

---

# Implementación y Validación del Algoritmo de Robótica de Enjambre *Particle Swarm Optimization* en Sistemas Físicos

---

Alex Daniel Maas Esquivel



UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



Implementación y Validación del Algoritmo de Robótica de  
Enjambre *Particle Swarm Optimization* en Sistemas Físicos

Trabajo de graduación presentado por Alex Daniel Maas Esquivel para  
optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2021

Vo.Bo.:

(f) \_\_\_\_\_  
Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) \_\_\_\_\_  
Dr. Luis Alberto Rivera Estrada

(f) \_\_\_\_\_

(f) \_\_\_\_\_

Fecha de aprobación: Guatemala, de de .

---

## Prefacio

---

La elaboración de la presente tesis surge del interés personal de poder poner en práctica los conocimientos adquiridos durante estos últimos años acerca de programación, inteligencia de enjambre, robótica, sistemas de control y poder aplicarlos en temas de actualidad como lo es la robótica *swarm* o robótica de enjambre, un tema del cual aun se puede profundizar mucho más y cuenta con una cantidad ilimitada de aplicaciones.

Quiero agradecer principalmente a Dios y a mis padres por haberme apoyado desde el momento que decidí estudiar en la capital y por nunca abandonarme durante todo este proceso, a mis hermanos por su apoyo. Agradezco a la Universidad del Valle de Guatemala y a sus catedráticos que me han brindado su conocimiento, principalmente me gustaría agradecer a mi asesor de tesis el Dr. Luis Alberto Rivera por el tiempo dedicado a resolver mis dudas y compartir su conocimiento para la realización de esta tesis.

Finalmente me gustaría agradecer a todos los amigos que tuve la oportunidad de hacer durante estos 5 años, amistades que se mantendrán para toda la vida, gracias por las risas, los momentos, los viajes compartidos, etc. sin duda hicieron mi experiencia en la Universidad mucho mejor.

<b>Prefacio</b>	<b>III</b>
<b>Lista de figuras</b>	<b>VIII</b>
<b>Lista de cuadros</b>	<b>IX</b>
<b>Resumen</b>	<b>X</b>
<b>Abstract</b>	<b>XI</b>
<b>1. Introducción</b>	<b>1</b>
<b>2. Antecedentes</b>	<b>3</b>
2.1. Robotarium de Georgia Tech . . . . .	3
2.2. Megaproyecto Robotat . . . . .	4
2.3. PSO . . . . .	5
2.4. Ant Colony . . . . .	5
<b>3. Justificación</b>	<b>6</b>
<b>4. Objetivos</b>	<b>7</b>
<b>5. Alcance</b>	<b>8</b>
<b>6. Marco teórico</b>	<b>9</b>
6.1. Robótica de Enjambre . . . . .	9
6.2. Ventajas de la Robótica de enjambre . . . . .	9
6.3. <i>Particle Swarm Optimization PSO</i> . . . . .	10
6.4. Funciones de Costo o Fitness . . . . .	12
6.4.1. Función <i>Sphere</i> . . . . .	12
6.4.2. Función <i>Rosenbrock</i> . . . . .	13
6.4.3. Función <i>Himmelblau</i> . . . . .	14
6.5. Programación Multihilos . . . . .	15
6.5.1. Casos en los que conviene usar programación Multihilos . . . . .	16

