robóticos.
- **Sobrecarga del protocolo:** cuando un gran número de personas se conectan al Robotat, se genera congestión al intentar obtener los datos, lo que provoca un aumento de latencia y errores en los datos. Además, la sobrecarga del protocolo puede saturar el servidor, llegando al punto de requerir un reinicio para restaurar su funcionamiento.
- **Desconexión con el protocolo TCP:** en varias ocasiones, la conexión con el servidor se ha perdido, siendo necesario restablecerla mediante la función *robotat_ connect*(). De manera similar, también se pierde la conexión con los agentes robóticos, lo que requiere reiniciar y volver a establecer la conexión con los agentes robóticos Pololu 3pi+.

### 9.2.2. Protocolo UDP

El protocolo UDP conocido como *User datagram protocol* permite la transmisión de datos de forma rápida en redes IP. Algunas características importantes de este protocolo:

- Este protocolo funciona sin conexión, permite el envío de datos por medio de la red sin que se haya establecido previamente una conexión entre el emisor y receptor.
- Este protocolo utiliza puertos al igual que el TCP.
- Este protocolo permite una comunicación rápida y sin retardos.
- Este protocolo no ofrece ninguna garantia de seguridad e integrad de los datos, tampoco garantiza el orden de los paquetes enviados.

### 9.2.3. Protocolo MQTT

El protocolo MQTT es un protocolo de mensajería eficiente y ligero diseñado para la comunicación entre dispositivos con limitados recursos o ancho de banda como el internet de las cosas (IoT). Se apoya con un protocolo de transporte normalmente como el TCP/IP. Algunas características importantes de este protocolo:

- **Ligero y eficiente:** los clientes MQTT son muy pequeños, requieren recursos mínimos, por lo que se puede utilizar microcontroladores pequeños.
- **Comunicaciones bidireccionales:** permite la mensajería entre dispositivo a nube y nube a dispositivo.
- **Seguridad habilitada:** el protocolo MQTT facilita el cifrado de mensajes mediante TLS y la autenticación de clientes mediante protocolos de autenticación modernos como OAuth.
- **Entrega de mensajes confiable:** el procolo MQTT tiene tres niveles de calidad de servicios definidos: 0 - como máximo una vez, 1 - al menos una vez, 2 - exactamente una vez.

### 9.2.4. Protocolo RTP

El protocolo RTP, también llamado protocolo en tiempo real, se utiliza para transmitir datos por medio del uso de protocolos como UDP para gestionar de forma eficiente los paquetes de datos. El RTP garantiza que los datos lleguen en orden y a tiempo, incluso si los paquetes toman diferentes caminos a través de la red.

### 9.2.5. *Robot operating system* (ROS)

ROS son un conjunto de bibliotecas y herramientas de software, permitiendo la creación de nodos, es decir, un único sistema ROS puede estar formado por nodos que corren sobre diferentes máquinas físicas. Para la comunicación entre nodos se utiliza el modelo publicador/suscriptor y a los canales de comunicación se les denomina *topics*. Por lo tanto, un nodo puedo publicar información en uno o varios *topics* y al mismo tiempo tener suscripciones a otros *topics* para recibir información que otros nodos publican. Para la coordinación de todo el sistema es necesario un nodo *Master*.

A continuación, se presentan ventajas de utilizar este sistema para aplicaciones de robótica, como el manejo de robots diferenciales:

- Facilita y acelera la programación de aplicaciones de robótica al no tener que empezar de cero construyendo un sistema propio para cada proyecto. ROS aporta un sistema básico de comunicación y programación que facilita el proceso de programación de robots.
- Uno de los objetivos de ROS es proporcionar un sistema que permita la programación de robots evitando su obsolescencia a corto plazo, utilizando protocolos de comunicación TCP/IP.

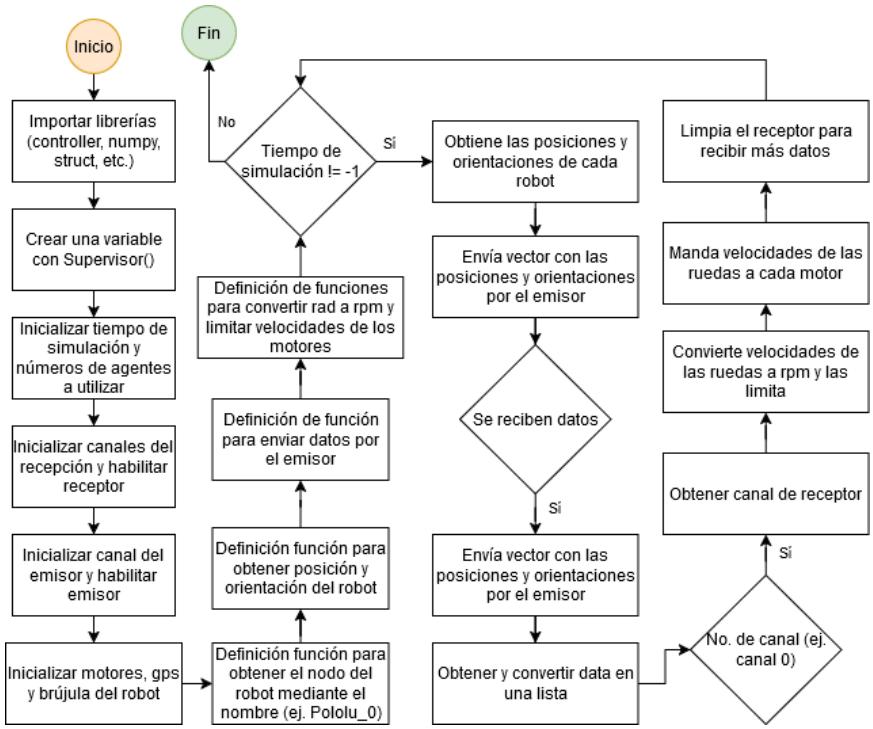
## 9.3. Migración del algoritmo MPSO a python

Otra posible solución para mejorar el tiempo de ejecución fue migrar el algoritmo MPSO al lenguaje de programación Python. La estructura del algoritmo migrado a Python se divide en el controlador para cada agente robótico encargado de enviar las posiciones y orientaciones de los robots, ilustrado en la Figura 46. Y el controlador principal encargado de realizar el cálculo del algoritmo PSO y envío de las velocidades de las ruedas a cada robot, ilustrado en la Figura 47.

Cabe mencionar que la migración se realizó únicamente en el entorno de simulación Webots, al igual que las pruebas y comparaciones entre los tiempos de convergencia. A continuación, se presentan algunos puntos importantes en la migración del algoritmo MPSO al lenguaje Python:

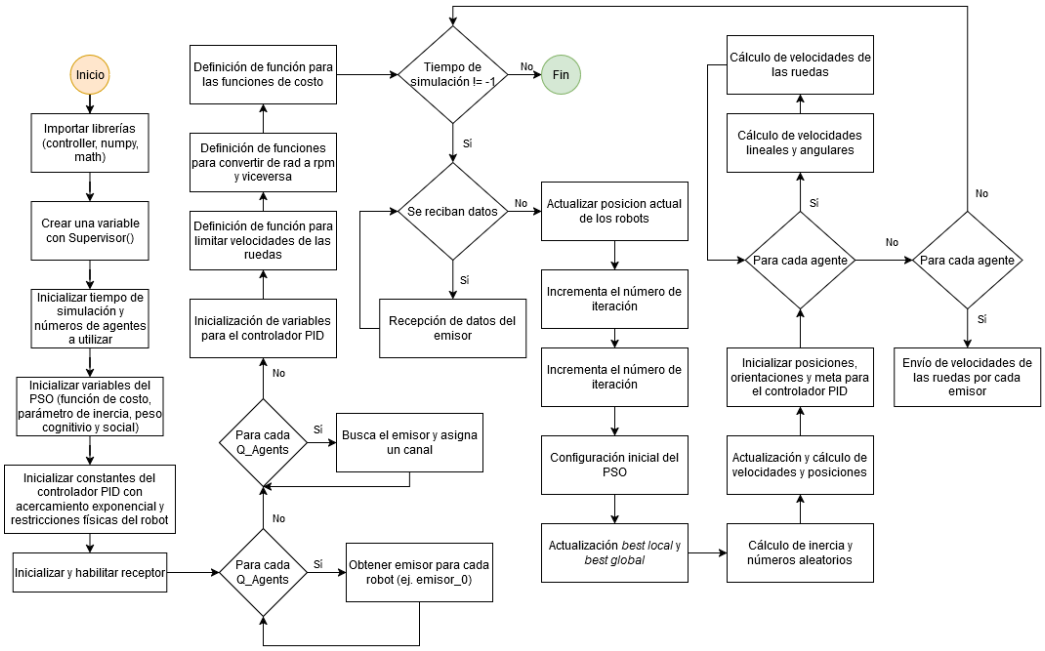
- En el simulador Webots únicamente se puede referir a un nodo definido como robot o supervisor.
- Se utilizó *numpy* para definir las operaciones matriciales y vectoriales.
- La indexación en Python para vectores y matrices comienza en 0.
- Se implementó el controlador PID vectorizado en lugar del controlador original.
- El envío y la recepción de las velocidades de las ruedas se realiza en formato *float* y *string*.
- Se crearon funciones para obtener las posiciones y orientaciones de los robots, así como para el envío de datos al controlador principal.

**Figura 46:** *Diagrama de flujo del controlador migrado a Python para cada agente robótico.*




Nota. Elaboración propia.

**Figura 47:** *Diagrama de flujo del controlador principal migrado a Python.*




Nota. Elaboración propia.

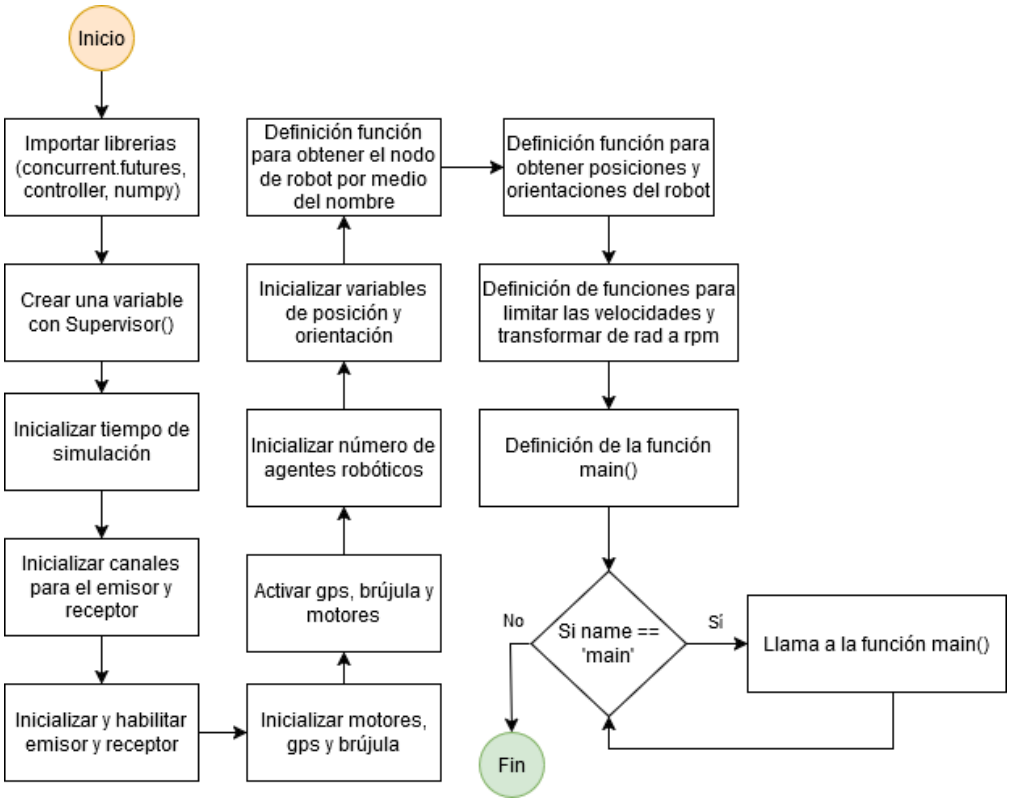
## 9.4. Paralelización

La paralelización consiste en implementar múltiples núcleos de procesamiento para acelerar los cálculos, distribuyendo las tareas entre varios trabajadores (*workers*). Matlab ofrece una librería (*toolbox*) específica que permite implementar ciclos "forçon paralelización. Al utilizarlos, Matlab ejecuta las operaciones dentro del ciclo en paralelo istribuyendo la carga de trabajo según los trabajadores disponibles. Cada trabajador realiza sus iteraciones de forma independiente y en orden aleatorio.

El algoritmo MPSO original utilizó ciclos "for"paralelizados para calcular las velocidades con la función del controlador PID, como se muestra en el pseudocógido presentado en la sección 7.1.4. Durante las pruebas, se observó que la implementación de ciclos "for"paralelizados con las funciones del Robotat generaban errores al intentar conectarse con el servidor utilizando el protocolo TCP. Por ese motivo, se optó por utilizar paralelización en el algoritmo MPSO migrado a Python como alternativa.

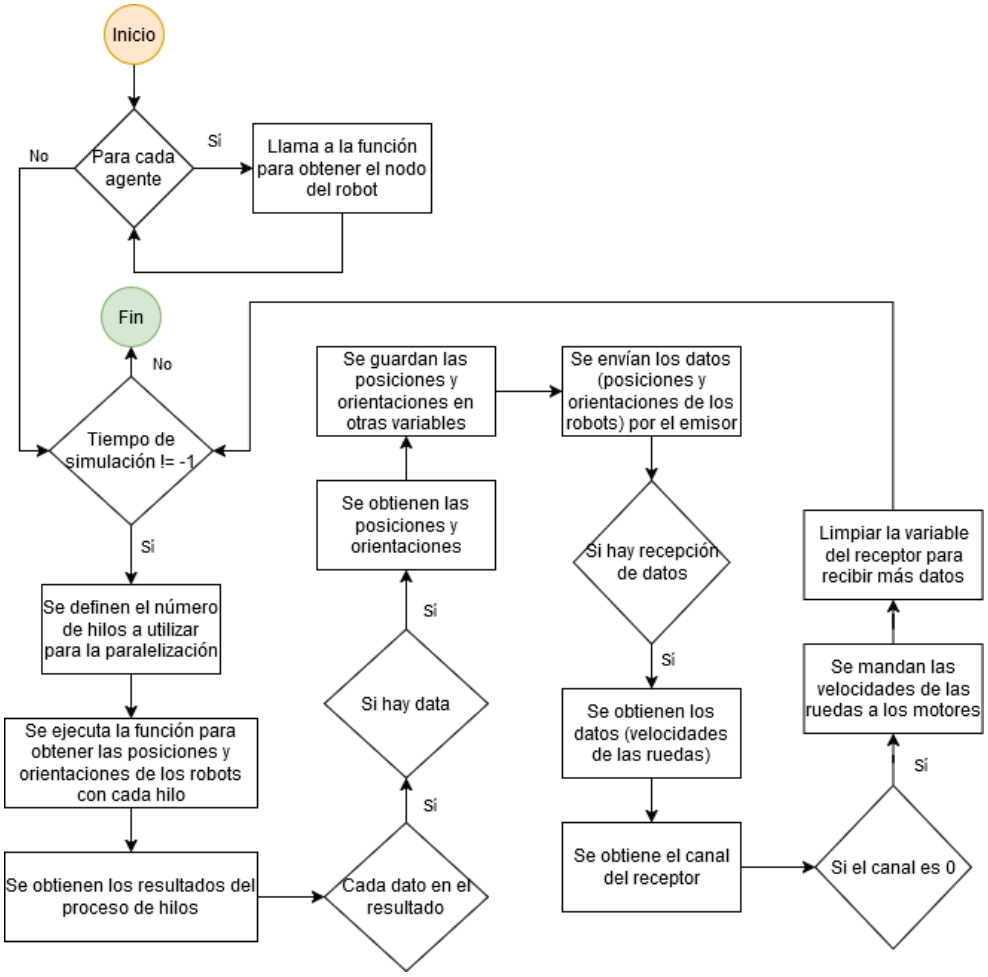
Se implementó el método de paralelización en el controlador encargado de enviar las coordenadas de los robots y de recibir las velocidades de las ruedas. En las Figura 48 y Figura 49, se muestra la estructura implementada para el envío de las posiciones y orientaciones de cada robot utilizando hilos. Se creó un grupo de hilos, donde cada robot se ejecuta de manera independiente en un hilo en específico.

**Figura 48:** *Diagrama de flujo del controlador paralelizado migrado a Python de cada agente robótico.*




Nota. Elaboración propia.

**Figura 49:** *Diagrama de flujo de la función main migrada a Python.*




Nota. Elaboración propia.

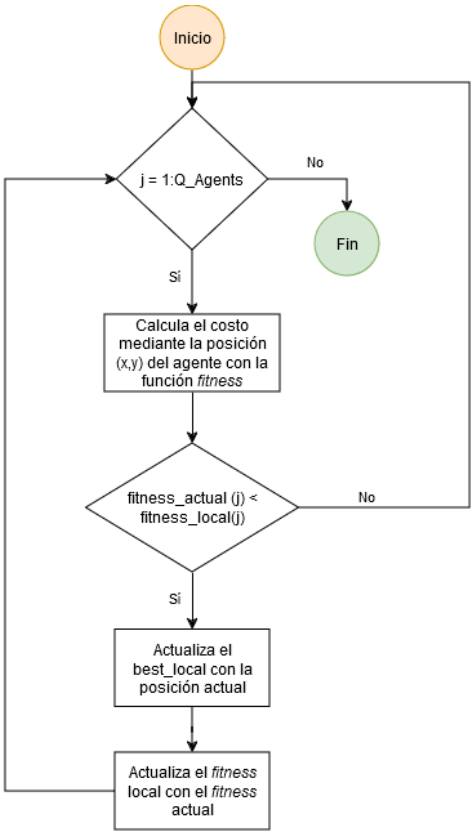
## 9.5. Vectorización

La vectorización consiste en optimizar operaciones, como los bucles "for", mediante el uso de operaciones matriciales y vectoriales, tales como suma, resta, funciones trigonométricas (seno, coseno), operaciones de algebra lineal (cálculo de norma), entre otras. Esta técnica permite realizar dichas operaciones de manera simultánea sobre un conjunto de valores organizados en forma de vector o matriz. Al analizar el algoritmo MPSO, se identificó un uso excesivo de ciclos "for", lo que llevó a la implementar vectorización para mejorar el rendimiento del algoritmo.

### 9.5.1. Vectorización de la *fitness function*

La función *fitness* evalúa las posiciones utilizando diferentes tipos de funciones de costo, como Sphere, Booth, Schaffer, entre otras. En su implementación inicial, el algoritmo calcula el costo de las posiciones para cada robot mediante un ciclo "for", como se muestra en la Figura 50.

**Figura 50:** *Diagrama de flujo de la función fitness original.*

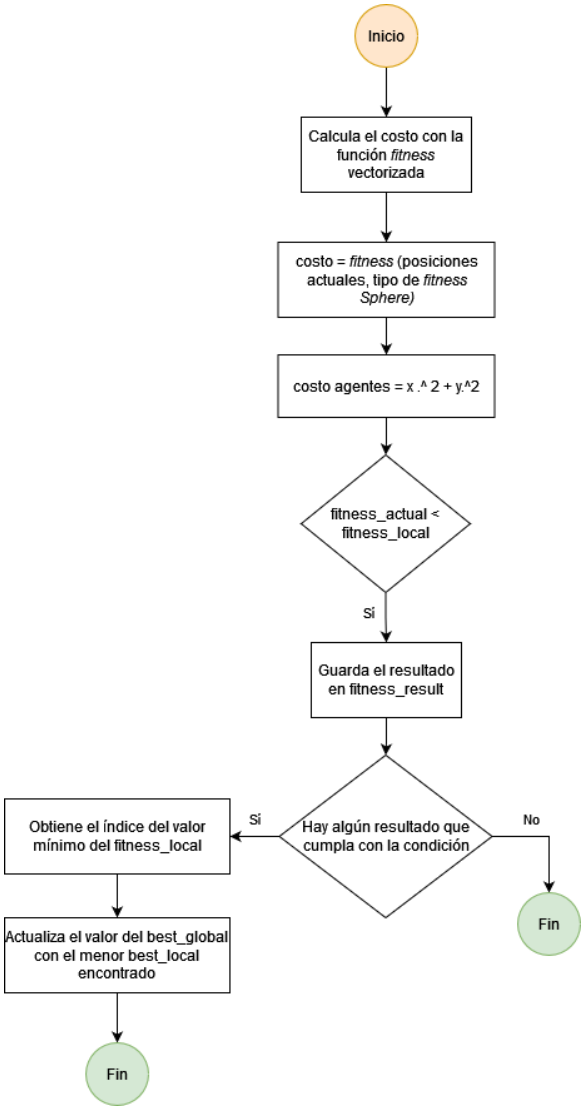



Nota. Elaboración propia.

Sin embargo, al trabajar con un enjambre de robots, esto puede presentar un aumento del tiempo de convergencia de los agentes. Por tal razón, una de las primeras mejoras implementadas fue la vectorización de las funciones de costo.

La vectorización de esta función se realizó mediante operaciones vectoriales, como suma, resta, elevado, multiplicación, división,seno, coseno y raíz cuadrada, eliminando el uso del ciclo "for", como se muestra en la Figura 51.

**Figura 51:** *Diagrama de flujo de la función fitness vectorizada.*




Nota. Elaboración propia.

Es importante mencionar que, en Matlab, se consideran operaciones vectoriales a aquellas que incluyen un punto antes del operador como división, multiplicación y elevado, mientras que para las demás operaciones, como suma, resta, seno, coseno y raíz cuadrada, no es necesario incluir el punto.

Para obtener más información sobre estas funciones, se puede consultar la documentación de Matlab utilizando la palabra *help*, seguido de la operación deseada.

### 9.5.2. Vectorización del cálculo de inercia

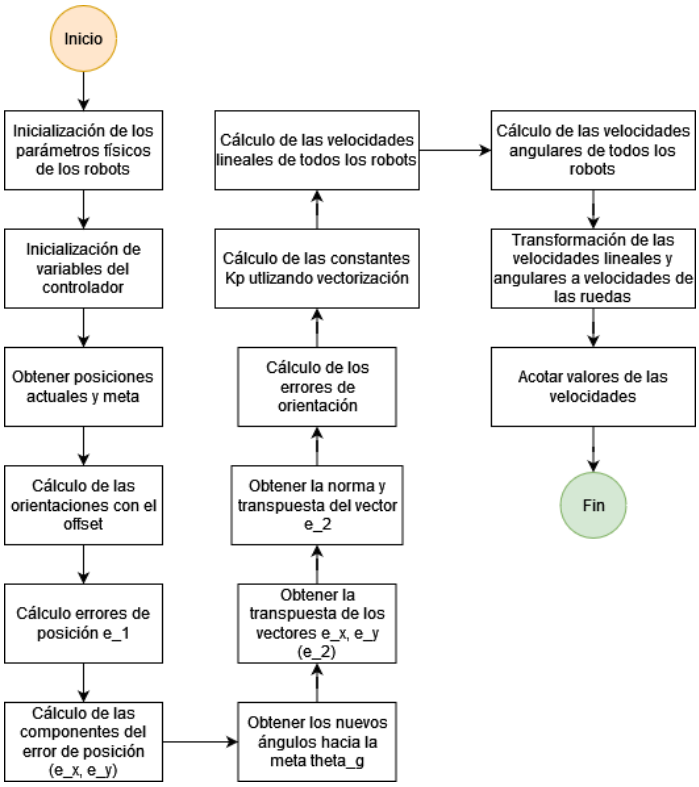
El cálculo del parámetro de la inercia se considera importante para garantizar la convergencia del enjambre. En el algoritmo original, se calcula el valor de la inercia y luego se almacena en una variable para que todos los agentes robóticos.

De igual forma, se emplearon operaciones vectoriales para realizar el cálculo de manera más eficiente y evaluar si hay algún cambio en el tiempo de convergencia de los robots.

### 9.5.3. Vectorización del controlador PID

El controlador PID es el encargado de guiar a los robots hacia la meta deseada, que en este caso es el *global best* encontrado por el enjambre. La implementación del controlador se realizó mediante una función que calcula el controlador de la posición con acercamiento exponencial y el de orientación.