6.6.	Protocolos de Comunicación a través de Internet . . . . .	17
6.6.1.	Protocolo de Control de Transmisión (TCP) . . . . .	17
6.6.2.	Protocolo de Datagramas de usuario (UDP) . . . . .	19
6.7.	Robots Móviles . . . . .	20
6.7.1.	Modelo Uniciclo . . . . .	20
6.7.2.	Modelo Diferencial . . . . .	21
6.7.3.	Difeomorfismo para la transformación de cinemática uniciclo . . . . .	21
6.8.	Controladores de Posición y Velocidad de Robots Diferenciales . . . . .	22
6.8.1.	Control proporcional de velocidades con saturación limitada . . . . .	22
6.8.2.	Control PID de velocidad lineal y angular . . . . .	23
6.8.3.	Control de pose . . . . .	23
6.8.4.	Control de pose de Lyapunov . . . . .	24
6.8.5.	Control Closed-Loop Steering . . . . .	24
6.8.6.	Control por medio de Regulador Lineal Cuadrático (LQR) . . . . .	24
6.8.7.	Controlador Lineal Cuadrático Integral (LQI) . . . . .	24
6.9.	Raspberry Pi . . . . .	25
6.9.1.	Hardware . . . . .	25
6.9.2.	Software . . . . .	25
6.9.3.	Modelos . . . . .	25
6.10.	Microcontroladores . . . . .	26
6.10.1.	Arquitectura ARM . . . . .	26
6.10.2.	Arquitectura AVR . . . . .	26
6.10.3.	Arquitectura PIC . . . . .	26
6.11.	Arduino Uno . . . . .	27
6.12.	Tiva C . . . . .	28
6.13.	Propuesta de Robots . . . . .	29
6.13.1.	Kilobot . . . . .	29
6.13.2.	E-puck . . . . .	30
6.13.3.	Pi-puck . . . . .	30
<b>7.</b>	<b>Validación de microcontrolador, sistemas embebidos, entorno de desarrollo, lenguaje de programación y robots</b>	<b>32</b>
7.1.	Evaluación . . . . .	32
7.2.	Selección del tipo de microcontrolador, plataforma u ordenador a utilizar . . . . .	34
7.2.1.	Criterios Usados . . . . .	35
7.2.2.	Peso por criterio . . . . .	35
7.2.3.	Selección del sistema físico . . . . .	38
7.3.	Lenguaje de Programación . . . . .	38
7.3.1.	Evaluación . . . . .	39
7.3.2.	Criterios Usados . . . . .	39
7.3.3.	Peso por criterio . . . . .	40
7.3.4.	Selección del lenguaje de programación . . . . .	41
<b>8.</b>	<b>Implementación del Particle Swarm Optimization</b>	<b>42</b>
8.1.	Validación del PSO . . . . .	42
8.1.1.	Función <i>Sphere</i> . . . . .	43
8.1.2.	Función <i>Himmelblau</i> . . . . .	46
8.2.	PSO enfocado a Robots Móviles . . . . .	49

8.2.1.	Creación de funciones costo . . . . .	49
8.2.2.	Creación y ejecución de hilos de programación . . . . .	50
8.2.3.	Configuración inicial del PSO . . . . .	51
8.2.4.	Obtención de posición y orientación del robot . . . . .	51
8.2.5.	Actualización del <i>local best</i> y <i>global best</i> . . . . .	51
8.2.6.	Algoritmo PSO . . . . .	53
8.2.7.	Cálculo de la velocidad lineal y velocidad angular mediante varios tipos de controladores . . . . .	53
8.2.8.	Ajuste de velocidad de los motores . . . . .	53
<b>9.</b>	<b>Validación e Implementación física del PSO</b>	<b>55</b>
9.1.	Programación Multihilos . . . . .	55
9.1.1.	Recepción del modulo de visión por computadora . . . . .	55
9.1.2.	Envío y recepción de información entre agentes . . . . .	57
9.1.3.	Combinación de ambos protocolos y programación multihilos . . . . .	61
9.2.	Pruebas del algoritmo PSO . . . . .	63
9.2.1.	Ambientes controlados . . . . .	63
9.2.2.	Mesa de Pruebas . . . . .	64
<b>10.</b>	<b>Conclusiones</b>	<b>65</b>
<b>11.</b>	<b>Recomendaciones</b>	<b>66</b>
<b>12.</b>	<b>Bibliografía</b>	<b>67</b>
<b>13.</b>	<b>Anexos</b>	<b>70</b>
13.1.	Código . . . . .	70
<b>14.</b>	<b>Glosario</b>	<b>72</b>

---

## Lista de figuras

---

1.	Mesa de Pruebas del <i>Robotarium</i> de Georgia Tech [3]. . . . .	4
2.	Ejemplo de convergencia [7] . . . . .	10
3.	Efectos de los hiperparametros sobre una partícula [6]. . . . .	12
4.	Representación de la función Sphere [10]. . . . .	13
5.	Representación de la función Rosenbrock [11]. . . . .	14
6.	Representación de la función Himmelblau [11]. . . . .	15
7.	Ejemplo de los programas o <i>main thread</i> encargados de correr los diferentes hilos [12]. . . . .	16
8.	Ejemplo de un proceso Multihilos [13]. . . . .	17
9.	Aplicación cliente-servidor usando TCP [14] . . . . .	18
10.	Aplicación cliente-servidor usando UDP [14] . . . . .	20
11.	Ajuste del difeomorfismo [17]. . . . .	22
12.	Arduino Uno [25] . . . . .	28
13.	Tiva C [26] . . . . .	28
14.	Kilobot [27]. . . . .	29
15.	E-puck [28]. . . . .	30
16.	Pi-puck [29]. . . . .	31
17.	Resultado del <i>Trade Study</i> 1 . . . . .	38
18.	Raspberry Pi 3B [18]. . . . .	38
19.	Resultado del <i>Trade Study</i> 2. . . . .	41
20.	Visual Studio Code [37] . . . . .	41
21.	PSO con 10 agentes Inercia Constante . . . . .	43
(a).	Matlab . . . . .	43
(b).	Raspberry Pi . . . . .	43
22.	PSO con 50 agentes Inercia Constante . . . . .	43
(a).	Matlab . . . . .	43
(b).	Raspberry Pi . . . . .	43
23.	PSO con 100 agentes Inercia Constante . . . . .	44
(a).	Matlab . . . . .	44
(b).	Raspberry Pi . . . . .	44