**Figura 52:** *Diagrama de flujo del controlador PID vectorizado.*



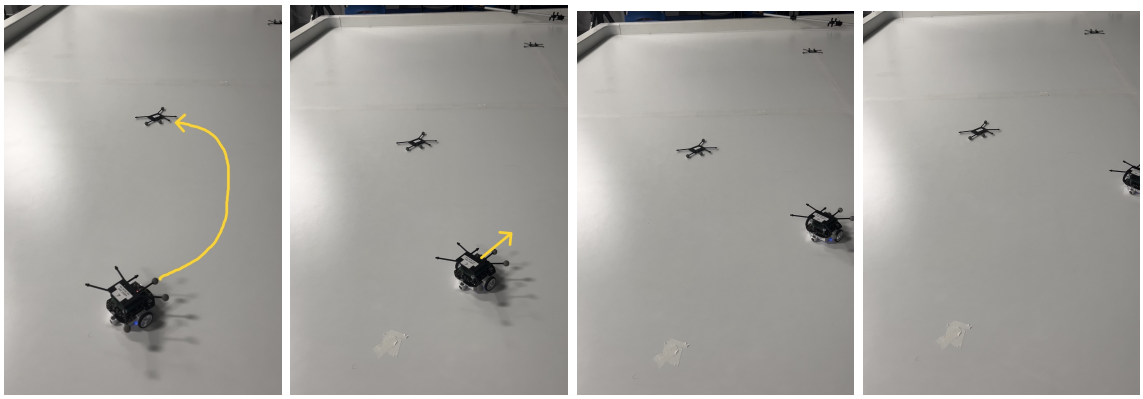

Nota. Elaboración propia.

Se observó que la implementación del controlador se realizaba mediante un ciclo for, que llamaba a la función *PID_ controller_ 1.m*, encargada de calcular las velocidades de las ruedas para cada agente. Como se mencionó anteriormente, debido a que el enjambre está conformado por varios robots, este cálculo puede tomar demasiado tiempo. Por tal razón, se utilizó vectorización para realizar el control y obtener las velocidades de las ruedas para todos los robots sin la necesidad de ciclos for, como se muestra en la Figura 52.

Cabe mencionar que, para esta implementación vectorizada, fue necesario agregar un ciclo while, para garantizar que los robots lleguen a la posición deseada. Esto asegura que el error de posición sea menor a 0.05 y que los robots lleguen más rápido a la meta.

De igual forma, durante la implementación del controlador PID original, se observó que los robots no lograban seguir las curvas tan abiertas y llegar a la meta, como se ilustra en la Figura 53.

**Figura 53:** *Trayectoria del agente Pololu 3pi+ hacia la meta utilizando el controlador PID original.*

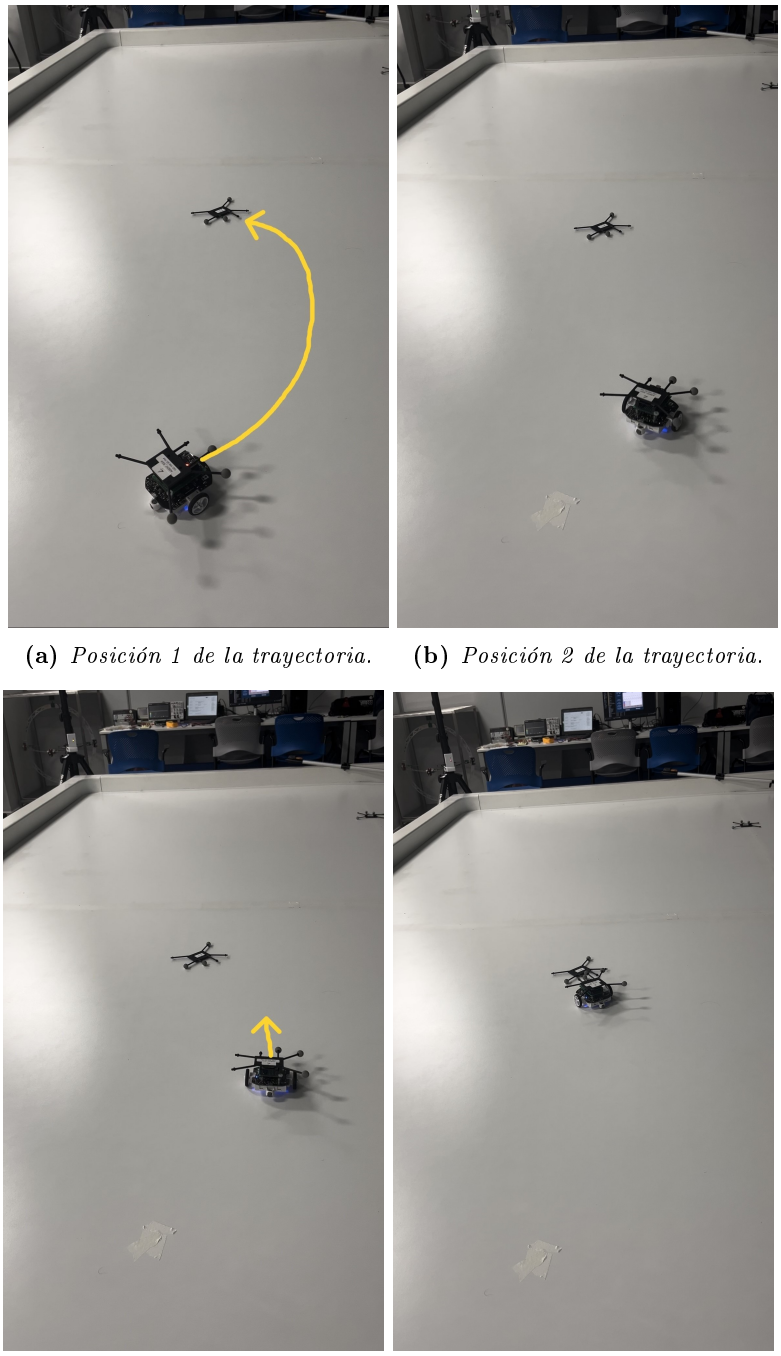


**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.* **(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

En cambio, con el controlador PID vectorizado, el robot sí pudo seguir la curva y llegar a la meta, como se muestra en la Figura 54.

**Figura 54:** *Trayectoria del agente Pololu 3pi+ hacia la meta utilizando controlador PID vectorizado.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

# CAPÍTULO 10

## Implementación y evaluación de mejoras encontradas en el algoritmo MPSO

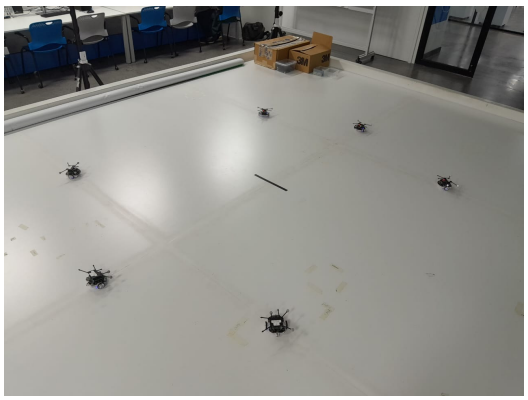
En este capítulo se presentan las implementaciones de las mejoras propuestas para el algoritmo MPSO, tanto en el ecosistema Robotat como en el simulador Webots. Asimismo, se realiza una comparación entre el tiempo de convergencia y las trayectorias generadas por el algoritmo MPSO y el optimizado, con el objetivo de evaluar si las mejoras implementadas incrementan el rendimiento del algoritmo.

## 10.1. Implementación física del algoritmo MPSO vectorizado

En el capítulo anterior, se implementaron técnicas de vectorización tanto en la función *fitness* como en el cálculo de la inercia, con el objetivo de optimizar el tiempo de convergencia del enjambre. Para evaluar si existe mejora se utilizaron 6 agentes Pololu 3pi+ y las funciones de costo Sphere, Booth y Schaffer.

Para cada función de costo, se colocaron a los agentes en una posición inicial, como se ilustra en la Figura 55. Se llevaron a cabo 5 pruebas para cada función, obteniendo los tiempos de convergencia del enjambre y el *global best* encontrado.

**Figura 55:** *Posiciones iniciales de agentes robóticos Pololu 3pi+.*



Nota. Elaboración propia.

### 10.1.1. *Fitness function* vectorizada

Para la función Sphere se presentan los tiempos de convergencia obtenidos, mostrados en el Cuadro 27 y los *global best* encontrados por el enjambre. Se analizaron las trayectorias generadas por el algoritmo vectorizado para evaluar si esta mejora generó trayectorias más optimizadas.

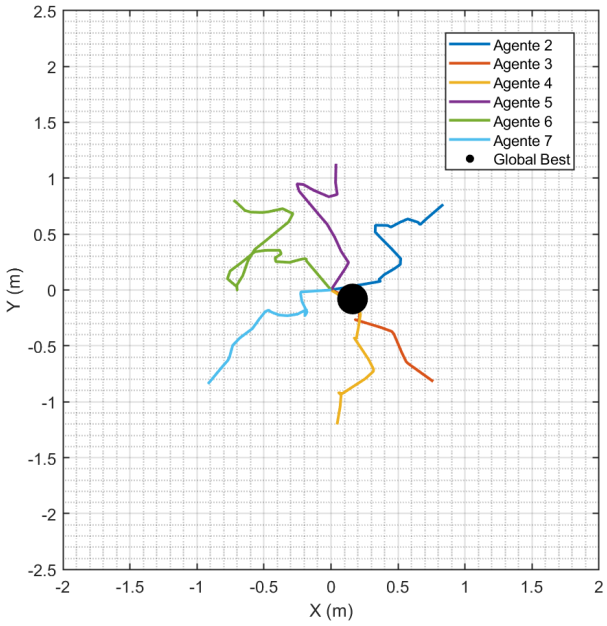
**Cuadro 27:** *Tiempos de convergencia y global best de la función Sphere vectorizada.*

| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Sphere | Original | 42.86 | (0.094, 0.128) |
| 2 | Sphere | Original | 47.73 | (-0.101, 0.0905) |
| 3 | Sphere | Original | 46.79 | (-0.112, -0.042) |
| 4 | Sphere | Original | 43.19 | (-0.120, 0.082) |
| 5 | Sphere | Original | 46.59 | (0.160, -0.080) |

Nota. Elaboración propia.

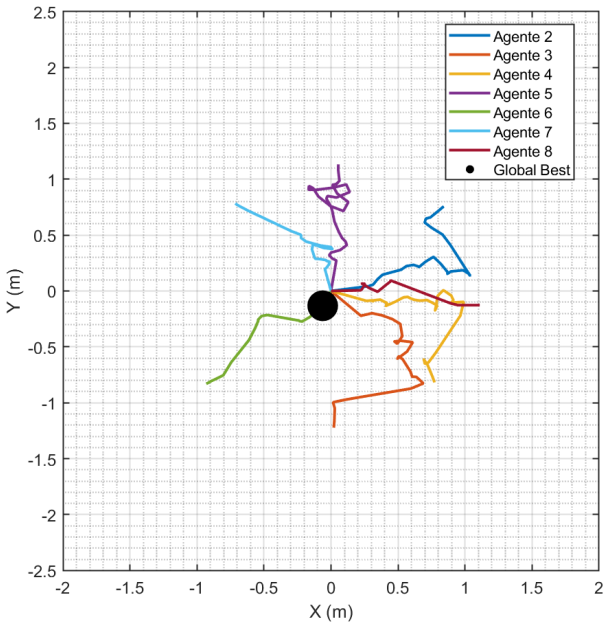
En la Figura 56, se presentan las trayectorias resultantes con la función de costo Sphere vectorizada, mientras que en la Figura 57 se presentan las trayectorias generadas con el algoritmo MPSO original.

**Figura 56:** *Trayectorias generadas con la función Sphere vectorizada en Matlab.*




Nota. Elaboración propia.

**Figura 57:** *Trayectorias generadas con la función Sphere original en Matlab.*




Nota. Elaboración propia.

En la Figura 58, se presentan las trayectorias generadas por los agentes Pololu 3pi+ en el ecosistema Robotat. En esta prueba los agentes llegaron a la meta y el *global best* se encuentra muy cercano al mínimo global de la función Sphere. Las posiciones iniciales de los agentes se presentan en el Cuadro 28.

**Figura 58:** *Trayectoria generada con la función Sphere vectorizada en el ecosistema Robotat.*

**(a)** *Posición 1 de la trayectoria.*

**(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.*

**(d)** *Posición 4 de la trayectoria.*

**(e)** *Posición 5 de la trayectoria.*

**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Cuadro 28:** *Posiciones iniciales de los agentes Pololu 3pi+ con función Sphere vectorizada.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (0.7475, -0.8403) |
| 3 | (0.0355, -1.2218) |
| 4 | (-0.8598, -0.8470) |
| 5 | (0.0442, 1.1414) |
| 6 | (-0.7353, 0.7951) |
| 7 | (0.8565, 0.7843) |

Nota. Elaboración propia.

Para la función Booth, los tiempos de convergencia obtenidos se presentan en el Cuadro 29, al igual que los *global best* encontrados por el enjambre.

**Cuadro 29:** *Tiempos de convergencia y global best de la función Booth vectorizada.*

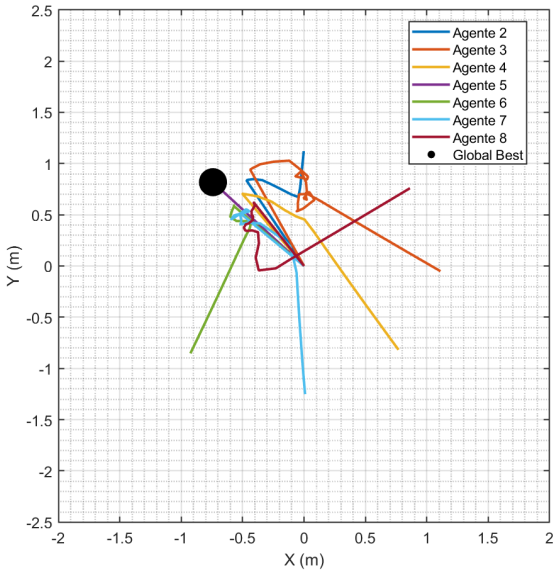
| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Booth | Original | 29.14 | (0.8853, 0.7666) |
| 2 | Booth | Original | 36.40 | (0.8853, 0.7666) |
| 3 | Booth | Original | 26.89 | (-0.7330, 0.7911) |
| 4 | Booth | Original | 39.31 | (-0.7330, 0.7911) |
| 5 | Booth | Original | 36.75 | (-0.7330, 0.7911) |

Nota. Elaboración propia.

En la Figura 60, se presentan las trayectorias resultantes con la función de costo Booth vectorizada mientras que en la Figura 59 se presentan las trayectorias generadas con el algoritmo original.

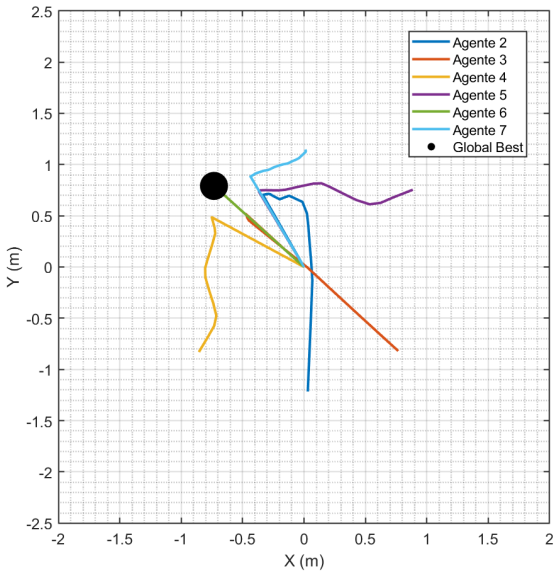
Al comparar las trayectorias generadas con las funciones de costo vectorizadas, se observa una optimización significativa en términos de suavidad. Asimismo, la distancia entre los agentes robóticos y la meta es menor, lo que demuestra una mejora en el desempeño del algoritmo MPSO.

**Figura 59:** *Trayectorias generadas con la función Booth original en Matlab.*




Nota. Elaboración propia.

**Figura 60:** *Trayectorias generadas con la función Booth vectorizada en Matlab.*




Nota. Elaboración propia.

En la Figura 61 se muestran las trayectorias generadas por la función Booth con los agentes robóticos Pololu 3pi+. En este caso, los agentes convergen a un mínimo local bastante cercano al mínimo global. Las posiciones iniciales de los robots se presentan en el Cuadro 30.

**Figura 61:** *Trayectoria generada con la función Booth vectorizada en el ecosistema Robotat.*

**(a)** *Posición 1 de la trayectoria.*

**(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.*

**(d)** *Posición 4 de la trayectoria.*

**(e)** *Posición 5 de la trayectoria.*

**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Cuadro 30:** *Posiciones iniciales de los agentes Pololu 3pi+ con la función Booth vectorizada.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (-0.8991, -0.8461) |
| 3 | (0.7668, -0.8248) |
| 4 | (0.0450, -1.1997) |
| 5 | (0.8853, 0.7666) |
| 6 | (0.0296, 1.1389) |
| 7 | (-0.7248, 0.7793) |

Nota. Elaboración propia.

Para la función Schaffer, se obtuvieron los tiempos de convergencia presentados en el Cuadro 31, así como los *global best* encontrados por el enjambre.

**Cuadro 31:** *Tiempos de convergencia y global best de la función Schaffer vectorizada.*

| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Schaffer | Original | 49.02 | (0.7161, 0.3785) |
| 2 | Schaffer | Original | 43.79 | (0.5788, 0.3396) |
| 3 | Schaffer | Original | 53.08 | (-0.8578, -0.3199) |
| 4 | Schaffer | Original | 59.48 | (0.8109, 0.2384) |
| 5 | Schaffer | Original | 59.06 | (-0.9201, -0.1071) |

Nota. Elaboración propia.

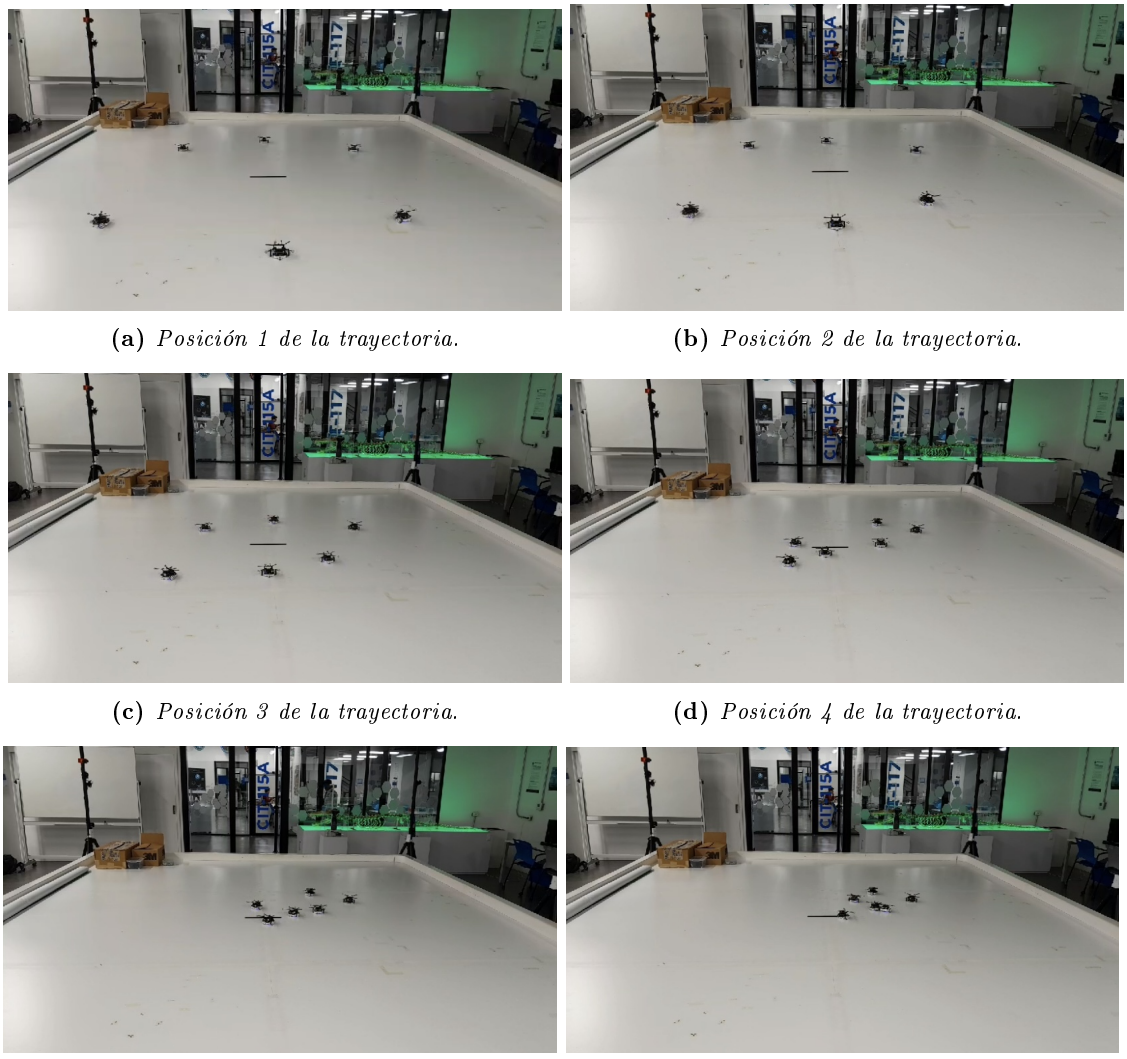
Las trayectorias generadas por los agentes robóticos Pololu 3pi+ se muestran en la Figura 62. En este caso, solo uno de los agentes no llegó al mínimo global. En el Cuadro 32, se presentan los tiempos promedio para cada una de las funciones de costo, se observa que la función de costo con mayor tiempo de convergencia fue la función Schaffer y la función con menor tiempo fue Booth.

**Cuadro 32:** *Tiempos promedio de convergencia de las funciones de costo vectorizadas.*

| Prueba | Tiempo promedio (s) |
|---|---|
| Sphere | 45.432 |
| Booth | 33.698 |
| Schaffer | 52.886 |

Nota. Elaboración propia.

**Figura 62:** *Trayectoria generada con la función Schaffer vectorizada en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.*



**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

### 10.1.2. Parámetro de inercia y *fitness function* vectorizada

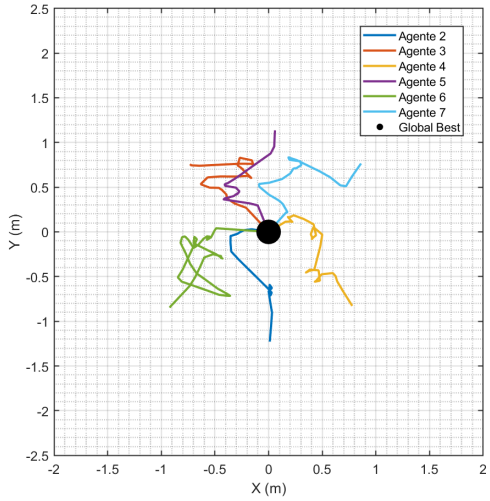
Para la función Sphere se obtuvieron los tiempos de convergencia presentados en el Cuadro 33, así como los *global best* encontrados por el enjambre. Las trayectorias obtenidas con esta función se muestran en la Figura 63.

**Cuadro 33:** *Tiempos de convergencia y global best de la función Sphere con el parámetro de inercia vectorizado.*

| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Sphere | Original | 34.28 | (-0.1071, -0.0808) |
| 2 | Sphere | Original | 54.59 | (0.1723, 0.0666) |
| 3 | Sphere | Original | 46.79 | (-0.0656, 0.0948) |
| 4 | Sphere | Original | 43.86 | (0.0063, 0.1831) |
| 5 | Sphere | Original | 37.27 | (-0.1327, 0.0748) |

Nota. Elaboración propia.

**Figura 63:** *Trayectorias generadas con la función Sphere con el parámetro de inercia vectorizado en Matlab.*

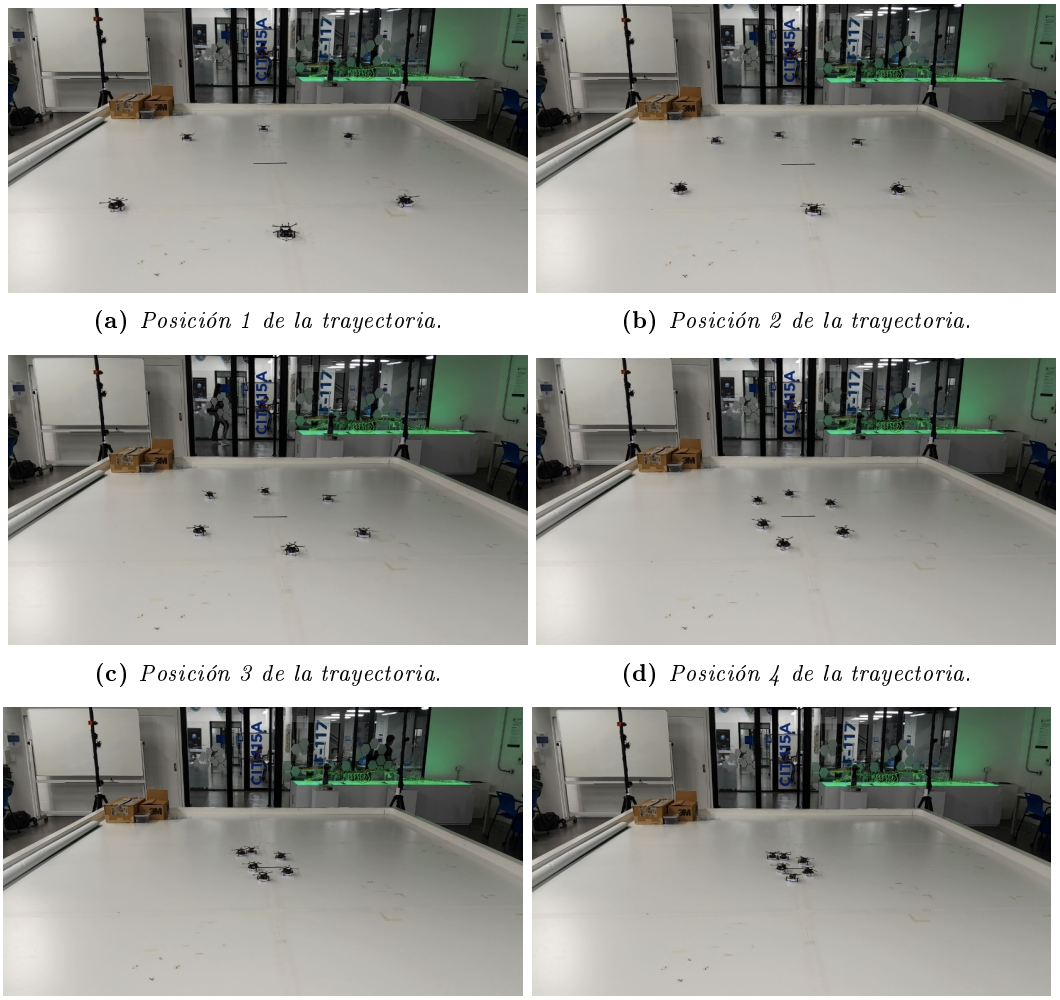



Nota. Elaboración propia.

Las trayectorias obtenidas por los agentes robóticos Pololu 3pi+ se muestran en la Figura 64. En este caso, los agentes llegaron al mínimo global, el origen del Robotat.

Para la función Booth se obtuvieron los tiempos de convergencia presentados en el Cuadro 34, así como los *global best* encontrados por el enjambre.

**Figura 64:** *Trayectoria generada con la función Sphere con el parámetro de inercia vectorizado en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

**(e)** *Posición 5 de la trayectoria.* **(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Cuadro 34:** *Tiempos de convergencia y global best de función Booth con el parámetro de inercia vectorizado.*

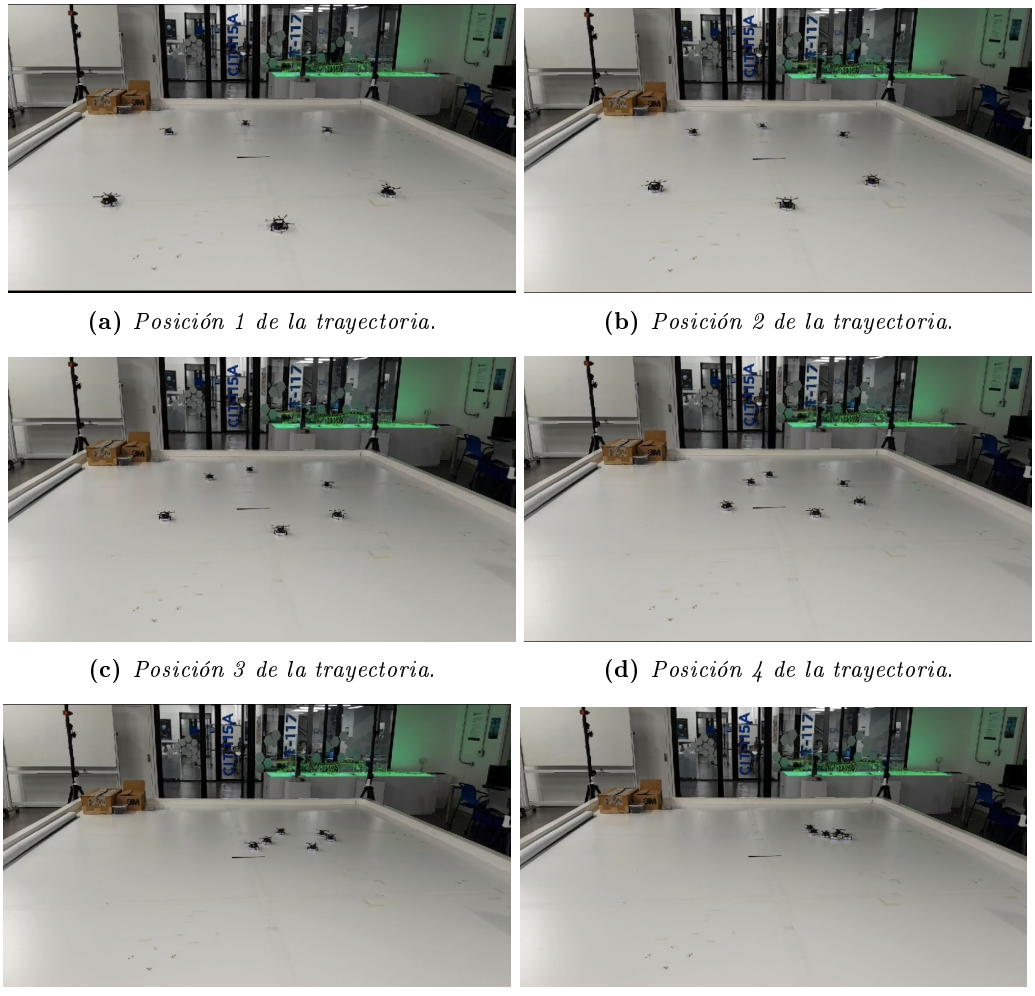
| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Booth | Original | 33.86 | (0.8795, 0.7741) |
| 2 | Booth | Original | 32.63 | (-0.7581, 0.7664) |
| 3 | Booth | Original | 33.84 | (0.8409, 0.7695) |
| 4 | Booth | Original | 38.66 | (-0.7377, 0.7692) |
| 5 | Booth | Original | 25.97 | (0.8385, 0.7611) |

Nota. Elaboración propia.

Las trayectorias obtenidas por los agentes robóticos Pololu 3pi+ se muestran en la Figura 65. En este caso los agentes llegaron al mínimo global encontrado por el enjambre.

Las trayectorias obtenidas para esta función se muestran en la Figura 66.

**Figura 65:** *Trayectoria generada con la función Booth con el parámetro de inercia vectorizado en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.*



**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*



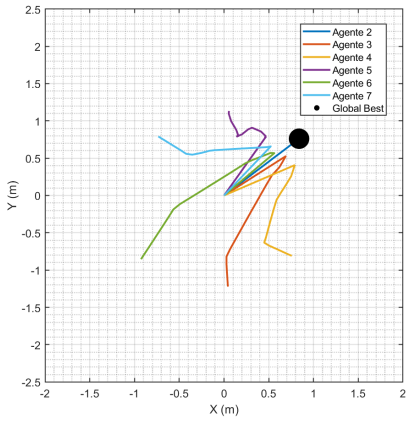
**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Figura 66:** *Trayectorias generadas con la función Booth con el parámetro de inercia vectorizado en Matlab.*




Nota. Elaboración propia.

Para la función Schaffer se obtuvieron los tiempos de convergencia presentados en el Cuadro 35, así como los *global best* encontrados por el enjambre.

**Cuadro 35:** *Tiempos de convergencia y global best de función Schaffer con el parámetro de inercia vectorizado.*

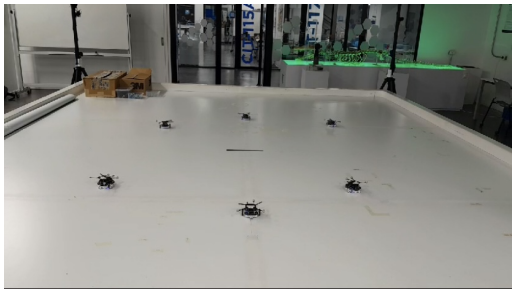
| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Schaffer | Original | 61.02 | (1.3114, 0.6147) |
| 2 | Schaffer | Original | 32.34 | (-0.6713, 0.2750) |
| 3 | Schaffer | Original | 61.01 | (1.7214, 0.8424) |
| 4 | Schaffer | Original | 48.96 | (-0.7271, -0.2186) |
| 5 | Schaffer | Original | 61.04 | (-0.9060, -0.4372) |

Nota. Elaboración propia.

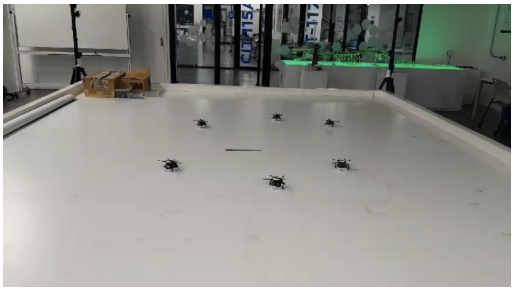
Las trayectorias generadas por los agentes robóticos Pololu 3pi+ se presentan en la Figura 67. En este caso, se puede observar que los agentes llegaron a un mínimo global. Las trayectorias obtenidas por esta función se muestra en la Figura 68.

**Figura 67:** *Trayectoria generada con la función Schaffer con el parámetro de inercia vectorizado en el ecosistema Robotat.*

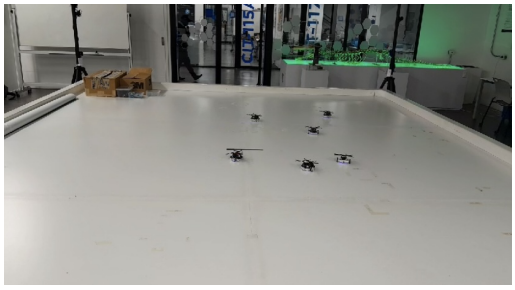
**(a)** *Posición 1 de la trayectoria.*



**(b)** *Posición 2 de la trayectoria.*

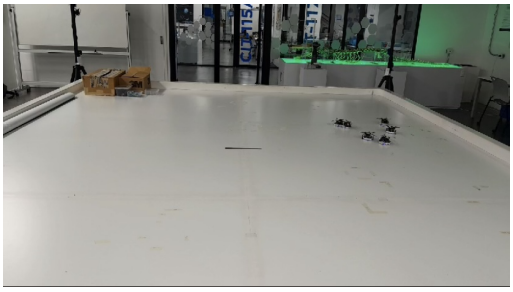


**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*

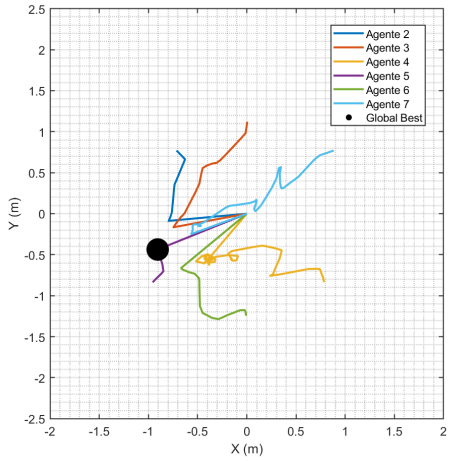
**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Figura 68:** *Trayectorias generadas con la función Schaffer con el parámetro de inercia vectorizado en Matlab.*




Nota. Elaboración propia.

Al comparar las trayectorias generadas con el parámetro de inercia vectorizado, se observa que las trayectorias son mejores a las generadas con el algoritmo original. Sin embargo, son menos suaves en comparación con las trayectorias generadas con las funciones de costo vectorizadas. La distancia entre los robots y la meta siguen siendo menores.

Los tiempos de convergencia promedio se presentan en el Cuadro 36. Se puede observar que la función Schaffer fue la que tuvo mayor tiempo de convergencia y la función Booth fue la que tuvo menor tiempo.

**Cuadro 36:** *Tiempos promedio de convergencia con el parámetro de inercia vectorizado.*

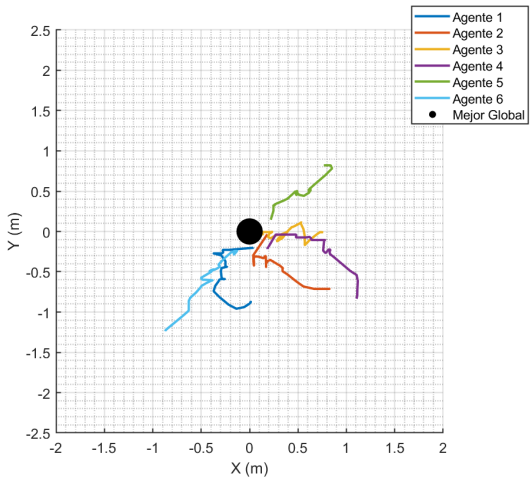
| **Prueba** | **Tiempo promedio (s)** |
|---|---|
| Sphere | 43.358 |
| Booth | 32.992 |
| Schaffer | 52.874 |

Nota. Elaboración propia.

### 10.1.3. Controlador PID vectorizado

La primera función que se evalúo fue Sphere. En el Cuadro 37, se presentan los tiempos de convergencia, así como los *global best* encontrados por el enjambre. En la Figura 69, se muestran las trayectorias generadas con esta función y el controlador PID vectorizado.

**Figura 69:** *Trayectorias generadas con la función Sphere con el controlador PID vectorizado en Matlab.*




Nota. Elaboración propia.

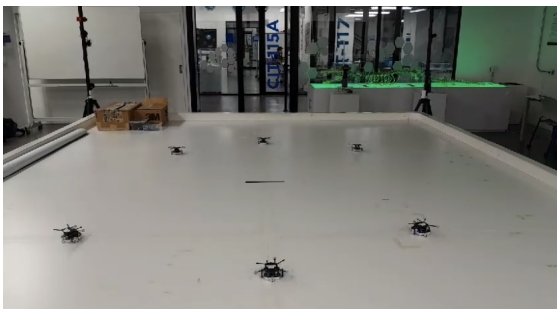
**Cuadro 37:** *Tiempos de convergencia y global best de función Sphere con el controlador PID vectorizado.*

| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Sphere | Vectorizado | 57.82 | (-0.0841, -0.1690) |
| 2 | Sphere | Vectorizado | 49.44 | (0.0007, 0.1959) |
| 3 | Sphere | Vectorizado | 59.07 | (-0.0356, 0.1451) |
| 4 | Sphere | Vectorizado | 56.23 | (0.0803, 0.1430) |
| 5 | Sphere | Vectorizado | 61.01 | (0.1030, -0.1296) |

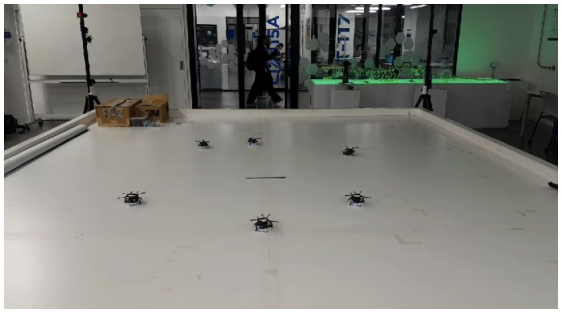
Nota. Elaboración propia.

Las trayectorias obtenidas por los agentes robóticos Pololu 3pi+ se muestran en la Figura 70. En este caso, todos los agentes robóticos llegaron a la meta. Las posiciones iniciales de los agentes se presentna en el Cuadro 38.

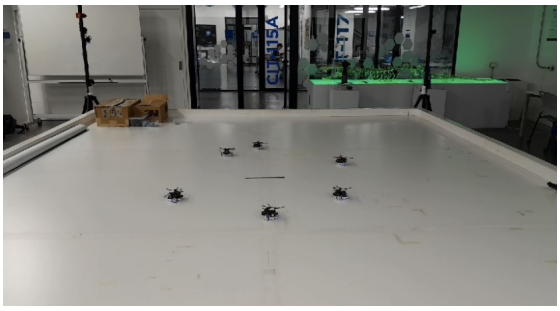
**Figura 70:** *Trayectoria generada con la función Sphere con el controlador PID vectorizado en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.*

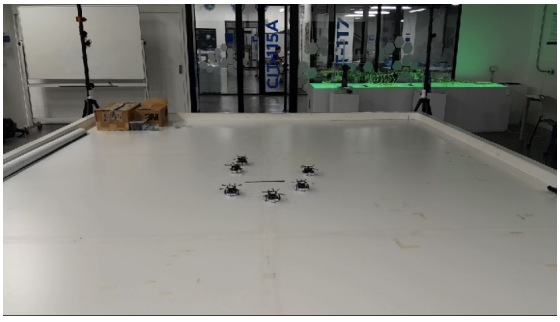


**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*

**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*

**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

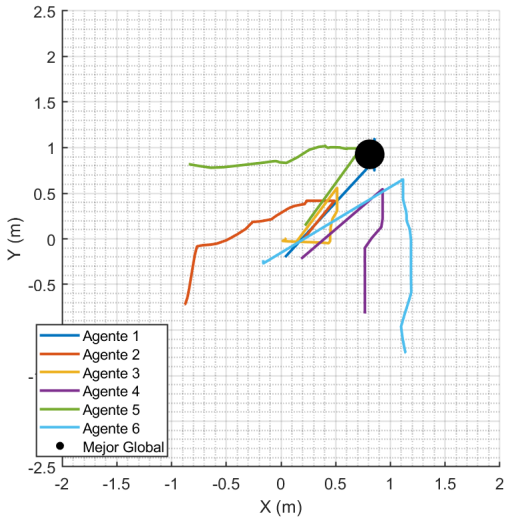
**Cuadro 38:** *Posiciones iniciales de los agentes Pololu 3pi+ con la función Sphere y el controlador PID vectorizado.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (-0.6981, 0.7911) |
| 3 | (-0.9090, -0.8335) |
| 4 | (0.8201, 0.7719) |
| 5 | (0.0574, 1.1683) |
| 6 | (0.7666, -0.8782) |
| 7 | (0.0111, -1.2516) |

Nota. Elaboración propia.

La segunda función evaluada fue Booth. En la Figura 71 se muestran las trayectorias generadas con esta función. En el Cuadro 39, se presentan los tiempos de convergencia obtenidos, así como los *global best* encontrados por el enjambre.

**Figura 71:** *Trayectorias generadas con la función Booth y controlador PID vectorizado en Matlab.*




Nota. Elaboración propia.

**Cuadro 39:** *Tiempos de convergencia y global best de función Booth con el controlador PID vectorizado.*

| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Booth | Vectorizado | 32.00 | (-0.6905, 0.8080) |
| 2 | Booth | Vectorizado | 47.59 | (-0.7249, 0.8287) |
| 3 | Booth | Vectorizado | 33.75 | (0.8433, 0.7654) |
| 4 | Booth | Vectorizado | 48.22 | (-0.7197, 0.8182) |
| 5 | Booth | Vectorizado | 61.02 | (0.8110, 0.9278) |

Nota. Elaboración propia.

Las posiciones iniciales se presentan en el Cuadro 40. Las trayectorias generadas por los agentes Pololu 3pi+ se muestran en la Figura 72. Para esta función, los agentes robóticos llegaron al mínimo global encontrado por el enjambre.

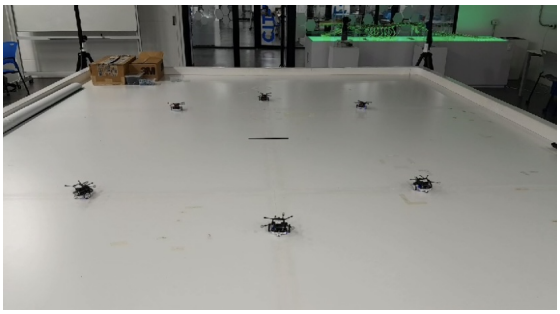
**Cuadro 40:** *Posiciones iniciales de los agentes Pololu 3pi+ con la función Booth y el controlador PID vectorizado.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (-0.6981, 0.7911) |
| 3 | (-0.9090, -0.8335) |
| 4 | (0.8201, 0.7719) |
| 5 | (0.0574, 1.1683) |
| 6 | (0.7666, -0.8782) |
| 7 | (0.0111, -1.2516) |

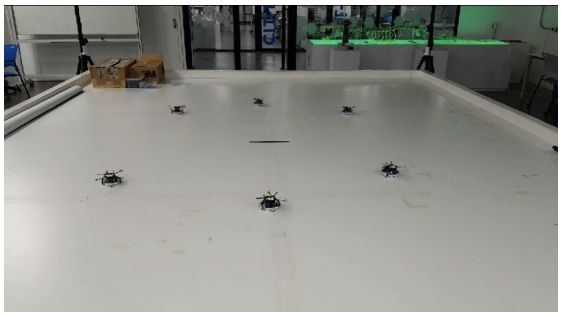
Nota. Elaboración propia.

La tercera función evaluada fue Schaffer. En el Cuadro 41, se presentan los tiempos de convergencia obtenidos, así como los *global best* encontrados por el enjambre. En la Figura 73 se muestran las trayectorias generadas por esta función.

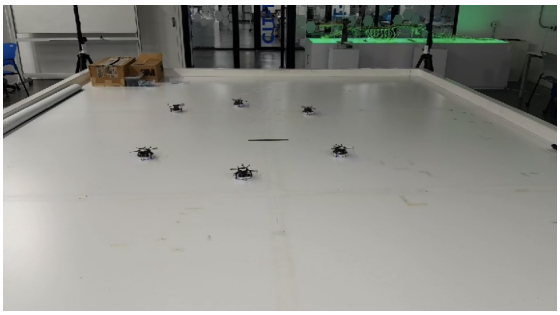
**Figura 72:** *Trayectoria generada con la función Booth con el controlador PID vectorizado en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.*

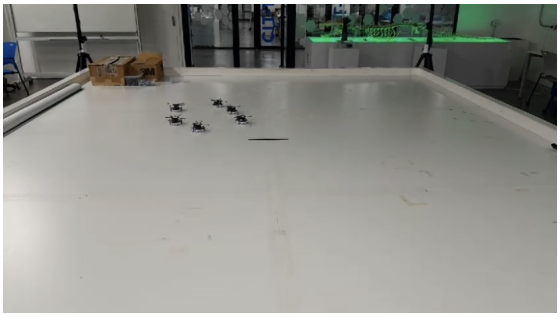


**(b)** *Posición 2 de la trayectoria.*

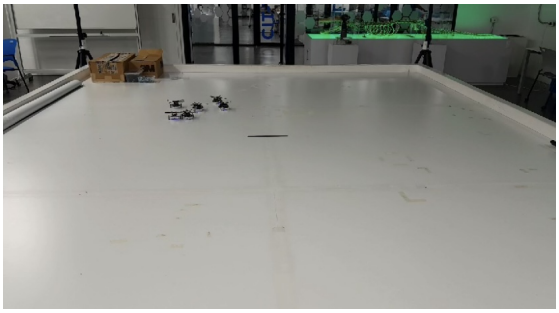


**(c)** *Posición 3 de la trayectoria.*

**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*



**(f)** *Posición 6 de la trayectoria.*

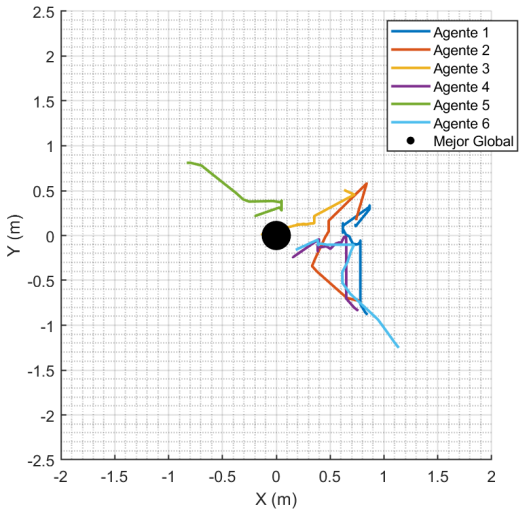
Nota. Elaboración propia.

**Cuadro 41:** *Tiempos de convergencia y global best de función Schaffer con el controlador PID vectorizado.*

| No. | Prueba | Controlador | Tiempo (s) | Global best (m) |
|---|---|---|---|---|
| 1 | Schaffer | Vectorizado | 52.19 | (0.7449, 0.4610) |
| 2 | Schaffer | Vectorizado | 59.43 | (0.7366, 0.1546) |
| 3 | Schaffer | Vectorizado | 56.45 | (0.8403, 0.0310) |
| 4 | Schaffer | Vectorizado | 50.16 | (0.4867, 0.0484) |
| 5 | Schaffer | Vectorizado | 56.40 | (0.7542, 0.4589) |

Nota. Elaboración propia.

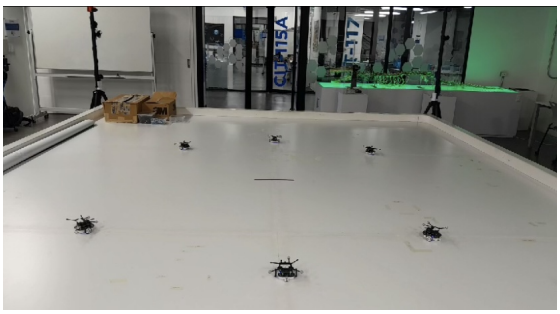
**Figura 73:** *Trayectorias generadas con la función Schaffer con el controlador PID vectorizado en Matlab.*



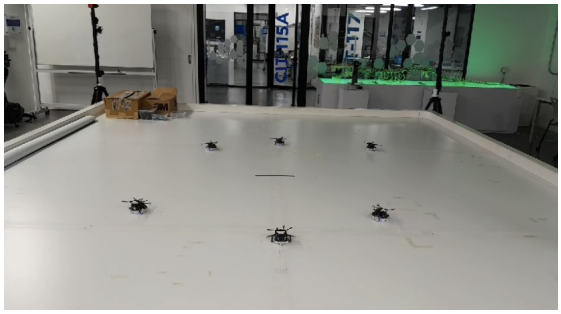

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos Pololu 3pi + se presentan en la Figura 74. Para esta función, los agentes robóticos llegaron al mínimo global encontrado por el enjambre. Las posiciones iniciales se presentan en el Cuadro 42.

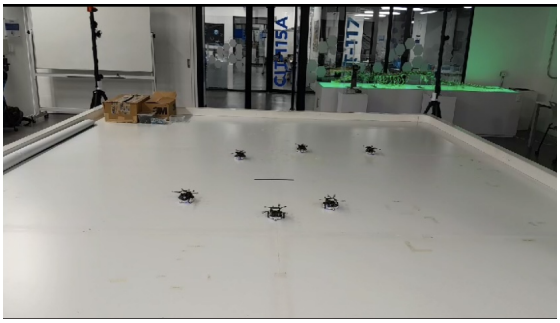
**Figura 74:** *Trayectoria generada con la función Schaffer con el controlador PID vectorizado en el ecosistema Robotat.*



**(a)** *Posición 1 de la trayectoria.*

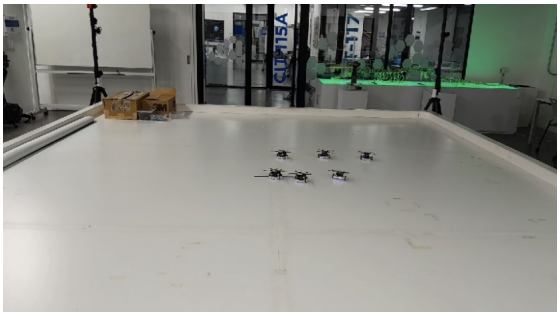


**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*

**(d)** *Posición 4 de la trayectoria.*



**(e)** *Posición 5 de la trayectoria.*

**(f)** *Posición 6 de la trayectoria.*

Nota. Elaboración propia.

**Cuadro 42:** *Posiciones iniciales de los agentes Pololu 3pi+ con la función Schaffer y controlador PID vectorizado.*

| Robot | Posición inicial (m) |
|---|---|
| 2 | (0.8622, 0.7665) |
| 3 | (-0.9016, -0.8426) |
| 4 | (0.7731, -0.8289) |
| 5 | (0.0809, 1.1339) |
| 6 | (-0.7343, 0.8030) |
| 7 | (0.0173, -1.2537) |

Nota. Elaboración propia.

Al comparar las trayectorias generadas con el controlador PID vectorizado, se observa que estas no son mejores en comparación con las trayectorias originales, debido a que no son trayectorias suaves.

En el Cuadro 43, se presentan los tiempos de convergencia promedio para cada una de las funciones de costo con el controlador PID vectorizado. La función con mayor tiempo fue la Sphere y con menor tiempo fue la función Booth.

**Cuadro 43:** *Tiempos de convergencia promedio con el controlador PID vectorizado.*

| Prueba | Tiempo promedio (s) |
|---|---|
| Sphere | 56.714 |
| Booth | 44.516 |
| Schaffer | 54.926 |

Nota. Elaboración propia.

## 10.2. Implementación del algoritmo MPSO migrado a python en webots

En el capítulo anterior, se migró el algoritmo MPSO a python utilizando Webots, con el objetivo de evaluar si existe mejora en el tiempo de convergencia del enjambre. Para esto se utilizaron 6 agentes robóticos Pololu 3pi+ simulados y las funciones de costo Sphere, Booth y Schaffer.

Las posiciones iniciales de los agentes robóticos utilizadas en estas pruebas se presentan en el Cuadro 44.

**Cuadro 44:** *Posiciones iniciales de los agentes Pololu 3pi+ en Webots.*

| Robot | Posiciones iniciales |
|---|---|
| 1 | (1.4137, 0.8268) |
| 2 | (-1.3633, 0.8456) |
| 3 | (0.0209, -1.6523) |
| 4 | (-0.0073, 1.8890) |
| 5 | (-1.2399, -0.8555) |
| 6 | (1.3116, -0.8107) |

Nota. Elaboración propia.

### 10.2.1. Pruebas con el algoritmo MPSO original en webots

La primera función evaluada fue Sphere. En el Cuadro 45, se presenta el tiempo de convergencia obtenido, así como el *global best* encontrado por el enjambre.

**Cuadro 45:** *Tiempo de convergencia y global best de función Sphere con el algoritmo original en Webots.*

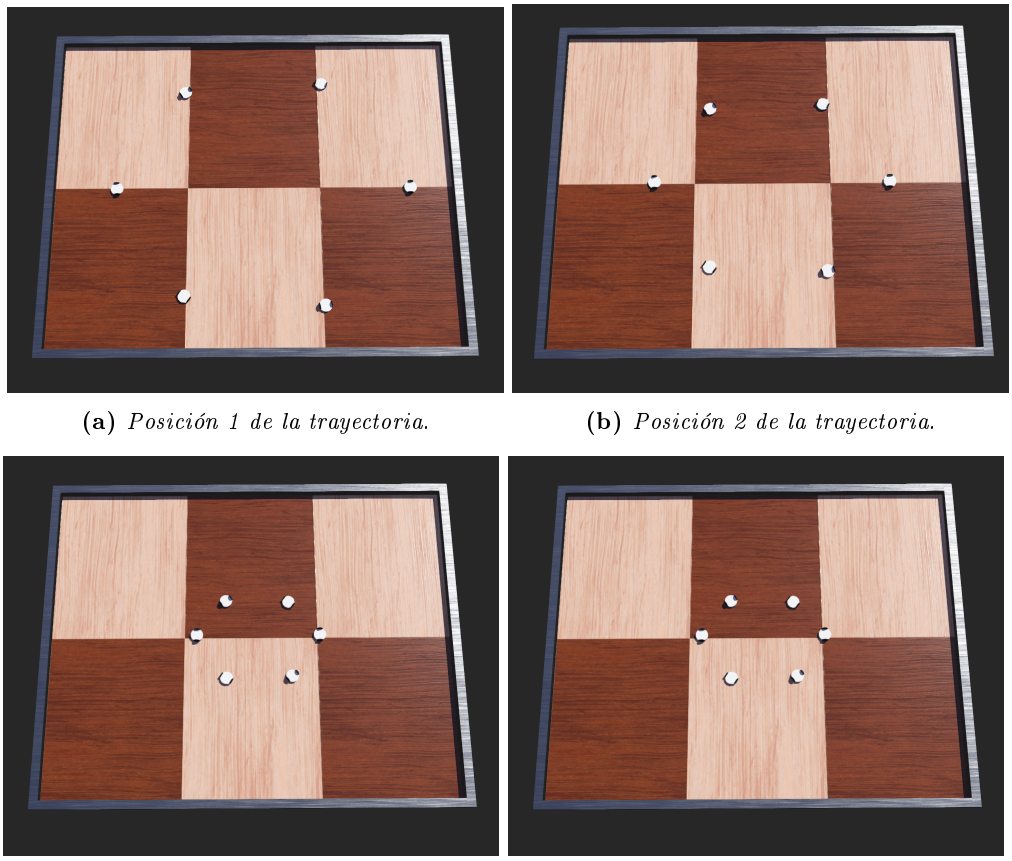
| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Sphere | 17.31 | (0,0) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 75. Para esta función, los agentes llegaron al mínimo global de la función.

La segunda función evaluada fue Booth. En el Cuadro 46, se presenta el tiempo de convergencia obtenido y el *global best* encontrado por el enjambre.

**Figura 75:** *Trayectoria generada con la función Sphere con el algoritmo original en Webots.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

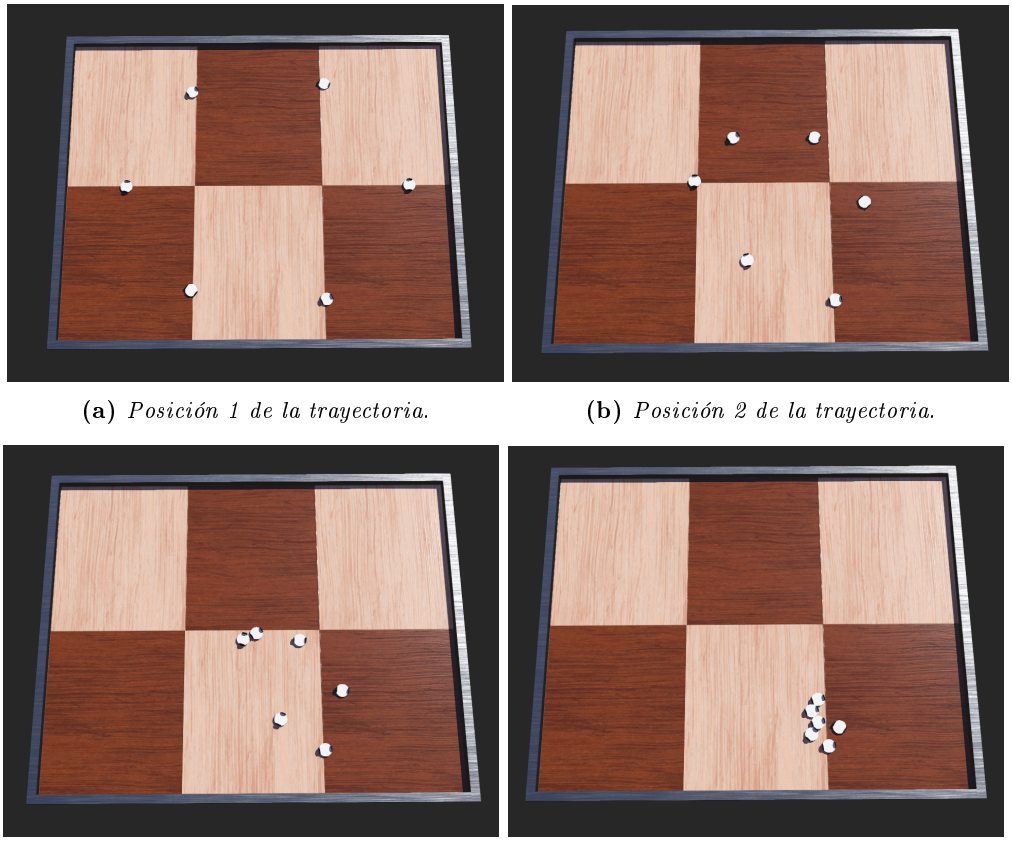
**Cuadro 46:** *Tiempo de convergencia y global best de función Booth con el algoritmo original en Webots.*

| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Booth | 26.33 | (1.5004, 1.5800) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 76. Para esta función, los agentes llegaron al mínimo global encontrado por el enjambre.

**Figura 76:** *Trayectoria generada con la función Booth con el algoritmo original en Webots.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

La tercera función evaluada fue Schaffer. En el Cuadro 47, se presenta el tiempo de convergencia obtenido y el *global best* encontrado por el enjambre.

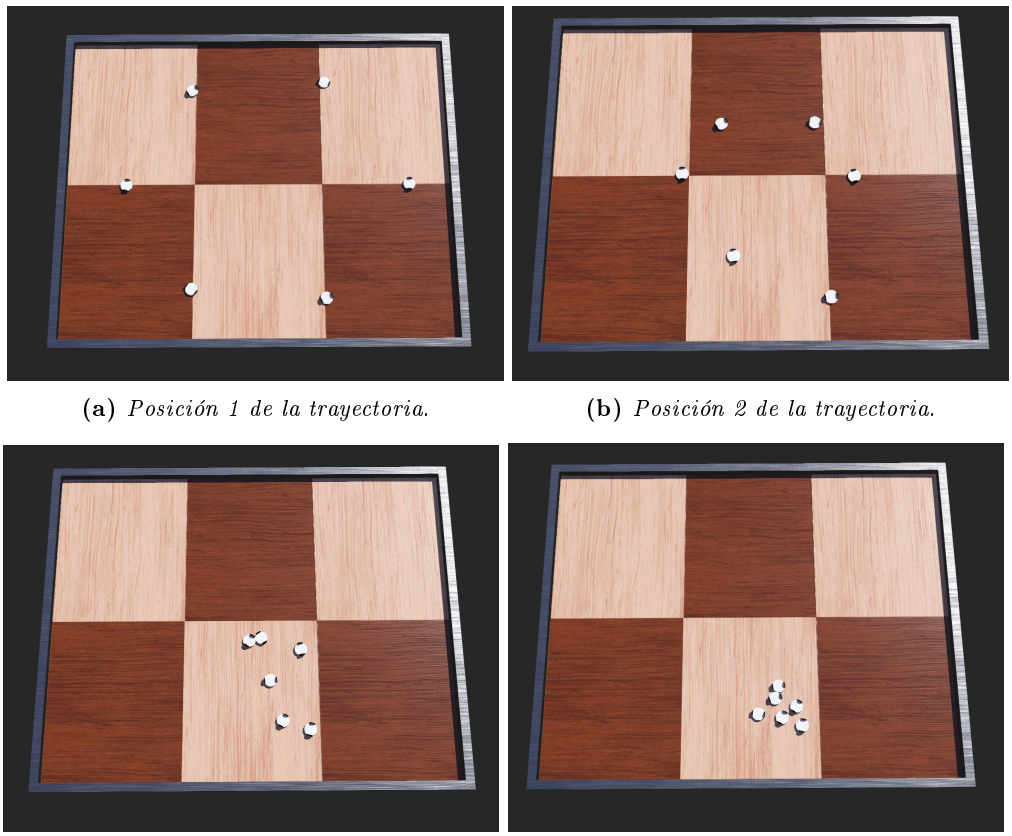
**Cuadro 47:** *Tiempo de convergencia y global best de función Schaffer con el algoritmo original en Webots.*

| **No.** | **Prueba** | **Tiempo (s)** | **Global best (m)** |
|---|---|---|---|
| 1 | Schaffer | 24.60 | (1.5808, 0.0849) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 77. Para esta función, agentes llegaron al mínimo global encontrado por el enjambre.

**Figura 77:** *Trayectoria generada con la función Schaffer con el algoritmo original en Webots.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

### 10.2.2. Pruebas con el algoritmo MPSO migrado a python

La primera función evaluada fue Sphere. En el Cuadro 48, se presenta el tiempo de convergencia y el *global best* encontrado por el enjambre.

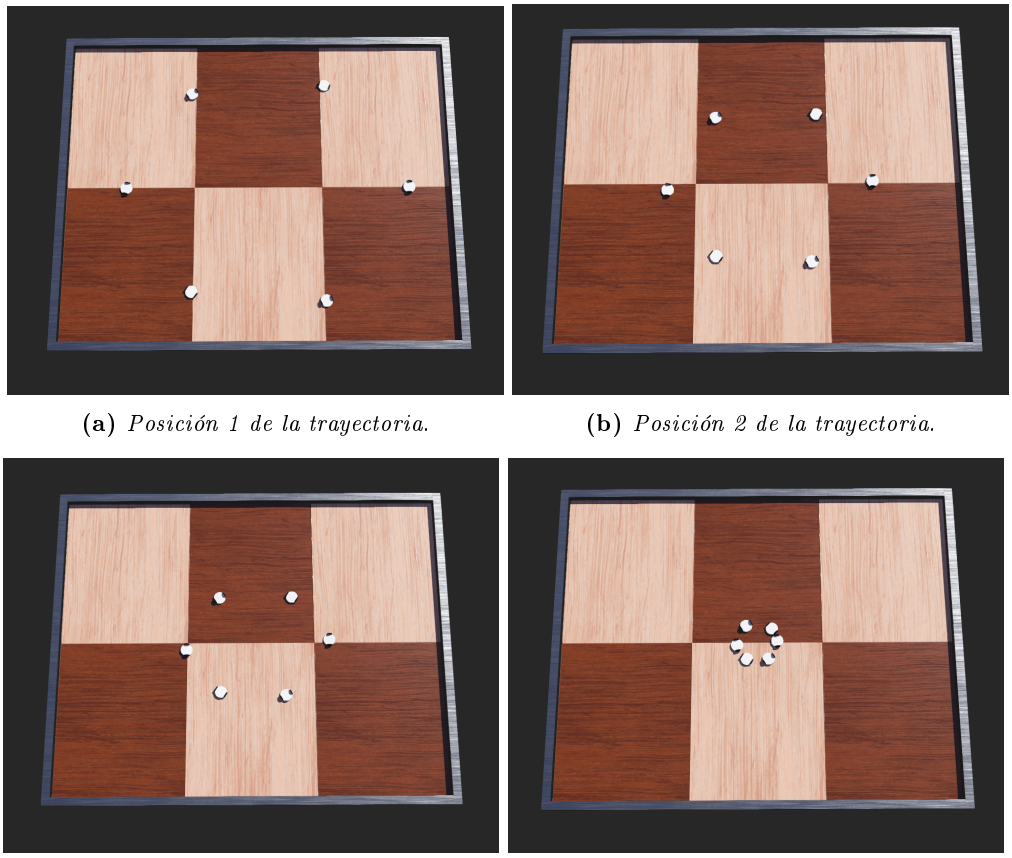
**Cuadro 48:** *Tiempo de convergencia y global best de función Sphere con el algoritmo migrado a Python.*

| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Sphere | 11.77 | (0,0) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 78. Para esta función, los agentes llegaron al mínimo global de la función.

**Figura 78:** *Trayectoria generada con la función Sphere con el algoritmo migrado a Python.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

La segunda función evaluada fue Booth. En el Cuadro 49, se presenta el tiempo de convergencia y el *global best* encontrado por el enjambre.

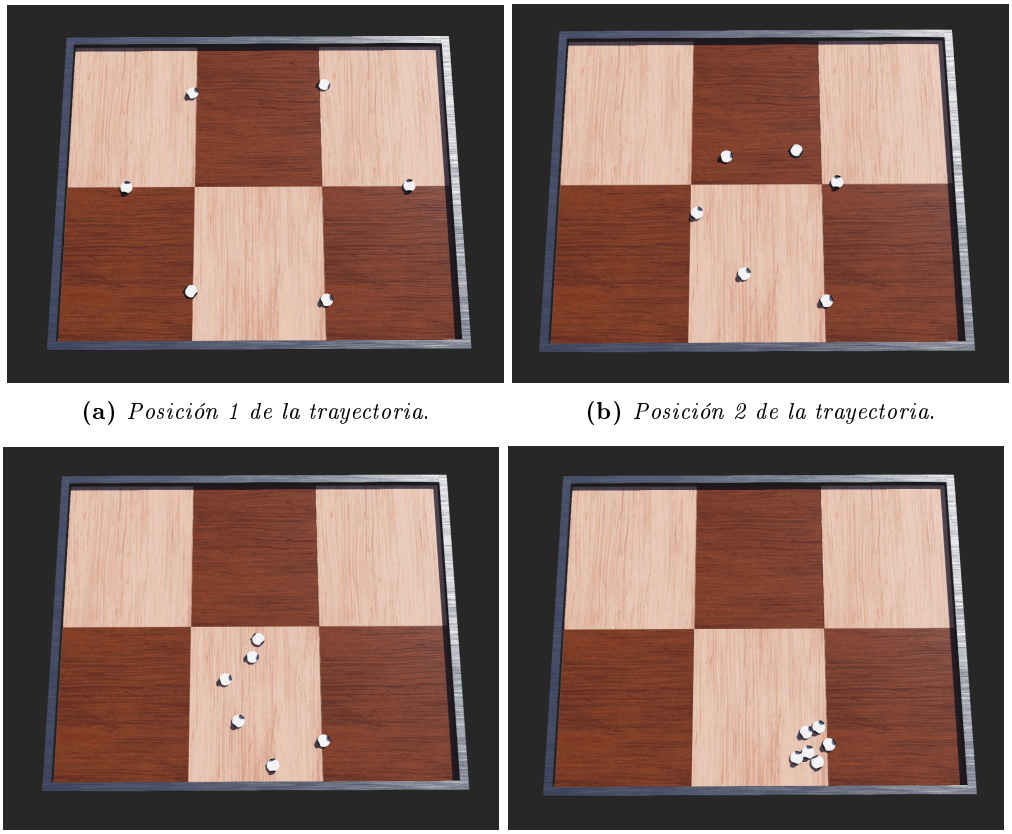
**Cuadro 49:** *Tiempo de convergencia y global best de la función Booth con el algoritmo migrado a Python.*

| **No.** | **Prueba** | **Tiempo (s)** | **Global best (m)** |
|---|---|---|---|
| 1 | Booth | 18.17 | (1.4138, 0.8267) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 79. Para esta función, los agentes llegaron al mínimo global de la función.

**Figura 79:** *Trayectoria generada con la función Booth con el algoritmo migrado a Python.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

La tercera función evaluada fue Schaffer. En el Cuadro 50, se presenta el tiempo de convergencia y el *global best* encontrado por el enjambre.

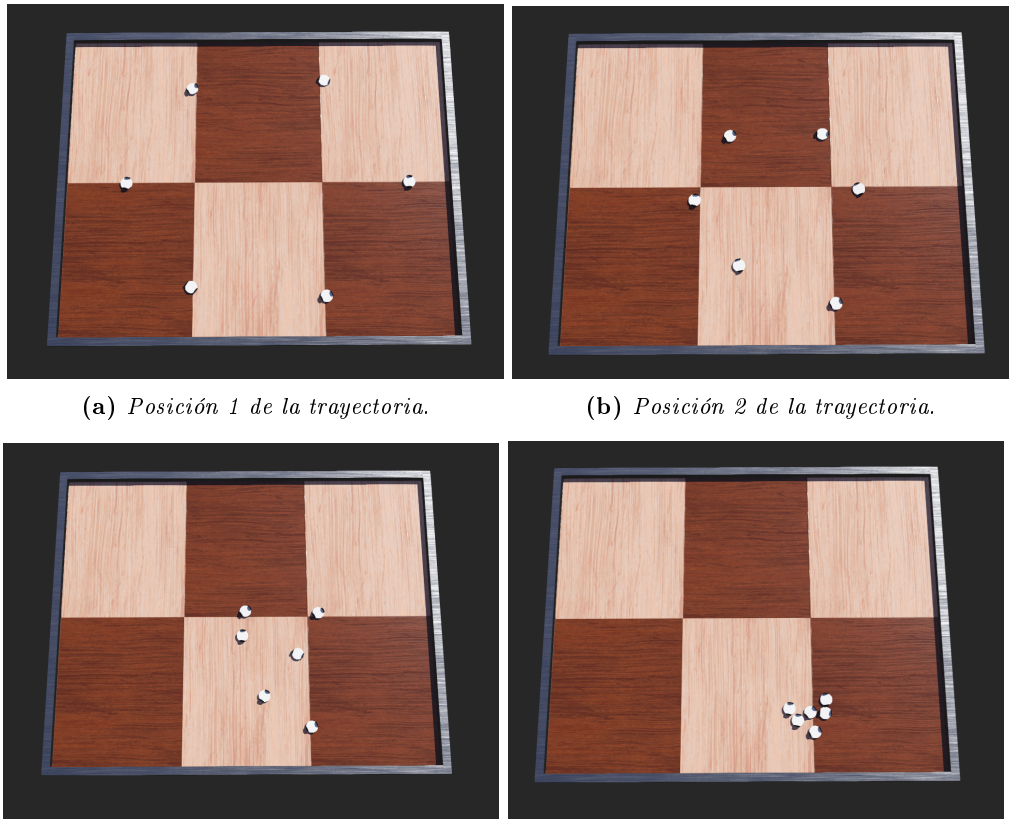
**Cuadro 50:** *Tiempo de convergencia y global best de función Schaffer con el algoritmo migrado a Python.*

| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Schaffer | 21.3 | (1.2054, 0.3716) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 80. Para esta función, los agentes llegaron al mínimo global encontrado por el enjambre.

**Figura 80:** *Trayectoria generada con la función Schaffer con el algoritmo migrado a Python.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

### 10.2.3. Pruebas con el algoritmo MPSO paralelizado en python

La primera función evaluada fue Sphere. En el Cuadro 51, se presenta el tiempo de convergencia y el *global best* encontrado por el enjambre.

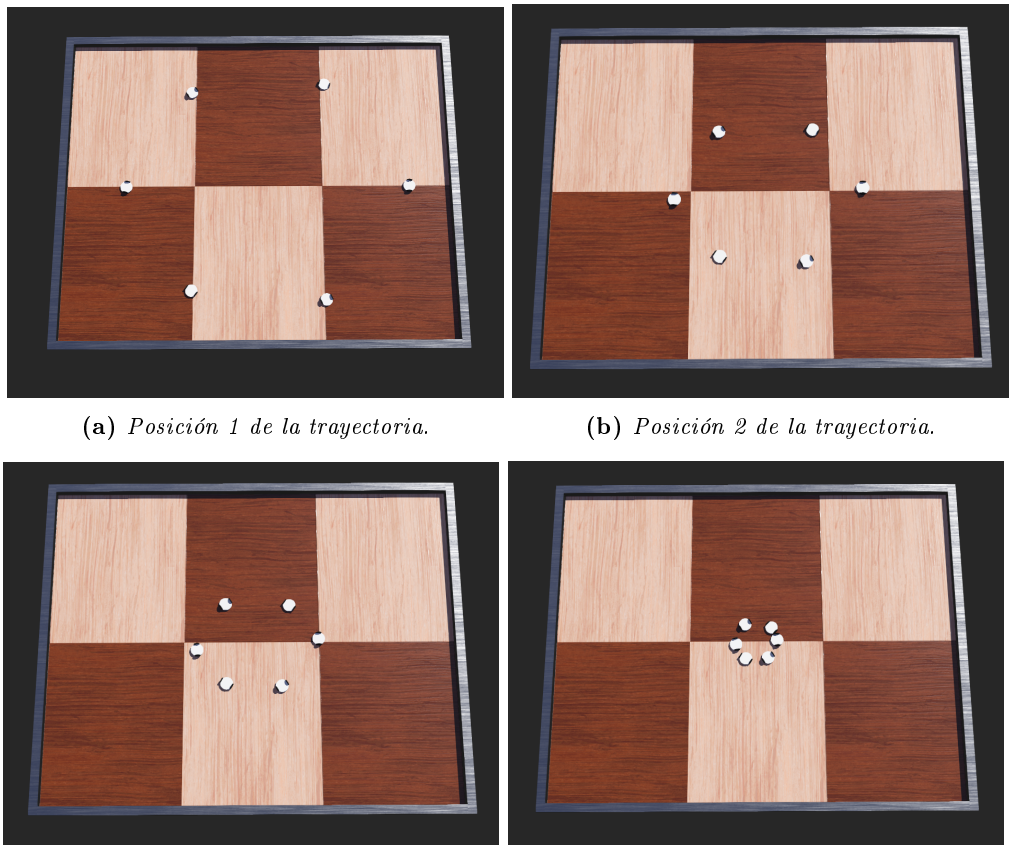
**Cuadro 51:** *Tiempo de convergencia y global best de función Sphere con el algoritmo paralelizado en Python en Webots.*

| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Sphere | 11.55 | (0,0) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 81. Para esta función, los agentes llegaron al mínimo global encontrado por el enjambre.

**Figura 81:** *Trayectoria generada con la función Sphere con el algoritmo paralelizado en Python.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

La segunda función evaluada fue Booth. En el Cuadro 52, se presenta el tiempo de convergencia y el *global best* encontrado por el enjambre.

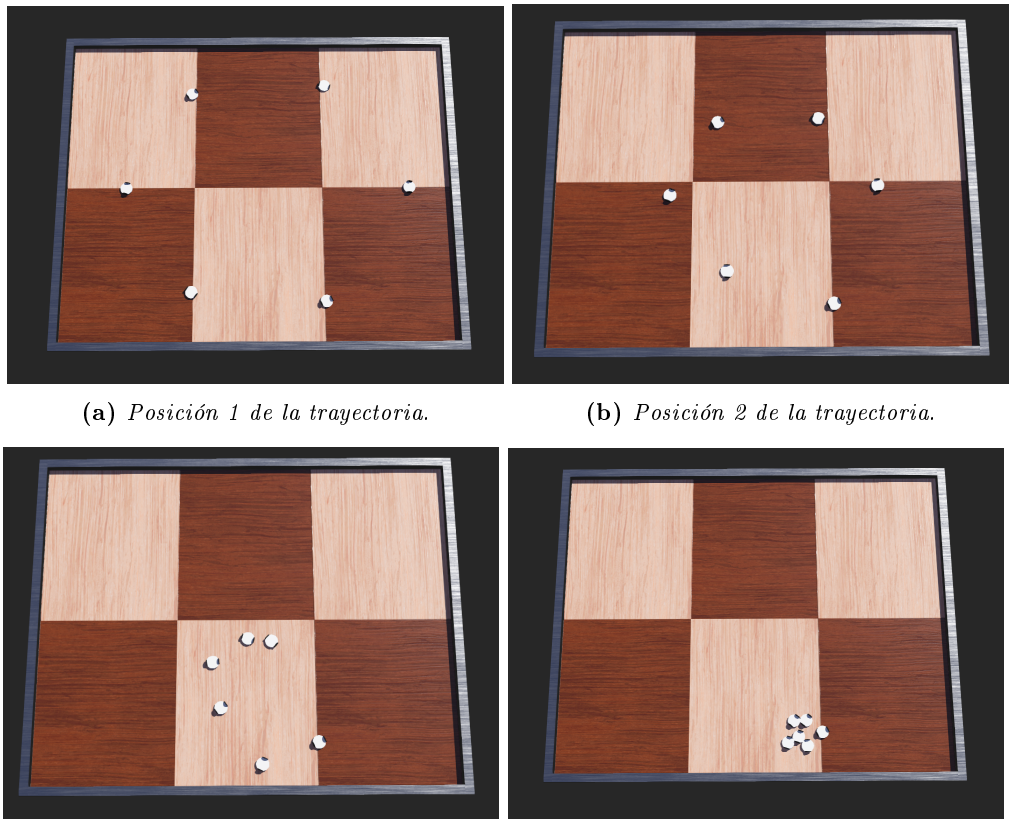
**Cuadro 52:** *Tiempo de convergencia y global best de función Booth con el algoritmo paralelizado en Python en Webots.*

| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Booth | 15.61 | (1.4138, 0.8267) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 82. Para esta función, los agentes llegaron al mínimo global encontrado por el enjambre.

**Figura 82:** *Trayectoria generada con la función Booth con el algoritmo paralelizado en Python.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

La tercera función evaluada fue Schaffer. En el Cuadro 53, se presenta el tiempo de convergencia y el *global best* encontrado por el enjambre.

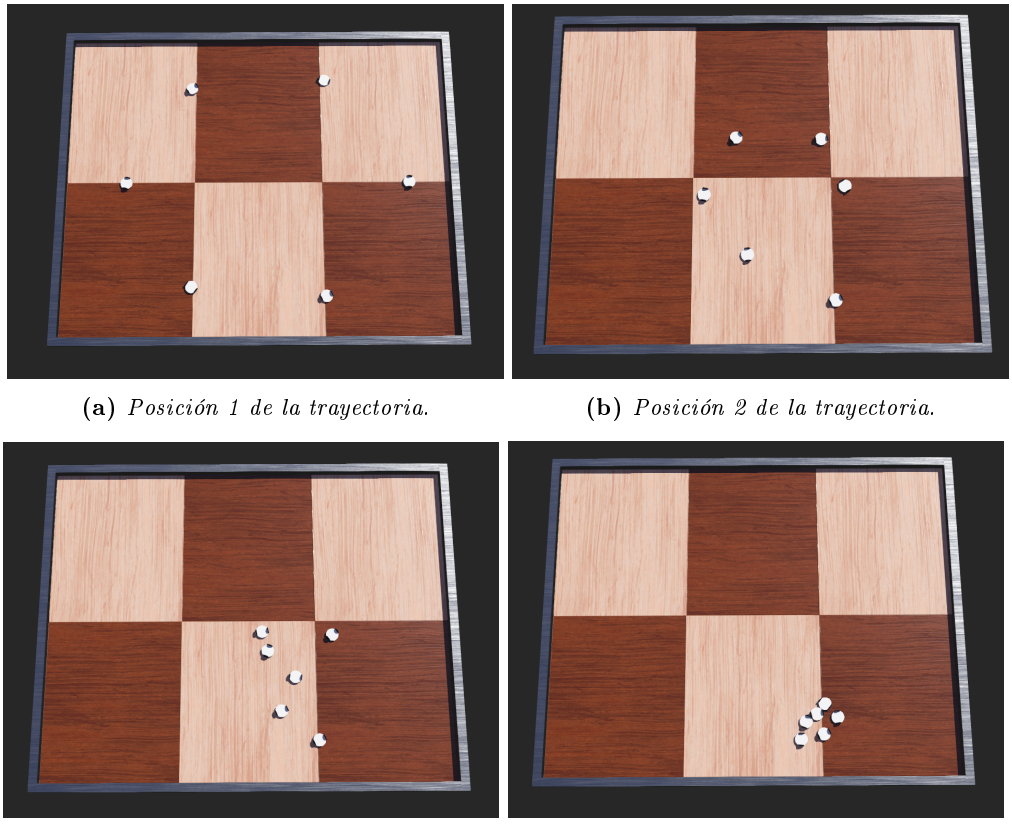
**Cuadro 53:** *Tiempo de convergencia y global best de función Booth con el algoritmo paralelizado en Python en Webots.*

| No. | Prueba | Tiempo (s) | Global best (m) |
|---|---|---|---|
| 1 | Schaffer | 15.36 | (1.2518, 0.5023) |

Nota. Elaboración propia.

Las trayectorias generadas por los agentes robóticos simulados se presentan en la Figura 83. Para esta función, los agentes llegaron al mínimo global encontrado por el enjambre.

**Figura 83:** *Trayectoria generada con la función Schaffer con el algoritmo paralelizado en Python.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

## 10.3. Análisis de resultados con el algoritmo MPSO

A continuación, se presentan los resultados de la comparación de los tiempos de convergencia obtenidos al implementar las mejoras tanto en el entorno físico como simulado.

### 10.3.1. Evaluación de resultados del algoritmo MPSO físico

Dentro de los resultados se presenta la comparación entre los tiempos de convergencia del algoritmo MPSO original implementado en físico con los tiempos del algoritmo MPSO vectorizado.

Los resultados obtenidos se presentan en el Cuadro 54, donde se observa que al implementar las funciones de costo vectorizadas y el parámetro de inercia vectorizado, presentan un menor tiempo de convergencia en comparación al algoritmo original.

Asimismo, al comparar los tiempos promedio del algoritmo MPSO utilizando las funciones de costo, el parámetro de inercia y el controlador PID vectorizado con los del algoritmo original, se observa que no hubo mejora en los tiempos. Al contrario, hubo un aumento en los tiempos para las funciones Sphere y Schaffer.

**Cuadro 54:** *Tiempos de convergencia del algoritmo MPSO vectorizado y original.*

| Funciones de costo | Tiempo promedio algoritmo original (s) | Tiempo promedio con funciones vectorizadas (s) | Tiempo promedio con fitness y parámetro de inercia vectorizado (s) | Tiempo promedio con funciones de costo, parámetro de inercia y controlador vectorizado (s) |
|---|---|---|---|---|
| Sphere | 55.33 | 45.43 | 43.35 | 56.71 |
| Booth | 51.75 | 33.69 | 32.99 | 44.51 |
| Schaffer | 42.66 | 52.88 | 52.87 | 54.92 |

Nota. Elaboración propia.

Al comparar los tiempos de la primera prueba con la segunda junto con el algoritmo original, se observa que la prueba que utiliza las funciones de costo y el parámetro de inercia vectorizados presenta tiempos menores. Esto indica que el rendimiento del algoritmo MPSO es mejor con ambas partes vectorizadas.

En el Cuadro 55, se presentan los resultados del algoritmo MPSO paralelizado. Se observa que este método mejoró el tiempo de convergencia para las funciones Sphere y Booth en comparación al algoritmo original. Sin embargo, para la función Schaffer el algoritmo paralelizado muestra un tiempo de convergencia mayor.

**Cuadro 55:** *Tiempos de convergencia del algoritmo MPSO paralelizado y original.*

| **Funciones de costo** | **Tiempo promedio algoritmo original (s)** | **Tiempo promedio algoritmo paralelizado (s)** |
|---|---|---|
| Sphere | 55.33 | 39.66 |
| Booth | 51.75 | 38.19 |
| Schaffer | 42.66 | 50.22 |

Nota. Elaboración propia.

Al comparar los resultados del método de paralelización con el enfoque de vectorización mostrado en el Cuadro 54, se observa una mejora significativa en el tiempo promedio de convergencia. Esto muestra que la paralelización es fue un buen método para mejorar el rendimiento del algoritmo MPSO.

### 10.3.2. Evaluación de resultados del algoritmo MPSO simulado

A continuación, se presenta la comparación entre los tiempos de convergencia del algoritmo MPSO original implementado en el simulador Webots con los tiempos del algoritmo migrado y paralelizado en Python. En el Cuadro 56, se observa una mejora significativa tanto en el algoritmo migrado a Python como el paralelizado, ya que los tiempos de convergencia son menores en comparación al algoritmom original. Esto indica que el método de migración del algoritmo MPSO al lenguaje de Python presentó un buen rendimiento, aumentando la eficiencia del algoritmo.

**Cuadro 56:** *Tiempos de convergencia para algoritmo MPSO en diferentes versiones.*

| **Funciones de costo** | **Tiempo promedio algoritmo original (s)** | **Tiempo promedio algoritmo migrado a Python (s)** | **Tiempo promedio algoritmo paralelizado en Python (s)** |
|---|---|---|---|
| Sphere | 17.31 | 11.77 | 11.55 |
| Booth | 26.33 | 18.17 | 15.61 |
| Schaffer | 24.60 | 21.3 | 15.36 |

Nota. Elaboración propia.

# CAPÍTULO 11

## Implementación del algoritmo PSO con campos potenciales artificiales para evasión de obstáculos

En este capítulo se presenta la implementación del algoritmo PSO con campos potenciales artificiales (APF) para la evasión de obstáculos. Se desarrolló una implementación en tiempo real para verificar el correcto funcionamiento del algoritmo.

A partir de esto, se diseñó un planificador de trayectorias capaz de generar rutas libres de obstáculos hacia la meta. Para validar estos algoritmos, se realizaron distintos experimentos utilizando obstáculos rectagulares.

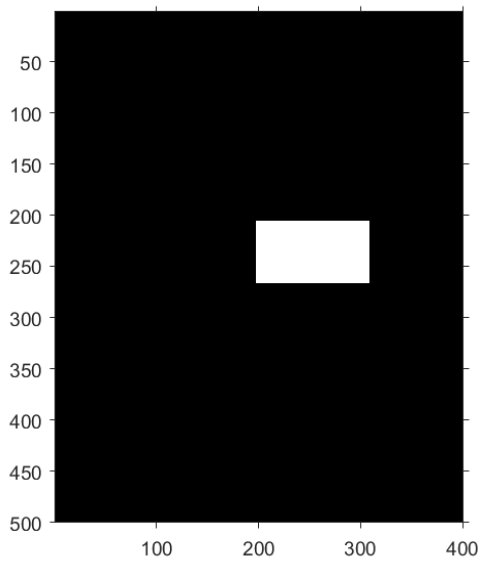
## 11.1. Algoritmo PSO como planificador de trayectorias

La implementación del algoritmo PSO como planificador de trayectorias, utilizando campos potenciales artificiales, implicó la evaluación y obtención de las mejores trayectorias a partir de un punto inicial, que simuló la posición inicial del agente robótico. Esto permitió obtener trayectorias libres de obstáculos, las cuales fueron implementadas en el simulador Webots como seguimiento de trayectoria.

### 11.1.1. Creación de los campos potenciales artificiales

La creación de los campos potenciales implicó el desarrollo de un mapa utilizando la técnica de *Occupancy Grids*. En este mapa, se emplearon las coordenadas $(x, y)$ de los obstáculos para generar prismas rectangulares, asignando un valor de 1 a las celdas ocupadas por los obstáculos y 0 a las celdas libres, como se ilustra en la Figura 84.

**Figura 84:** *Creación del mapa con un obstáculo rectangular.*

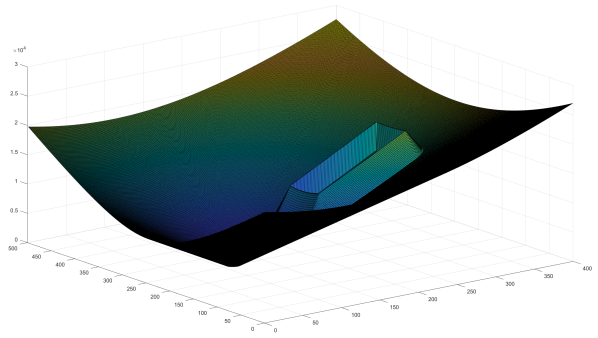



Nota. Elaboración propia.

Es importante mencionar que el eje $x$ del mapa está reflejado con respecto del eje $x$ del Robotat. Además, fue necesario realizar un mapeo de las coordenadas del Robotat a coordenadas del mapa, dado que el mapa tiene la estructura de una matriz que admite únicamente valores positivos. La escala del mapa se definió en unidades de cm.

Los campos potenciales fueron calculados de forma matricial. A partir del mapa, se crearon los campos atractivo y repulsivo para cada celda, generando un mapa del campo total, como se muestra en la Figura 85.

**Figura 85:** *Campo total potencial generado a partir del mapa.*

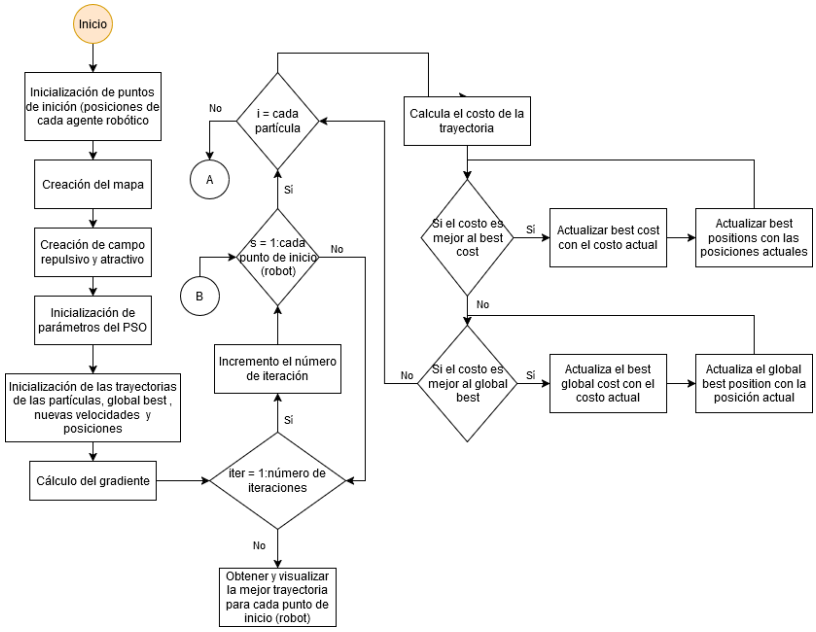


Nota. Elaboración propia.

### 11.1.2. Estructura del algoritmo PSO como planificador de trayectorias

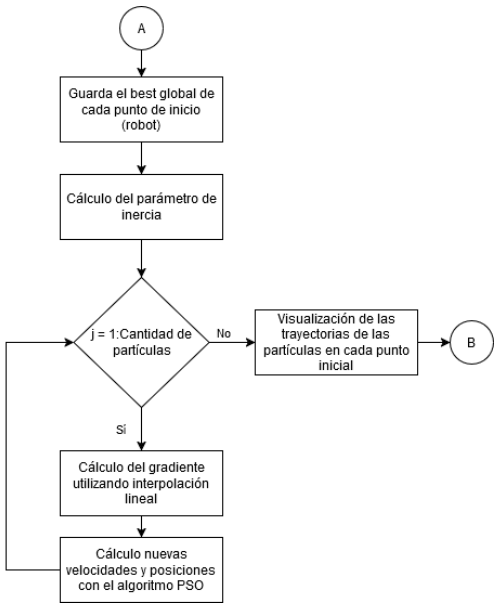
La estructura principal del planificador de trayectorias se basa en la estructura inicial del algoritmo PSO, con algunas modificaciones, como se ilustra en la Figura 86 y Figura 87. Entre estas modificaciones se encuentra la definición de la función para evaluar el costo de las trayectorias, la incorporación del gradiente del campo potencial en las velocidades de las partículas y la inicialización de las trayectorias individuales para cada partícula.

**Figura 86:** *Estructura del planificador con el algoritmo PSO parte 1.*




Nota. Elaboración propia.

**Figura 87:** *Estructura del planificador con el algoritmo PSO parte 2.*




Nota. Elaboración propia.

El planificador emplea tres funciones para su correcto funcionamiento. La primera función se encarga de inicializar las trayectorias, las velocidades, los costos y las mejores posiciones de cada robot (punto de inicio).

La segunda función genera trayectorias iniciales aleatorias mediante un número máximo de pasos y pasos aleatorios, asegurando que las trayectorias tengan tamaño adecuado. Además, verifica si los puntos generadas para la trayectoria están ubicados sobre un obstáculo y se asegura que se alcanza el punto final deseado.

La tercer función calcula el costo asociado a cada punto de las trayectorias. Este cálculo penaliza las que salen del mapa, aquellas que se aproximan demasiado a los obstáculos y las que se alejan del objetivo final.

### 11.1.3. Implementación del planificador de trayectorias en matlab y webots

La validación del planificador basado en el algoritmo PSO se llevó a cabo en Matlab y en Webots. Matlab se utilizó para verificar y ajustar los parámetros del planificador, con el fin de obtener los resultados deseados. Posteriormente, el simulador Webots permitió evaluar el planificador por medio de seguimiento de trayectorias generadas, empleando cuatro agentes robóticos Pololu 3pi+.

Para la primera prueba, se utilizaron los parámetros presentados en el Cuadro 68. Las posiciones iniciales de los agentes se presentan en el Cuadro 57. Las posiciones del obstáculo y la meta, en el Cuadro 58.

**Cuadro 57:** *Posiciones iniciales de los agentes Pololu 3pi+ para la primera prueba en Webots.*

| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (-0.5205, -2.3284) |
| 2 | (-1.4733, -2.1839) |
| 3 | (0.3897, -2.3148) |
| 4 | (1.2702, -2.1881) |

Nota. Elaboración propia.

**Cuadro 58:** *Posiciones del obstáculo y meta para la primera prueba en Webots.*

| Descripción | Posición (m) |
|---|---|
| Obstáculo 1 | (-0.1135, -1.1366) |
| Meta | (0.2194, 1.6341) |

Nota. Elaboración propia.

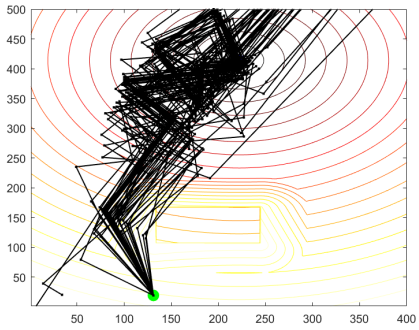
Para esta prueba, se evaluaron distintos parámetros para el algoritmo PSO, núm. de partículas y número de pasos (Cuadro 59).

**Cuadro 59:** *Parámetros del algoritmo PSO para la primera prueba con un obstáculo en Webots.*

| Descripción | Ajuste seleccionado |
|---|---|
| Tipo de inercia | Constante |
| Factor de constricción | 0.40 |
| Peso cognitivo | 2 |
| Peso social | 10 |
| Escalador de velocidad | 0.4 |
| Número máx. de pasos | 10 |
| No. de partículas | 100 |

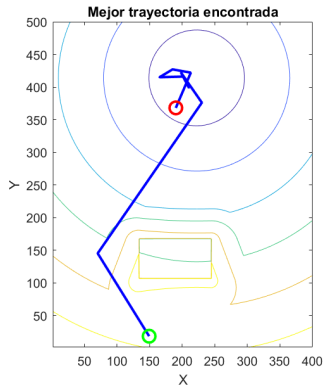
Nota. Elaboración propia.

**Figura 88:** *Planificador de trayectorias para la primera prueba.*

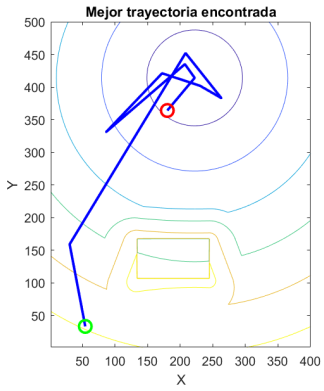


Nota. Elaboración propia.

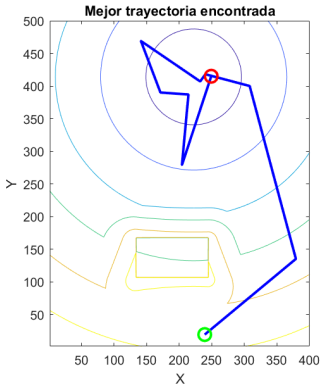
**Figura 89:** *Trayectorias obtenidas por el planificador utilizando el algoritmo PSO para la primera prueba.*



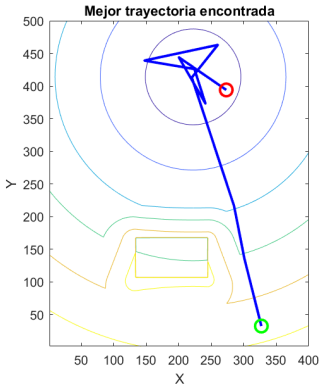

**(a)** *Trayectoria agente robótico 1.*




**(b)** *Trayectoria agente robótico 2.*




**(c)** *Trayectoria agente robótico 3.*




**(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

Durante la simulación de esta pruba Figura 88, se observó que el planificador fue capaz de encontrar trayectorias libres de obstáculos para cada agente robótico, como se ilustra en la Figura 89. Sin embargo, algunas trayectorias no lograron converger completamente hacia la meta deseada. Cabe resaltar que el círculo verde representa la posición inicial del agente robótico, mientras que el círculo rojo indica la posición final de la trayectoria.

Debido a que el algoritmo PSO genera trayectorias con cambios poco suaves, se implementó un suavizador para mejorar la continuidad de las trayectorias, como se ilustra en la Figura 91. En la Figura 90, se presenta la simulación del seguimiento de estas trayectorias utilizando el simulador Webots y los agentes robóticos Pololu 3pi+.

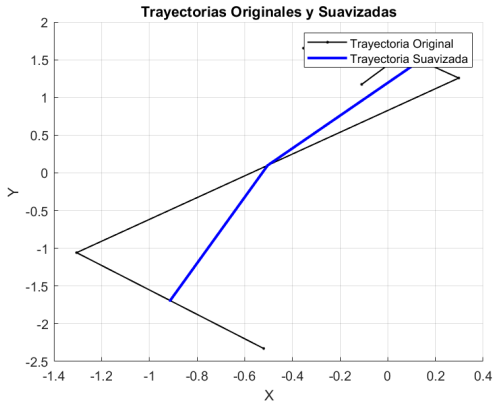
**Figura 90:** *Simulación de seguimiento de trayectorias obtenidas por el planificador en Webots con un obstáculo.*

**(a)** *Trayectoria agente robótico 1.* **(b)** *Trayectoria agente robótico 2.*

**(c)** *Trayectoria agente robótico 3.* **(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

**Figura 91:** *Trayectoria original y suavizada para el agente robótico 1.*




Nota. Elaboración propia.

Para la segunda prueba, se emplearon los parámetros de los campos potenciales y las posiciones iniciales de la prueba anterior. Las posiciones de los obstáculos y la meta se presentan en el Cuadro 60. Asimismo, los parámetros del algoritmo PSO utilizados para esta prueba se presentan en el Cuadro 61.

**Cuadro 60:** *Posiciones de los obstáculos y meta para la segunda prueba en Webots.*

| **Descripción** | **Posición (m)** |
|---|---|
| Obstáculo 1 | (-1.1155, -0.7200) |
| Obstáculo 2 | (0.7301,-1.0952) |
| Meta | (0.0053, 1.1128) |

Nota. Elaboración propia.

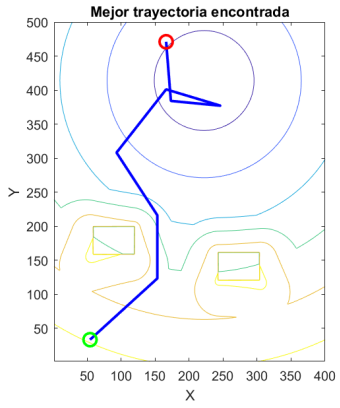
**Cuadro 61:** *Parámetros del algoritmo PSO para la segunda prueba con dos obstáculos en Webots.*

| **Descripción** | **Ajuste seleccionado** |
|---|---|
| Tipo de inercia | Constante |
| Factor de constricción | 0.55 |
| Peso cognitivo | 5 |
| Peso social | 15 |
| Escalador de velocidad | 0.4 |
| Número máx. de pasos | 10 |
| No. de partículas | 150 |

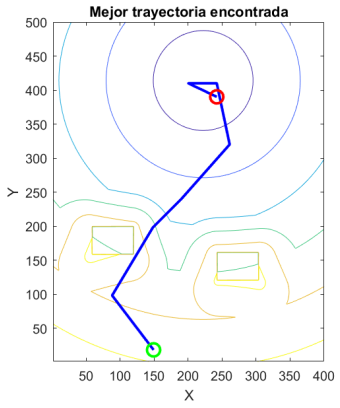
Nota. Elaboración propia.

Durante la simulación de esta prueba Figura 93, se observó que el planificador fue capaz de encontrar trayectorias libres de obstáculos para cada agente robótico, como se ilustra en la Figura 92. Sin embargo, las trayectorias no lograron converger a la meta deseada.

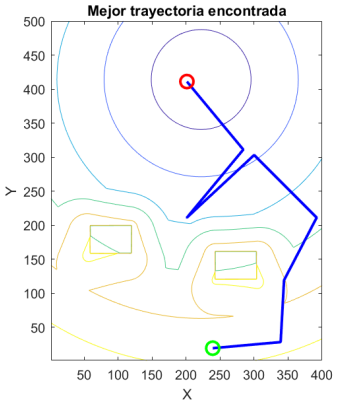
**Figura 92:** *Trayectorias obtenidas por el planificador utilizando el algoritmo PSO para la segunda prueba.*



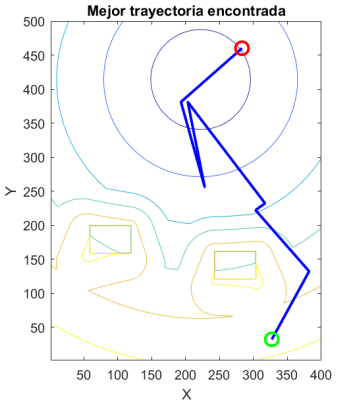

**(a)** *Trayectoria agente robótico 1.*




**(b)** *Trayectoria agente robótico 2.*



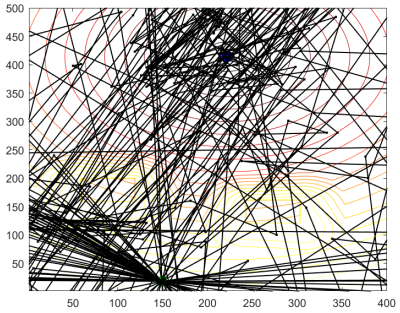

**(c)** *Trayectoria agente robótico 3.*




**(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

**Figura 93:** *Planificador de trayectorias para la segunda prueba.*




Nota. Elaboración propia.

**Figura 94:** *Simulación de seguimiento de trayectorias obtenidas por el planificador en Webots con dos obstáculos.*

**(a)** *Trayectoria agente robótico 1.* **(b)** *Trayectoria agente robótico 2.*

**(c)** *Trayectoria agente robótico 3.* **(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

En la Figura 94, se presenta la simulación del seguimiento de las trayectorias obtenidas para esta prueba, realizadas en el simulador Webots con los agentes robóticos Pololu 3pi+.

Para la tercera prueba, se emplearon los parámetros de los campos potenciales y posiciones iniciales de la prueba anterior. Las posiciones de los obstáculos y la meta se presentan en el Cuadro 62. Asimismo, los parámetros del algoritmo PSO utilizados para esta prueba se presentan en el Cuadro 63.

**Cuadro 62:** *Posiciones de los obstáculos y meta para la tercer prueba en Webots.*

| **Descripción** | **Posición (m)** |
|---|---|
| Obstáculo 1 | (-1.0846, -0.6775) |
| Obstáculo 2 | (0.7757,-1.2663) |
| Obstáculo 3 | (0.3052, 0.2337) |
| Meta | (0.2194, 1.6341) |

Nota. Elaboración propia.

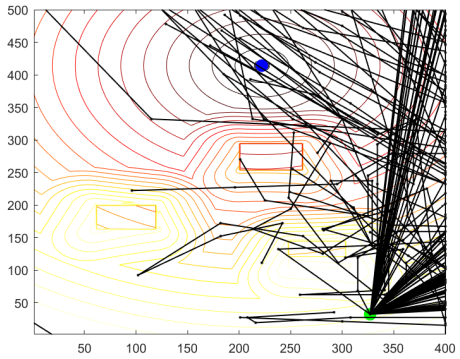
**Cuadro 63:** *Parámetros del algoritmo PSO para la tercer prueba con tres obstáculos en Webots.*

| **Descripción** | **Ajuste seleccionado** |
|---|---|
| Tipo de inercia | Constante |
| Factor de constricción | 0.2087 |
| Peso cognitivo | 2 |
| Peso social | 7 |
| Escalador de velocidad | 0.4 |
| Número máx. de pasos | 10 |
| No. de partículas | 100 |

Nota. Elaboración propia.

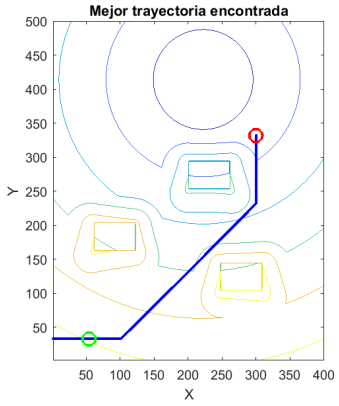
Durante la simulación de esta prueba Figura 95, se observó que el planificador de trayectorias encontró trayectorias libres de obstáculos para cada agente robótico, como se ilustra en la Figura 96. Sin embargo, algunas de las trayectorias no lograron converger a la meta.

**Figura 95:** *Planificador de trayectorias para la tercer prueba.*

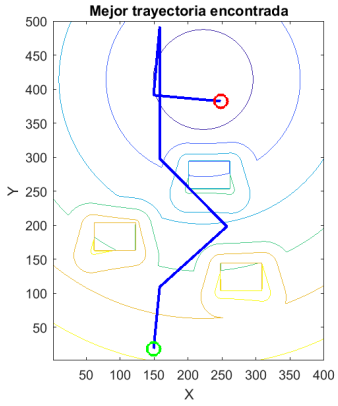



Nota. Elaboración propia.

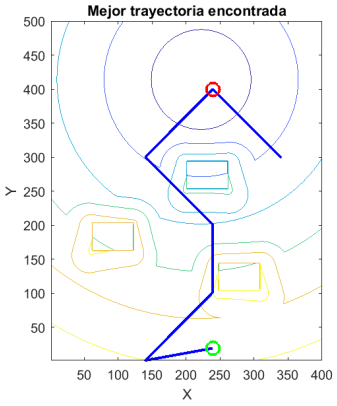
**Figura 96:** *Trayectorias obtenidas por el planificador utilizando el algoritmo PSO para la tercer prueba..*



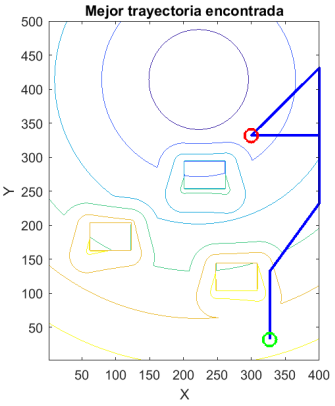

**(a)** *Trayectoria agente robótico 1.*




**(b)** *Trayectoria agente robótico 2.*




**(c)** *Trayectoria agente robótico 3.*

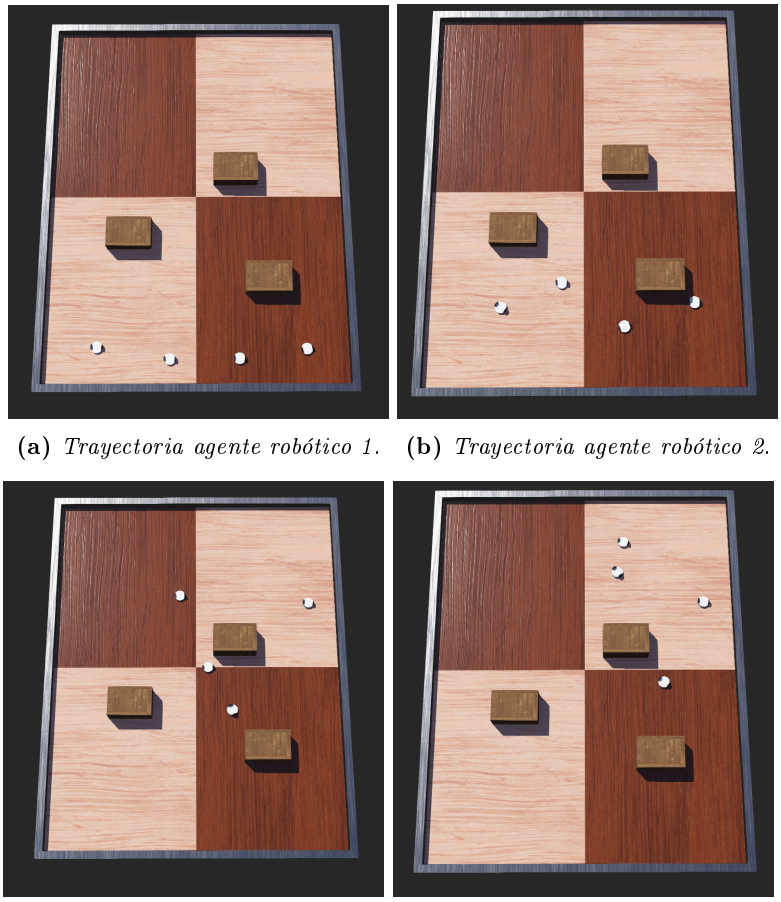



**(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

En la Figura 97, se presenta la simulación del seguimiento de las trayectorias obtenidas en el simulador Webots.

**Figura 97:** *Simulación de seguimiento de trayectorias obtenidas por el planificador en Webots con tres obstáculos.*



**(a)** *Trayectoria agente robótico 1.* **(b)** *Trayectoria agente robótico 2.*

**(c)** *Trayectoria agente robótico 3.* **(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

Para la cuarta prueba, se emplearon los parámetros de los campos potenciales y posiciones iniciales de la prueba anterior. Las posiciones de los obstáculos y la meta se presentan en el Cuadro 64. Asimismo, los parámetros del algoritmo PSO utilizados para esta prueba se presentan en el Cuadro 65.

**Cuadro 64:** *Posiciones de los obstáculos y meta para la cuarta prueba en Webots.*

| Descripción | Posición (m) |
|---|---|
| Obstáculo 1 | (-1.0846, -0.6775) |
| Obstáculo 2 | (0.7757,-1.2663) |
| Obstáculo 3 | (0.3052, 0.2337) |
| Obstáculo 4 | (-1.1433, 1.1192) |
| Meta | (0.2194, 1.6341) |

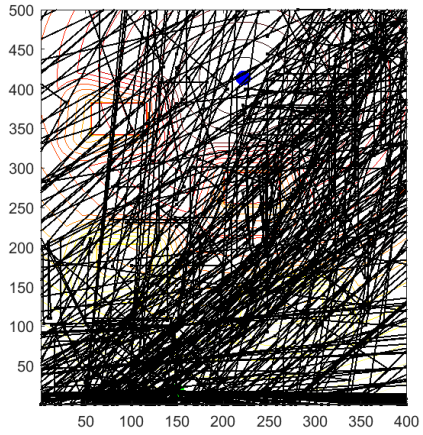
Nota. Elaboración propia.

**Cuadro 65:** *Parámetros del algoritmo PSO para la cuarta prueba con cuatro obstáculos en Webots.*

| Descripción | Ajuste seleccionado |
|---|---|
| Tipo de inercia | Constante |
| Factor de constricción | 0.25 |
| Peso cognitivo | 2 |
| Peso social | 50 |
| Escalador de velocidad | 0.4 |
| Número máx. de pasos | 10 |
| No. de partículas | 500 |

Nota. Elaboración propia.

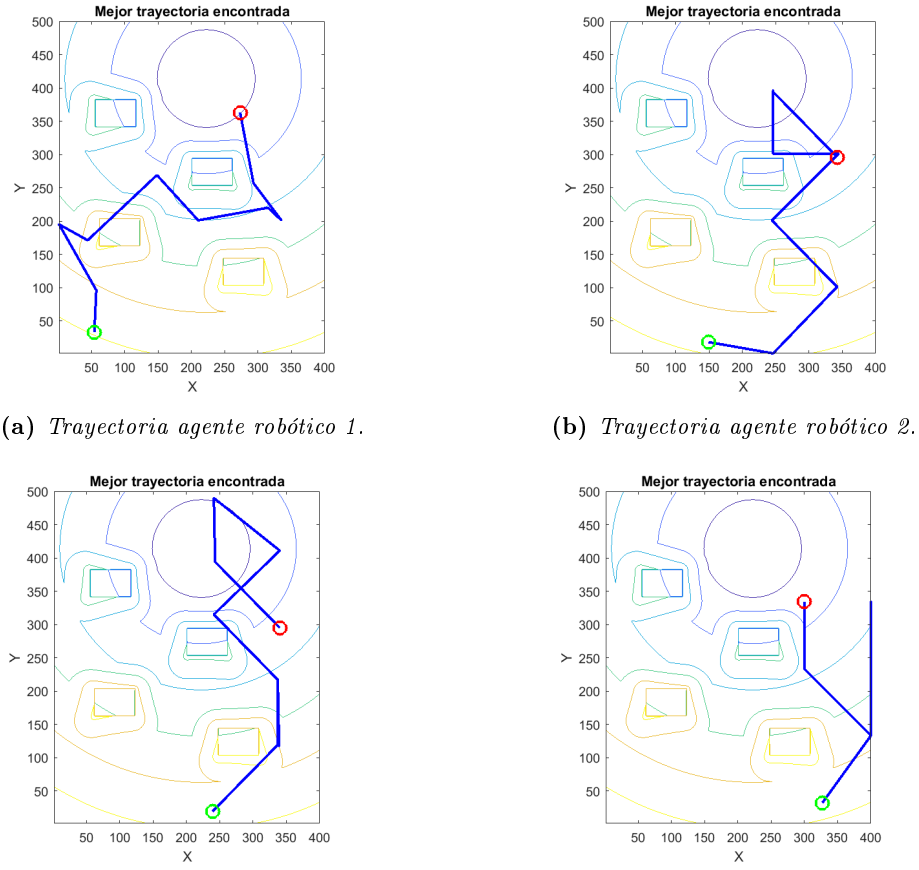
**Figura 98:** *Planificador de trayectorias para la cuarta prueba.*




Nota. Elaboración propia.

Durante la simulación de esta prueba Figura 98, se observó que el planificador encontró trayectorias libres de obstáculos para cada agente robótico, como se ilustra en la Figura 99. Sin embargo, dos de las cuatro trayectorias no lograron converger a la meta deseada.

**Figura 99:** *Trayectorias obtenidas por el planificador utilizando el algoritmo PSO para la cuarta prueba.*




**(a)** *Trayectoria agente robótico 1.*

**(b)** *Trayectoria agente robótico 2.*

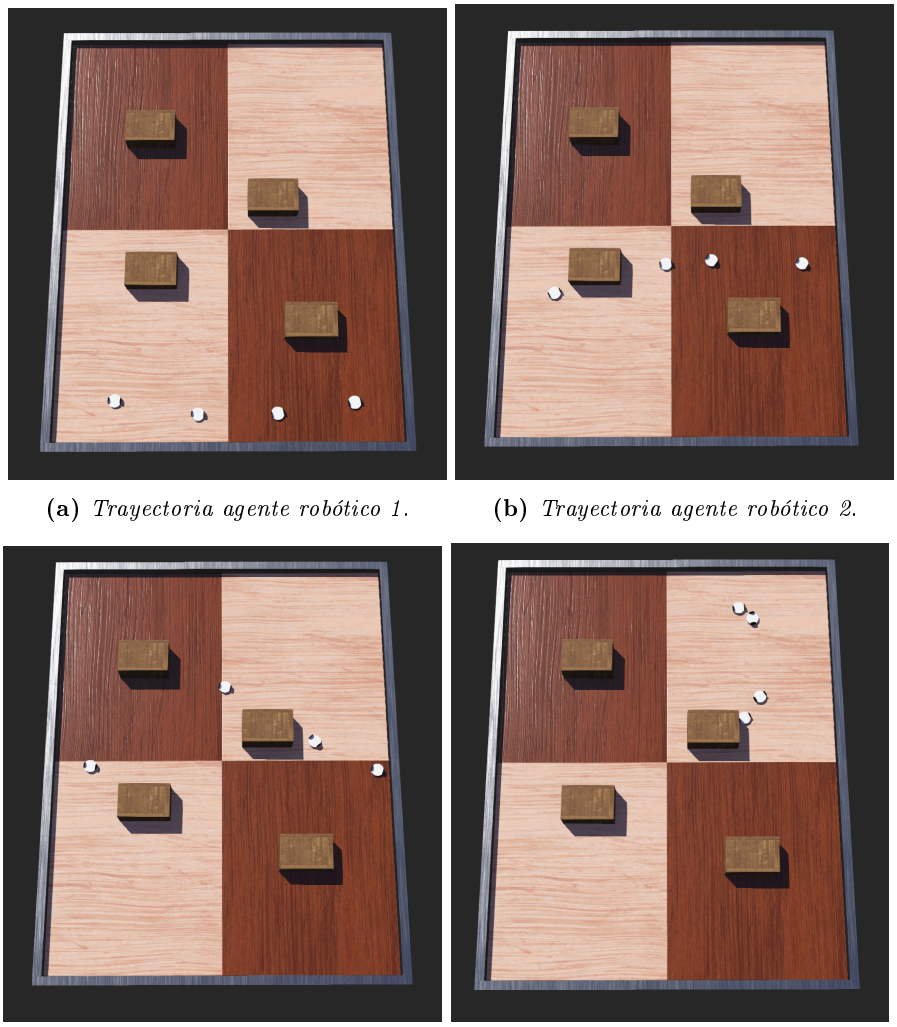
**(c)** *Trayectoria agente robótico 3.*

**(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

En la Figura 100, se presenta la simulación del seguimiento de las trayectorias obtenidas en el simulador Webots.

**Figura 100:** *Simulación de seguimiento de trayectorias obtenidas por el planificador en Webots con cuatro obstáculos.*



**(a)** *Trayectoria agente robótico 1.* **(b)** *Trayectoria agente robótico 2.*

**(c)** *Trayectoria agente robótico 3.* **(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

Para la quinta prueba, se emplearon los parámetros de los campos potenciales y posiciones iniciales de la prueba anterior. Las posiciones de los obstáculos y la meta se presentan en el Cuadro 66.

**Cuadro 66:** *Posiciones de los obstáculos y meta para la quinta prueba en Webots.*

| **Descripción** | **Posición (m)** |
|---|---|
| Obstáculo 1 | (-1.4282, -0.6113) |
| Obstáculo 2 | (-0.6971,-1.3765) |
| Obstáculo 3 | (0.8691,-0.0501) |
| Obstáculo 4 | (-1.1108,0.8580) |
| Obstáculo 5 | (1.0712,-1.1929) |
| Obstáculo 6 | (-0.2273,-0.2118) |
| Meta | (0.2194, 1.6341) |

Nota. Elaboración propia.

Asimismo, los parámetros del algoritmo PSO utilizados para esta prueba se presentan en el Cuadro 67.

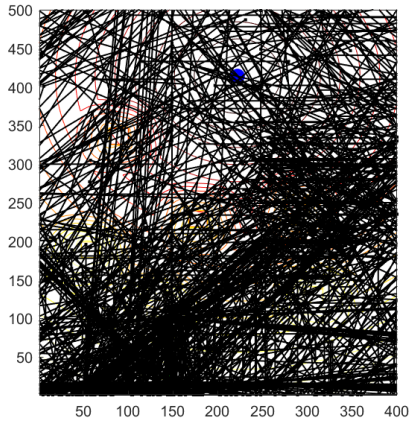
**Cuadro 67:** *Parámetros del algoritmo PSO para la quinta prueba con seis obstáculos en Webots.*

| **Descripción** | **Ajuste seleccionado** |
|---|---|
| Tipo de inercia | Constante |
| Factor de constricción | 0.40 |
| Peso cognitivo | 5 |
| Peso social | 35 |
| Escalador de velocidad | 0.4 |
| Número máx. de pasos | 10 |
| No. de partículas | 500 |

Nota. Elaboración propia.

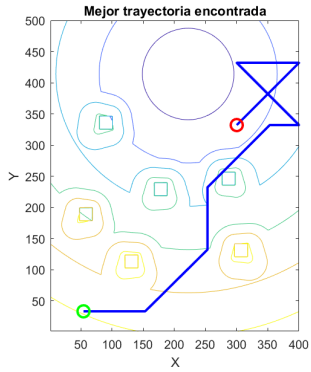
Durante la simulación de esta prueba Figura 101, se observó que el planificador encontró trayectorias libres de obstáculos para cada agente robótico, como se ilustra en la Figura 102. Sin embargo, solo uno de las cuatro trayectorias no logró converger a la meta deseada.

**Figura 101:** *Planificador de trayectorias para la quinta prueba.*

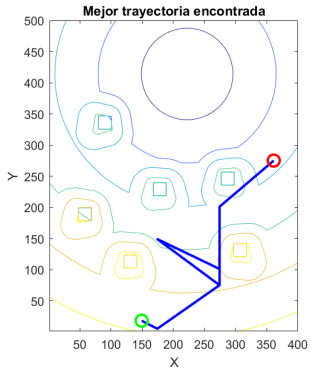



Nota. Elaboración propia.

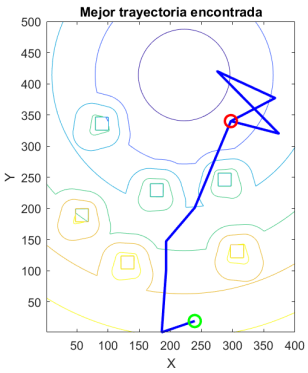
**Figura 102:** *Trayectorias obtenidas por el planificador utilizando el algoritmo PSO para la quinta prueba.*



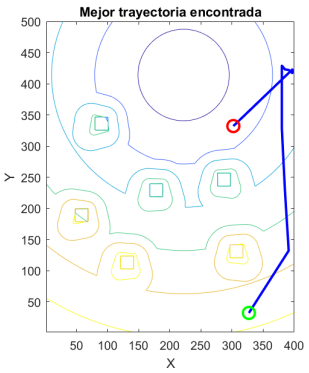

**(a)** *Trayectoria agente robótico 1.*




**(b)** *Trayectoria agente robótico 2.*




**(c)** *Trayectoria agente robótico 3.*

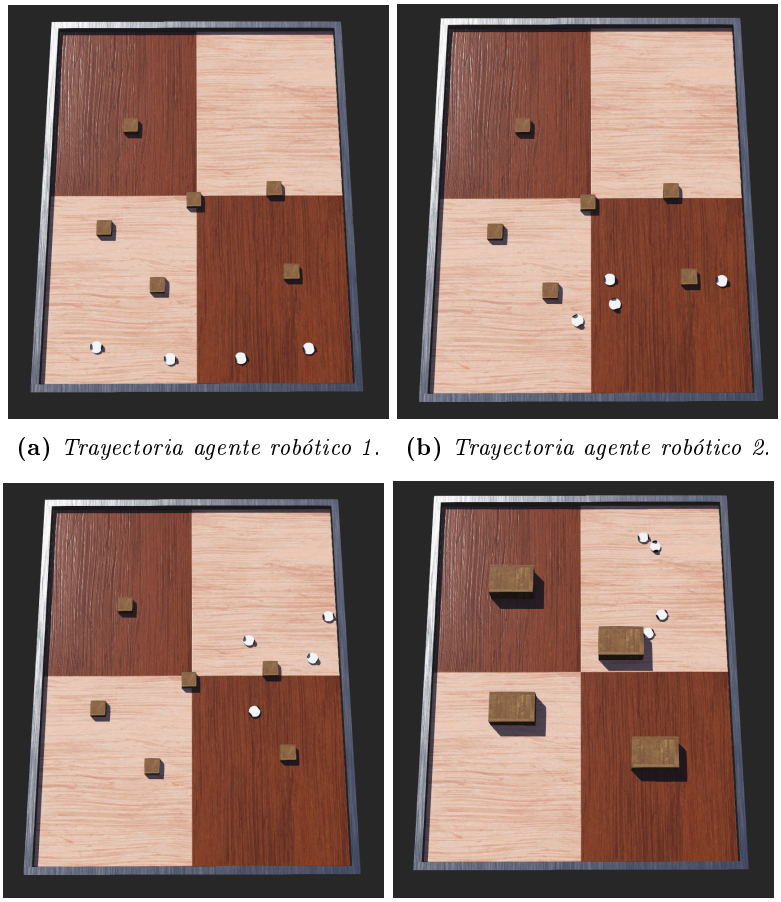



**(d)** *Trayectoria agente robótico 4.*

Nota. Elaboración propia.

En la Figura 103, se presenta la simulación del seguimiento de las trayectorias obtenidas en el simulador Webots.

**Figura 103:** *Simulación de seguimiento de trayectorias obtenidas por el planificador en Webots con seis obstáculos.*



**(a)** *Trayectoria agente robótico 1.* **(b)** *Trayectoria agente robótico 2.*

**(c)** *Trayectoria agente robótico 3.* **(d)** *Trayectoria agente robótico 5.*

Nota. Elaboración propia.

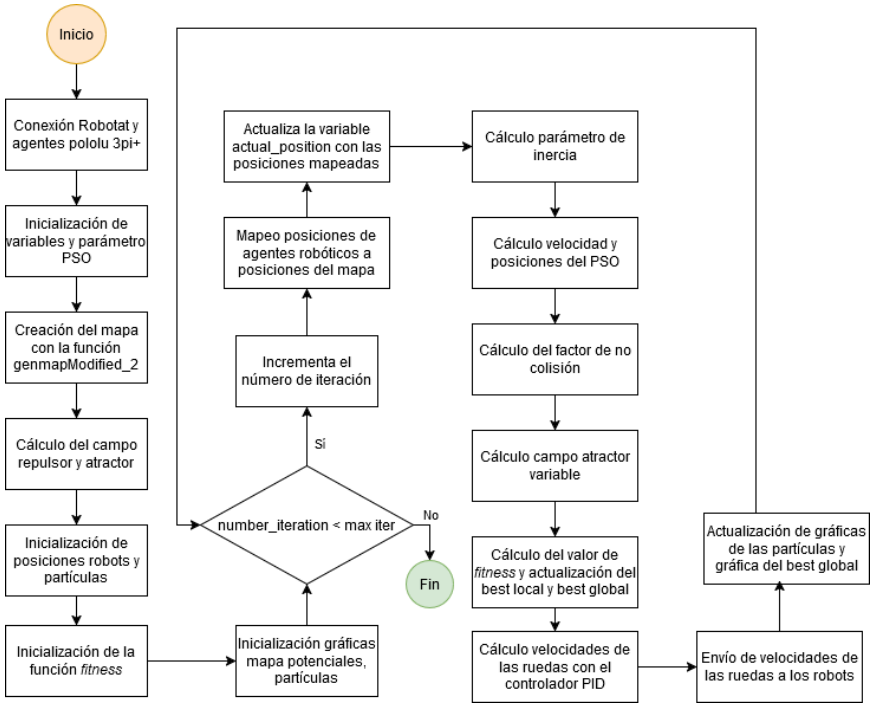
## 11.2. Algoritmo PSO con campos potenciales artificiales

La implementación del algoritmo PSO con campos potenciales artificiales requirió realizar modificaciones al algoritmo MPSO. En base al trabajo desarrollado por [11], se llevó a cabo una nueva implementación del algoritmo PSO con campos potenciales artificiales.

### 11.2.1. Estructura del algoritmo PSO con campos potenciales artificiales

La modificación principal realizada al algoritmo PSO fue la implementación de más particulas. Esto se implementó con el objetivo de encontrar de forma más rápida la mejor posición y lograr que el algoritmo converja a un mínimo global. La estructura del algoritmo implementado en físico se muestra en la Figura 104.

**Figura 104:** *Diagrama de flujo de la estructura del algoritmo PSO con campos potenciales artificiales.*




Nota. Elaboración propia.

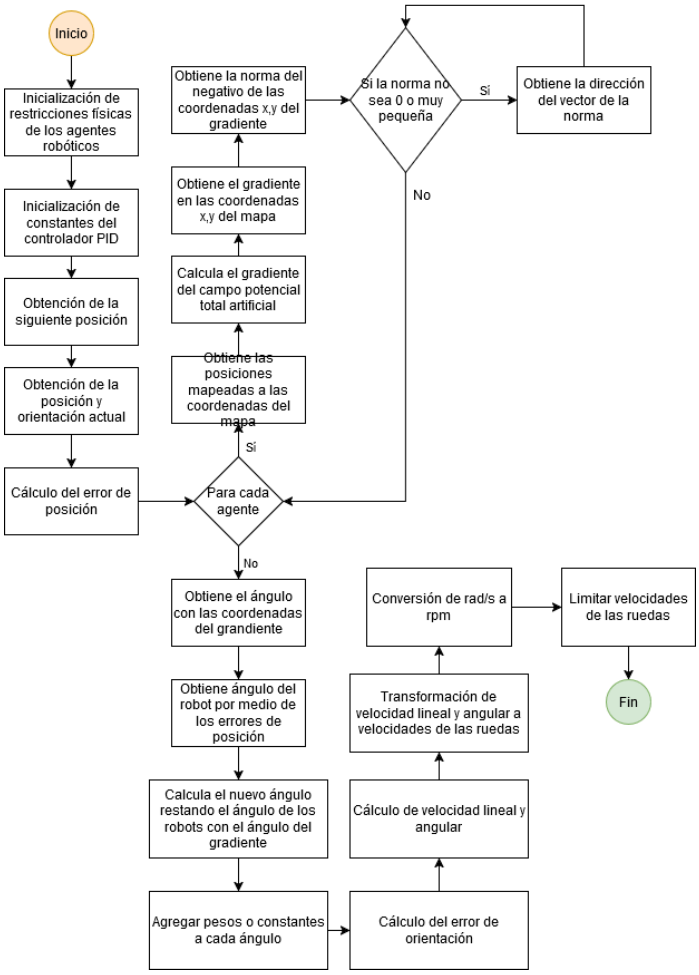
Para ello, se definieron las posiciones iniciales de un conjunto de partículas, así como de robots diferenciales a utilizar, junto con sus posiciones y orientaciones iniciales. Además, se implementó la inicialización del cálculo de la función *fitness* para cada partícula y agente robótico.

Otra modificación importante fue el cálculo de la función *fitness*. En este caso, se uitlizó como función de costo el valor del campo total potencial en la posición actual $(x, y)$ de los robots. Además, se modificó el cálculo del campo atractivo para que fuera variable, incluyendo objetivos temporales por medio de las posiciones generadas por el PSO.

Durante la implementación del algoritmo, se pudo observar que los agentes robóticos no lograban evadir correctamente los obstáculos. Para solucionar este problema, se incorporó el

gradiente de los campos potenciales en la orientación del controlador PID, lo que permitió guiar a los robots hacia la meta, evadiendo los obstáculos. La estructura de esta modificación se muestra en la Figura 105.

**Figura 105:** *Estructura de la modificación realizada al controlador PID para evadir obstáculos implementando campos potenciales artificiales.*




Nota. Elaboración propia.

### 11.2.2. Implementación del algoritmo PSO con campos potenciales en el simulador webots

Se realizó la validación del algoritmo PSO con campos potenciales en el simulador Webots, con el fin de observar si el algoritmo era capaz de ejecutar el PSO para encontrar trayectorias hacia la meta y evadir los obstáculos. Para esto se realizaron tres pruebas utilizando obstáculos rectangulares. En el Cuadro 68, se presentan los parámetros utilizados para cada una de las pruebas.

**Cuadro 68:** *Parámetros de campos potenciales artificiales para pruebas en Webots.*

| Descripción | Ajuste seleccionado |
|---|---|
| Ganancia atractiva (*atractive gain)* | 9 |
| Distancia campo atractivo (*dstar)* | 8 |
| Ganancia repulsiva (*repulsive gain)* | 10000 |
| Distancia campo repulsivo (*rho0*) | 45 |
| Factor $n$ | 2 |

Nota. Elaboración propia.

Para la primera prueba, se utilizó un obstáculo rectangular. En el Cuadro 69 se muestra la posición del obstáculo y la meta. Asimismo, las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 70.

**Cuadro 69:** *Posiciones iniciales de los agentes Pololu 3pi+ para la primera prueba con campos potenciales.*

| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (0.3371, -2.2176) |
| 2 | (-0.8390, -2.3183) |
| 3 | (-1.0263, -1.8454) |
| 4 | (1.0040, -1.9183) |

Nota. Elaboración propia.

**Cuadro 70:** *Posición del obstáculo y meta para la primera prueba con campos artificiales.*

| Descripción | Posición (m) |
|---|---|
| Obstáculo | (-0.0017, -0.8763) |
| Meta | (0.0053, 1.1128) |

Nota. Elaboración propia.

Durante la simulación de esta prueba Figura 106, se observó que todos los agentes robóticos logran evadir el obstáculo y llegar a la meta.

**Figura 106:** *Simulación en Webots con campos potenciales artificiales y un obstáculo rectangular.*

**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

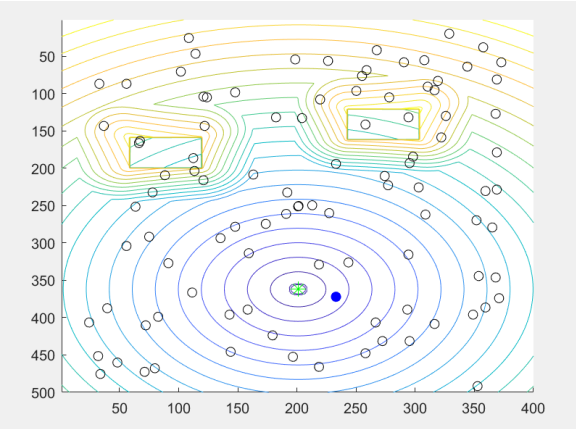
**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

Asimismo, en la Figura 107 se muestra la simulación de todas las partículas y el campo potencial total. Es importante mencionar que los ejes $x$ y $y$ están reflejados con respecto de los ejes originales del Robotat.

Para la segunda prueba, se incluyeron dos obstáculos rectangulares. Las posiciones de estos y la meta se presentan en el Cuadro 60. Asimismo, las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 71.

**Figura 107:** *Simulación de partículas con dos obstáculos en Matlab.*




Nota. Elaboración propia.

**Cuadro 71:** *Posiciones iniciales de los agentes Pololu 3pi+ para la segunda prueba con campos potenciales.*

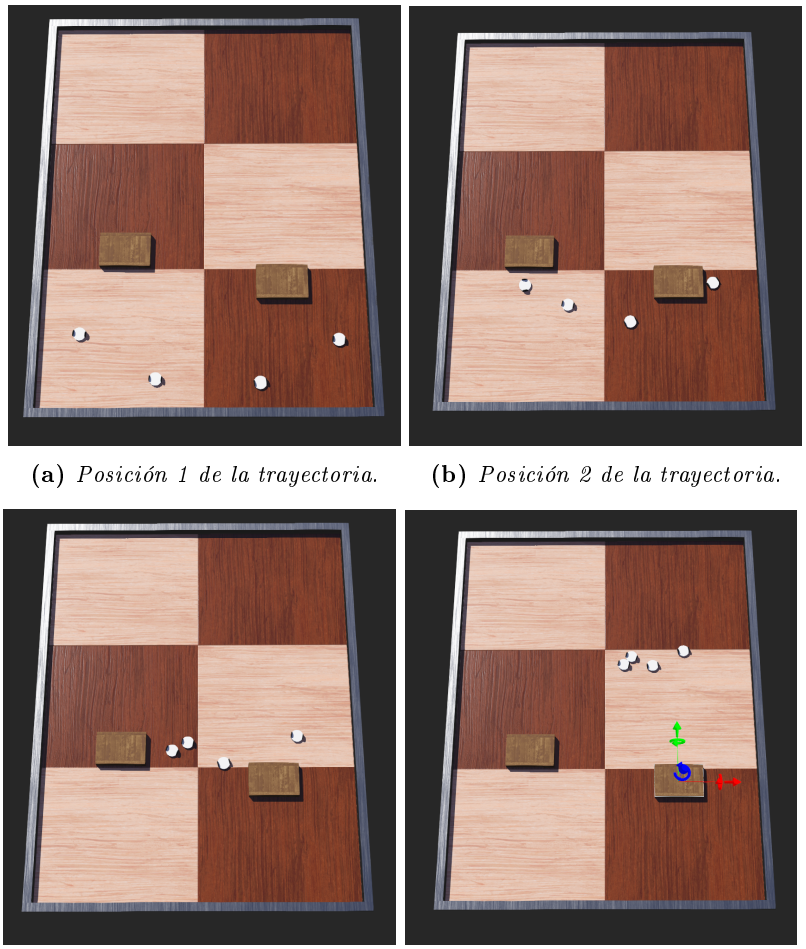
| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (-0.7241, -2.2206) |
| 2 | (-1.6119, -1.7142) |
| 3 | (1.3737, -1.7643) |
| 4 | (0.4656, -2.2540) |

Nota. Elaboración propia.

Durante la simulación de esta prueba Figura 108, se observó que todos los agentes robóticos logran evadir los obstáculos y llegar a la meta. En la Figura 109, se muestra la simulación de las partículas con ambos obstáculos.

Para la tercer prueba, se incluyeron tres obstáculos rectangulares. Las posiciones de estos y la meta se presentan en el Cuadro 62. Asimismo, las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 72.

**Figura 108:** *Simulación en Webots con campos potenciales artificiales y dos obstáculos rectangulares.*

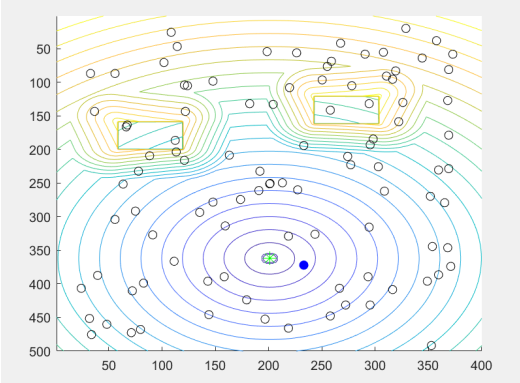


**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

**Figura 109:** *Simulación de partículas con dos obstáculos en Matlab.*




Nota. Elaboración propia.

**Cuadro 72:** *Posiciones iniciales de los agentes Pololu 3pi+ para la tercer prueba con campos potenciales.*

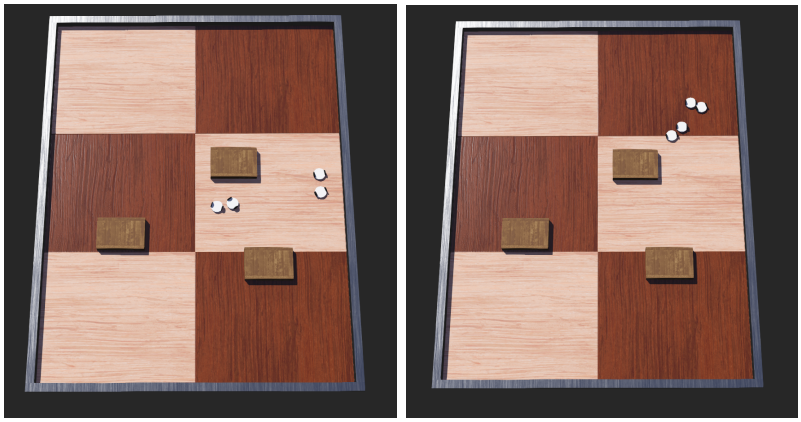
| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (-0.7241, -2.2206) |
| 2 | (-1.6119, -1.7142) |
| 3 | (1.3737, -1.7643) |
| 4 | (0.4656, -2.2540) |

Nota. Elaboración propia.

Durante la simulación de esta prueba Figura 110, se observó que todos los agentes robóticos logran evadir los obstáculos y llegar a la meta. En la Figura 111, se muestra la simulación de las partículas con los obstáculos.

**Figura 110:** *Simulación en Webots con campos potenciales artificiales y tres obstáculos rectangulares.*

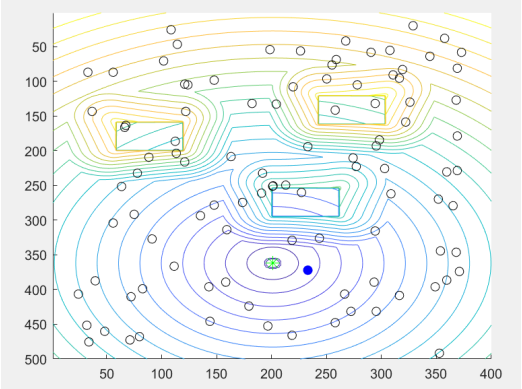
**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

**Figura 111:** *Simulación de partículas con tres obstáculos en Matlab.*




Nota. Elaboración propia.

Para la cuarta prueba, se incluyeron cuatro obstáculos rectangulares. Las posiciones de estos y la meta se presentan en el Cuadro 64. Asimismo, las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 73.

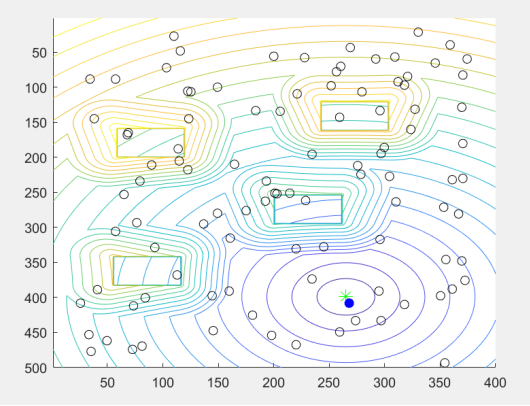
**Cuadro 73:** *Posiciones iniciales de los agentes Pololu 3pi+ para la cuarta prueba con campos potenciales.*

| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (-0.7241, -2.2206) |
| 2 | (-1.6119, -1.7142) |
| 3 | (1.3737, -1.7643) |
| 4 | (0.4656, -2.2540) |

Nota. Elaboración propia.

Durante la simulación Figura 112, se observó que todos los agentes robóticos logran evadir los obstáculos y llegar a la meta. En la Figura 113, se muestra la simulación de las partículas con los obstáculos.

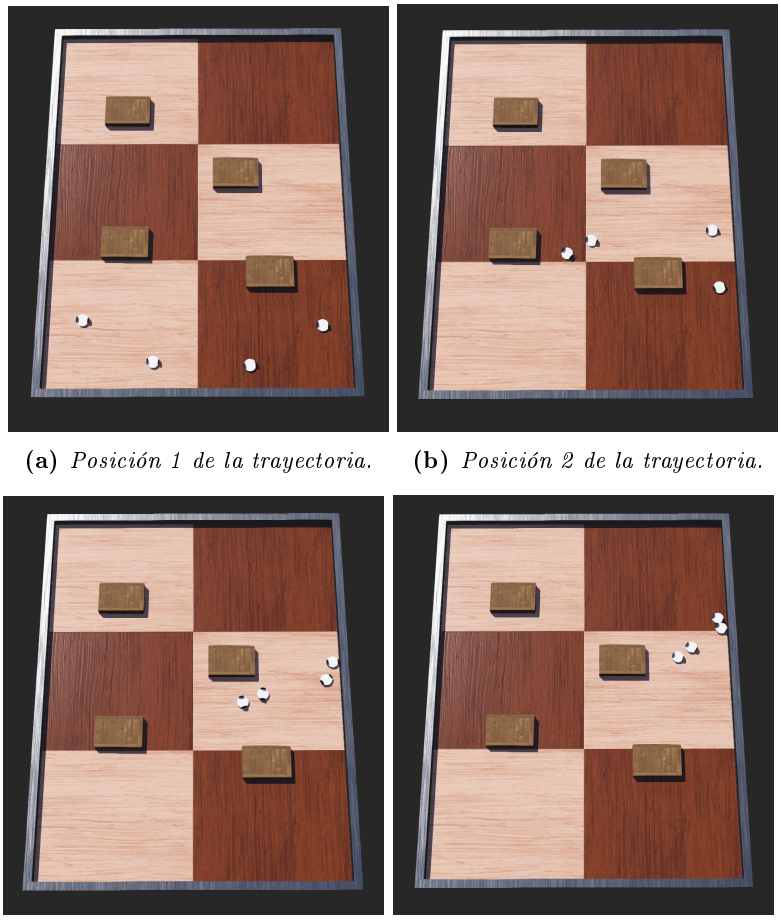
**Figura 112:** *Simulación de partículas con cuatro obstáculos en Matlab.*




Nota. Elaboración propia.

Para la quinta prueba, se incluyeron seis obstáculos rectangulares. Las posiciones de estos y la meta se presentan en el Cuadro 66. Asimismo, las posiciones iniciales de los agentes robóticos se presentan en el Cuadro 74.

**Figura 113:** *Simulación en Webots con campos potenciales artificiales y cuatro obstáculos rectangulares.*



**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

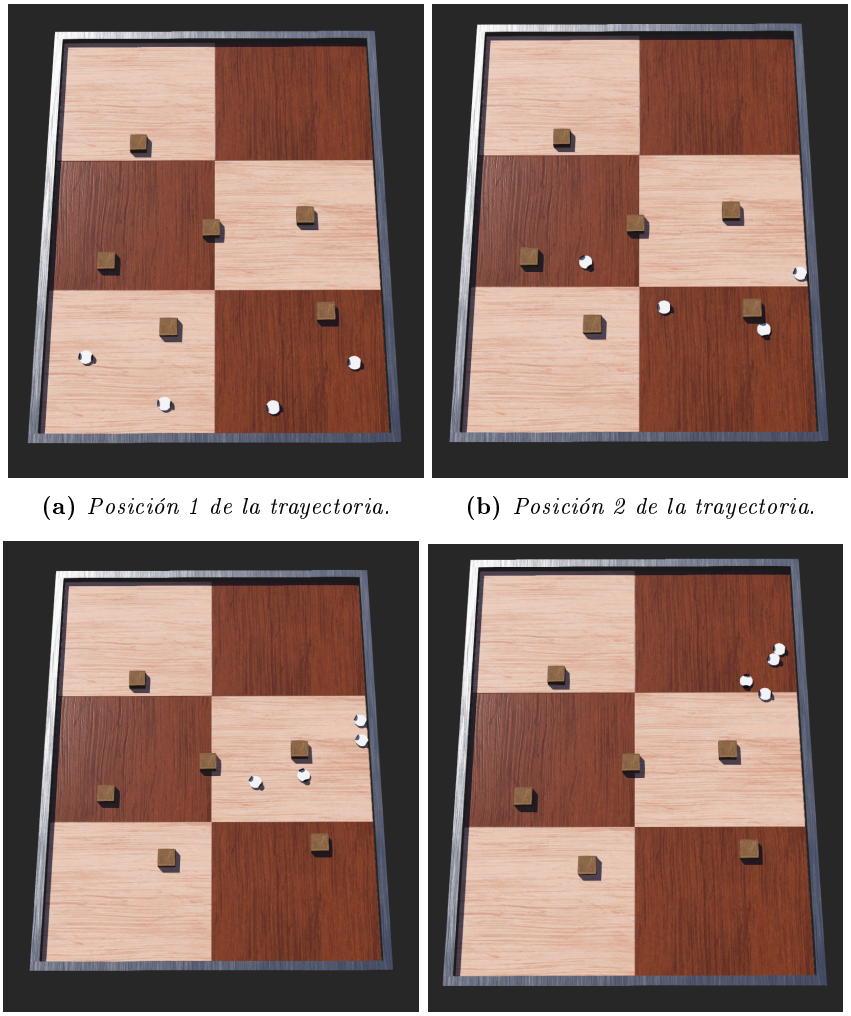
**Cuadro 74:** *Posiciones iniciales de los agentes Pololu 3pi+ para la quinta prueba con campos potenciales.*

| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (-0.7241, -2.2206) |
| 2 | (-1.6119, -1.7142) |
| 3 | (1.3737, -1.7643) |
| 4 | (0.4656, -2.2540) |

Nota. Elaboración propia.

En la Figura 114 se muestra la simulación de las partículas con los obstáculos.En la Figura 115, se observó que todos los agentes robóticos logran evadir los obstáculos y llegar a la meta.

**Figura 114:** *Simulación en Webots con campos potenciales artificiales y seis obstáculos rectangulares.*

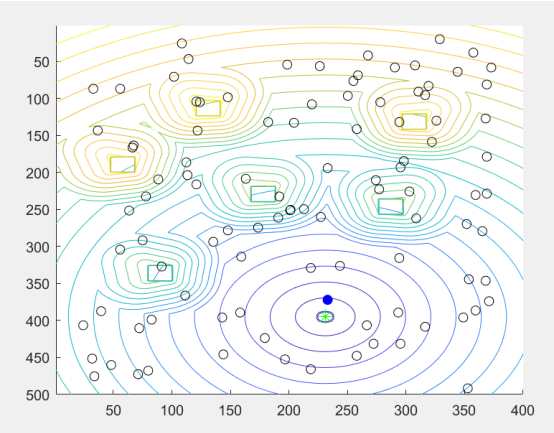


**(a)** *Posición 1 de la trayectoria.* **(b)** *Posición 2 de la trayectoria.*

**(c)** *Posición 3 de la trayectoria.* **(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

**Figura 115:** *Simulación de partículas con seis obstáculos en Matlab.*




Nota. Elaboración propia.

### 11.2.3. Implementación física del algoritmo PSO con campos potenciales artificiales

Para validar el algoritmo implementado en el simulador, se realizaron diferentes pruebas en tiempo real, es decir, el algoritmo PSO calcula las nuevas posiciones al mismo tiempo que se calculan los campos potenciales, con el objetivo de ajustar los parámetros de los campos potenciales artificiales. Estas pruebas permitieron analizar como los agentes robóticos Pololu 3pi+ lograban evadir los obstáculos y llegar a la meta. Los parámetros utilizados para cada prueba se presentan en el Cuadro 75.

**Cuadro 75:** *Parámetros de campos potenciales artificiales para pruebas en físico.*

| Descripción | Ajuste seleccionado |
|---|---|
| Ganancia atractiva (*atractive gain*) | 9 |
| Distancia campo atractivo (*dstar*) | 8 |
| Ganancia repulsiva (*repulsive gain*) | 10000 |
| Distancia campo repulsivo (*rho0*) | 45-50 |
| Factor $n$ | 2 |

Nota. Elaboración propia.

La primera prueba fue replicada utilizando un obstáculo rectangular hecho de esponja. Las posiciones del obstáculo y la meta se presentan en el Cuadro 76, mientras que las posiciones iniciales de los agentes robóticos Pololu 3pi+ se presentan en el Cuadro 77.

**Cuadro 76:** *Posiciones del obstáculo y meta para la primera prueba en el ecosistema Robotat.*

| Descripción | Posición (m) |
|---|---|
| Obstáculo 1 | (-0.0017, -0.8763) |
| Meta | (0.0053, 1.1128) |

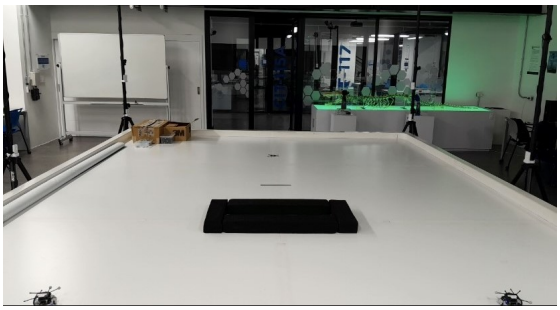
Nota. Elaboración propia.

**Cuadro 77:** *Posiciones iniciales de los agentes Pololu 3pi+ para la primera prueba física.*

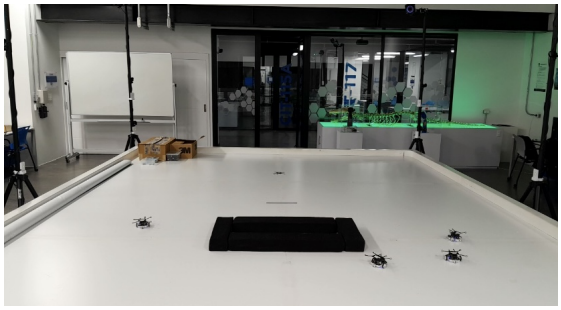
| Robot | Posiciones iniciales (m) |
|---|---|
| 2 | (0.3371, -2.2176) |
| 3 | (-0.8390, -2.3183) |
| 4 | (-1.0263, -1.8454) |
| 5 | (1.0040, -1.9183) |

Nota. Elaboración propia.

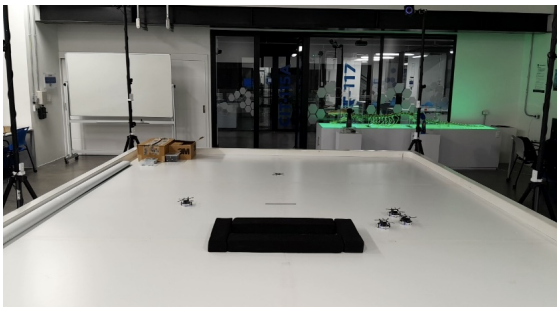
**Figura 116:** *Implementación física de campos potenciales con un obstáculo rectangular en el ecosistema robotat.*



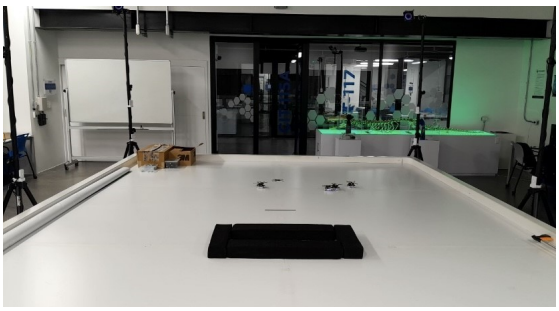
**(a)** *Posición 1 de la trayectoria.*



**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

Durante la primera prueba Figura 116, se observó que todos los agentes robóticos Pololu 3pi+ lograron evadir el obstáculo rectangular de esponja y llegar a la meta.

La segunda prueba fue replicada utilizando dos obstáculos rectangulares. Las posiciones de los obstáculos y de la meta se presentan en el Cuadro 78, mientras que las posiciones iniciales de los agentes robóticos Pololu 3pi+ se presentan en el Cuadro 79.

**Cuadro 78:** *Posiciones de los obstáculos y meta para la segunda prueba en el ecosistema Robotat.*

| **Descripción** | **Posición (m)** |
|---|---|
| Obstáculo 1 | (-1.1155, -0.7200) |
| Obstáculo 2 | (0.7301, -1.0952) |
| Meta | (0.0053, 1.1128) |

Nota. Elaboración propia.

**Cuadro 79:** *Posiciones iniciales de los agentes Pololu 3pi+ para la segunda prueba físicas.*

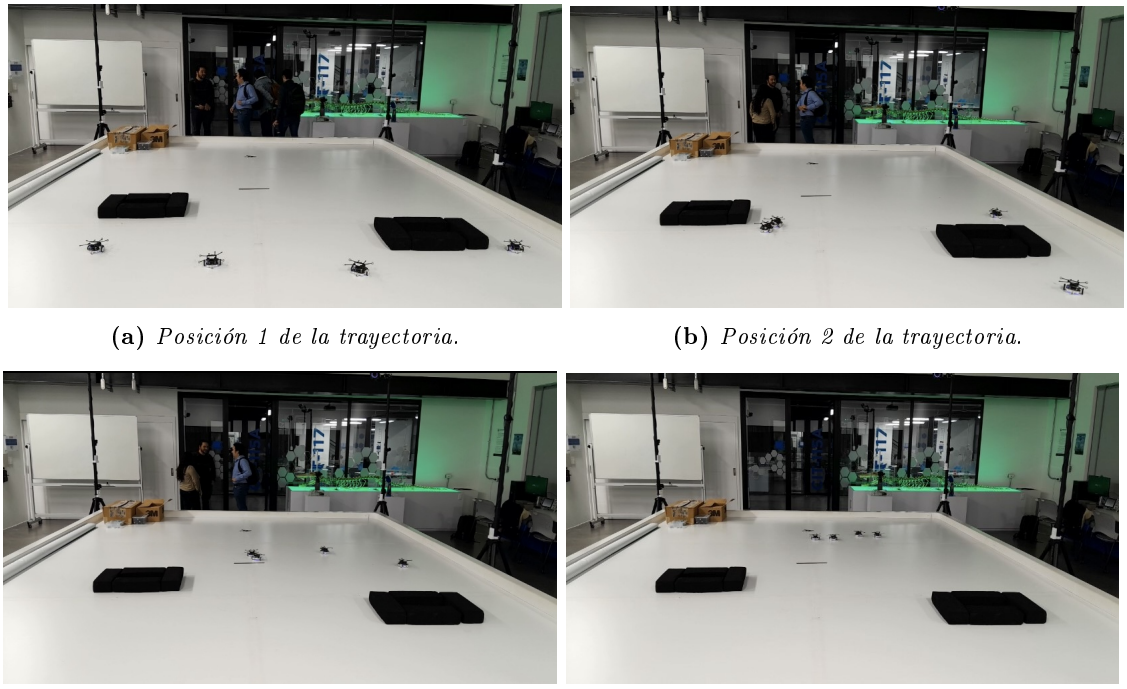
| **Robot** | **Posiciones iniciales (m)** |
|---|---|
| 2 | (-0.1482, -1.2747) |
| 3 | (-1.0819, -1.5481) |
| 4 | (1.3737, -1.7643) |
| 5 | (0.4656, -2.2540) |

Nota. Elaboración propia.

Durante la segunda prueba Figura 117, se observó que todos los agentes robóticos Pololu 3pi+ logran evadir ambos obstáculos y llegar a la meta.

La tercer prueba fue replicada utilizando tres obstáculos rectangulares. Las posiciones de los obstáculos y la meta se presentan en el Cuadro 80, mientras que las posiciones iniciales de los agentes robóticos Pololu 3pi+ se presentan en el Cuadro 81.

**Figura 117:** *Implementación física de campos potenciales con dos obstáculos rectangulares en el ecosistema robotat.*



**(a)** *Posición 1 de la trayectoria.*



**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*



**(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

**Cuadro 80:** *Posiciones de los obstáculos y meta para la tercera prueba en el ecosistema Robotat.*

| **Descripción** | **Posición (m)** |
|---|---|
| Obstáculo 1 | (-1.0846, -0.6775) |
| Obstáculo 2 | (0.7757,-1.2663) |
| Obstáculo 3 | (0.3052, 0.2337) |
| Meta | (0.0053, 1.1128) |

Nota. Elaboración propia.

**Cuadro 81:** *Posiciones iniciales de los agentes Pololu 3pi+ para la tercer prueba en físico.*

| Robot | Posiciones iniciales (m) |
|---|---|
| 1 | (-0.7241, -2.2206) |
| 2 | (-1.6119, -1.7142) |
| 3 | (1.3737, -1.7643) |
| 4 | (0.4656, -2.2540) |

Nota. Elaboración propia.

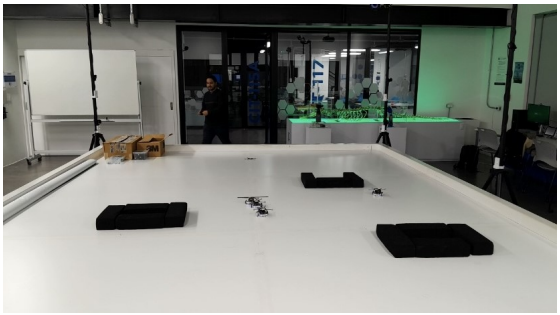
Durante la tercer prueba Figura 118, se observó que únicamente uno de los cuatro agentes robóticos Pololu 3pi+ colisionó con uno de los obstáculos.

Se lograron validar únicamente los primeros tres escenarios con el algoritmo PSO y los campos potenciales en físico, debido a falta de tiempo para utilizar el ecosistema Robotat.

**Figura 118:** *Implementación física de campos potenciales con tres obstáculos rectangulares en el ecosistema robotat.*

**(a)** *Posición 1 de la trayectoria.*

**(b)** *Posición 2 de la trayectoria.*



**(c)** *Posición 3 de la trayectoria.*

**(d)** *Posición 4 de la trayectoria.*

Nota. Elaboración propia.

# CAPÍTULO 12

## Conclusiones

- La implementación del método de vectorización con las funciones de costo y el parámetro de inercia resultó en una disminución significativa en el tiempo de convergencia en comparación al algoritmo original.
- Al aplicar el mismo método de vectorización en las funciones de costo, parámetro de inercia y controlador PID, se observó un aumento en el tiempo de convergencia respecto a la implementación original.
- La implementación del algoritmo paralelizado presentó una reducción en el tiempo de ejecución, mejorando el rendimiento del algoritmo.
- El método de paralelización demostró ser más efectivo que el de vectorización para optimizar su rendimiento.
- La migración del algoritmo al lenguaje de programación Python, junto con la implementación de paralelización, presentó una mejora significativa en el rendimiento del algoritmo.
- El algoritmo PSO, utilizado como planificador, generó trayectorias libres de obstáculos; sin embargo, no todas las convergieron cerca de la meta.
- El algoritmo PSO implementado con campos potenciales artificiales en tiempo real permitió encontrar trayectorias libres de obstáculos, aunque el proceso demoró más tiempo en comparación al planificador.
- La distancia entre los obstáculos y los agentes robóticos Pololu 3pi+ resultó ser de gran importancia a la hora de seleccionar las ganancias del campo repulsor y atractor.

La correcta elección de estos parámetros influyó en la capacidad de los agentes para converger a la meta evitando los obstáculos.

- Los parámetros del algoritmo PSO, junto con la cantidad de partículas utilizadas, fueron fundamentales en la obtención de trayectorias libres de obstáculos para el planificador.

# CAPÍTULO 13

## Recomendaciones

- Se recomienda investigar la posibilidad de implementar un protocolo de comunicación alternativo con el ecosistema Robotat con el fin de evaluar si genera mejoras en el tiempo de ejecución de las funciones.
- Se sugiere implementar el algoritmo PSO migrado a Python dentro del ecosistema Robotat para evaluar si existen mejoras en comparación al rendimiento del algoritmo MPSO original.
- Se sugiere profundizar en el estudio del método de paralelización, realizando más experimentos con el algoritmo MPSO paralelizado.
- Se recomienda implementar el algoritmo PSO con campos potenciales artificiales en tiempo real, utilizando obstáculos dinámicos para evaluar el desempeño del algoritmo.
- Se sugiere implementar el seguimiento de las trayectorias obtenidas por el planificador en el ecosistema Robotat para verificar si los agentes robóticos Pololu 3pi+ son capaces de seguir las trayectorias libres de obstáculos.

# CAPÍTULO 14

## Referencias


[1] NCYT, *Inteligencia colectiva y robótica: La maravilla de los enjambres de robots*, 2023. dirección: https://noticiasdelaciencia.com/art/48242/inteligencia-colectiva-y-robotica-la-maravilla-de-los-enjambres-de-robots#google_vignette.

[2] W. Institute, *RoboBees: Autonomous Flying Microrobots*, 2019. dirección: https://wyss.harvard.edu/technology/robobees-autonomous-flying-microrobots/.

[3] W. Institute, *Programmable Robot Swarms*, 2021. dirección: https://wyss.harvard.edu/technology/programmable-robot-swarms/.

[4] E. Castillo, *Secure and secret cooperation in robot swarms*, 2021. dirección: https://www.media.mit.edu/publications/secure-and-secret-cooperation-in-robot-swarms/.

[5] A. S. Kumar, G. Manikutty, R. R. Bhavani y M. S. Couceiro, "Search and rescue operations using robotic darwinian particle swarm optimization," en *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, págs. 1839-1843. DOI: 10.1109/ICACCI.2017.8126112.

[6] A. Salunke, C. Patil, R. Mude y R. D. Joshi, "Simultaneous Localization and Mapping (SLAM) in Swarm Robots for Map-Merging and Uniform Map Generation Using ROS," en *2023 15th International Conference on Computer and Automation Engineering (ICCAE)*, 2023, págs. 411-415. DOI: 10.1109/ICCAE56788.2023.10111365.

[7] D. W. Angelos Dimakos y S. Asif, "A Study on Centralised and Decentralised Swarm Robotics Architecture for Part Delivery System," *Academic Journal of Engineering Studies*, vol. 2, n.º 3, pág. 016 007, 2021.

[8] OptiTrack, *Primex 41*, 2023. dirección: https://optitrack.com/cameras/primex-41/.

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

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

[11] J. P. Cahueque, "Implementación de enjambre de robots en operaciones de búsqueda y rescate)," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2019.

[12] E. A. Santizo, "Aprendizaje reforzado y aprendizaje profundo en aplicaciones de robótico de enjambre," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2021.

[13] 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, 2022.

[14] J. Menéndez, "Validación de los algoritmos de robótica de enjambre Particle Swarm Optimization y Ant Colony Optimization con sistemas robóticos físicos en el ecosistema Robotat," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2023.

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

[16] A. Engelbrecht, *Computational Intelligence: An Introduction*, 2.ª ed. John Wiley & Sons, 2007.

[17] R. Arnold, K. Carey, B. Abruzzo y C. Korpela, "What is A Robot Swarm: A Definition for Swarming Robotics," en *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2019, págs. 0074-0081. DOI: 10.1109/UEMCON47517.2019.8993024.

[18] L. Garattoni y M. Birattari, "Swarm Robotics," en *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2016.

[19] M. Brambilla, E. Ferrante, M. Birattari y M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," en *Springer Science+Business Media*, 2013.

[20] D. Wang, D. Tan y L. Liu, "Particle swarm optimization algorithm: an overview," en *Soft Comput 22*, 2018. DOI: 10.1007/s00500-016-2474-6.

[21] A. P, *Computational Intelligence*, 2.ª ed. Wiley Publishing, 2007.

[22] R. Eberhart e Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," en *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, 2000, 84-88 vol.1. DOI: 10.1109/CEC.2000.870279.

[23] 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.

[24] R. King, *What You Need To Know About Mobile Robots*, 2023. dirección: https://www.rowse.co.uk/blog/post/what-you-need-to-know-about-mobile-robots.

[25] R. Siegwart e illah R, *Introduction to Autonomous Mobile Robots*, 1.ª ed. The MIT Press, 2007.

[26] S. Han, B. Choi y J. Lee, "A precise curved motion planning for a differential driving mobile robot," *Mechatronics*, vol. 18, n.º 9, págs. 486-494, 2008, ISSN: 0957-4158. DOI: https://doi.org/10.1016/j.mechatronics.2008.04.001. dirección: https://www.sciencedirect.com/science/article/pii/S0957415808000512.

[27] M. Grimble y L. Marconi, *Advanced Textbooks in Control and Signal Processing*, 1.ª ed. Springer Science+Business Media, 2008.

[28] M. Zea, *Control de robots móviles con ruedas*, 2023. dirección: C%C3%A1tedra%20en%20Universidad%20del%20Valle%20de%20Guatemala.

[29] H. Choset et al., *Principles of robot motion: Theory, Algorithms and implementations*. Ronald C. Arkin, 2005.

[30] R. Siegwart e I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Ronald C. Arkin, 2004.

[31] MathWorks, *Matlab*, 2024. dirección: https://la.mathworks.com/products/matlab.html.

[32] S. W. Zaranek, B. Chou, G. Sharma, H. Zarrinkoub y MathWorks, *Accelerating MATLAB Algorithms and Applications*, 2013. dirección: https://la.mathworks.com/company/technical-articles/accelerating-matlab-algorithms-and-applications.html#Serial.

[33] MathWorks, *Parallel Computing Toolbox*, 2024. dirección: https://la.mathworks.com/products/parallel-computing.html.

[34] A. Walilko, *What are CPU cores vs threads*, 2023. dirección: https://www.liquidweb.com/blog/difference-cpu-cores-thread/.

[35] Robotics y Electronics, *Pololu 3pi+ 32U4 User's Guide*, 2024. dirección: https://www.pololu.com/docs/0J83.