24.	PSO con 10 agentes Inercia Caótica . . . . .	45
(a).	Matlab . . . . .	45
(b).	Raspberry Pi . . . . .	45
25.	PSO con 50 agentes Inercia Caótica . . . . .	45
(a).	Matlab . . . . .	45
(b).	Raspberry Pi . . . . .	45
26.	PSO con 100 agentes Inercia Caótica . . . . .	46
(a).	Matlab . . . . .	46
(b).	Raspberry Pi . . . . .	46
27.	PSO con 10 agentes Inercia Constante . . . . .	46
(a).	Matlab . . . . .	46
(b).	Raspberry Pi . . . . .	46
28.	PSO con 50 agentes Inercia Constante . . . . .	47
(a).	Matlab . . . . .	47
(b).	Raspberry Pi . . . . .	47
29.	PSO con 100 agentes Inercia Constante . . . . .	47
(a).	Matlab . . . . .	47
(b).	Raspberry Pi . . . . .	47
30.	PSO con 10 agentes Inercia Caótica . . . . .	48
(a).	Matlab . . . . .	48
(b).	Raspberry Pi . . . . .	48
31.	PSO con 50 agentes Inercia Caótica . . . . .	48
(a).	Matlab . . . . .	48
(b).	Raspberry Pi . . . . .	48
32.	PSO con 100 agentes Inercia Caótica . . . . .	49
(a).	Matlab . . . . .	49
(b).	Raspberry Pi . . . . .	49
33.	Ejecución de Hilos de Programación. . . . .	50
34.	Actualización del local y global best. . . . .	52
35.	Proceso de truncar velocidades máximas. . . . .	54
36.	Prueba 1 de la comunicación Matlab con C. . . . .	56
37.	Prueba 2 de la comunicación Matlab con C. . . . .	57
38.	Representación del envío y recepción de datos entre agentes. . . . .	58
39.	Diagrama de envío y recepción de información entre agentes. . . . .	59
40.	Pruebas del envío y recepción de datos 1. . . . .	60
41.	Pruebas del envío y recepción de datos 2. . . . .	61
42.	Poses detectadas de 4 agentes. . . . .	62
43.	Validación de programación multihilos. . . . .	62

---

## Lista de cuadros

---

1.	Características del PIC16F877A . . . . .	33
2.	Características del PIC24F16KM202 . . . . .	33
3.	Características del PIC32MZ0512EFE064 . . . . .	33
4.	Características del Arduino Uno . . . . .	34
5.	Características de la Tiva C . . . . .	34
6.	Comparación de modelos Raspberry Pi . . . . .	34
7.	Adaptabilidad a Robots móviles . . . . .	36
8.	Disponibilidad de dispositivos . . . . .	36
9.	Costo por dispositivos . . . . .	36
10.	Capacidad de memoria por dispositivos . . . . .	37
11.	Frecuencia de operación por dispositivos . . . . .	37
12.	Previo Uso de dispositivos . . . . .	37
13.	Disponibilidad de Librerías . . . . .	40
14.	Previo Uso de lenguajes de programación . . . . .	40
15.	Adaptabilidad de lenguajes de programación . . . . .	41
16.	Resultados de Comparacion Matlab-RaspberryPi Sphere W1 . . . . .	44
17.	Resultados de Comparacion Matlab-RaspberryPi Sphere W2 . . . . .	46
18.	Resultados de Comparación Matlab-RaspberryPi Himmelblau W1 . . . . .	47
19.	Resultados de Comparación Matlab-RaspberryPi Himmelblau W2 . . . . .	49
20.	Validación de envío y recepción . . . . .	60
21.	Primera validación del algoritmo, prueba 5 agentes 7 primeras iteraciones . . .	64
22.	Primera validación del algoritmo, prueba 5 agentes 7 iteraciones finales . . .	64

---

## Resumen

---

La robótica de enjambre emplea patrones de comportamiento colectivo mediante interacciones entre robots y robots con su entorno con el fin de alcanzar objetivos o metas establecidas. Uno de los algoritmos de robótica de enjambre comúnmente usado es el *Particle Swarm Optimization* (PSO), el cual es utilizado como planificador de trayectorias.

Se usó el *Modified Particle Swarm Optimization* (MPSO) desarrollado en la fase II [1] en combinación con los hiperparámetros obtenidos gracias al diseño de una red neuronal recurrente III [2]. Ambos trabajos se realizaron a nivel de simulación por lo que se evalúan distintos tipos de controladores, plataformas, ordenadores, lenguajes de programación y robots móviles para una correcta implementación del PSO en un sistema físico. Se usó la herramienta de *trade study* para seleccionar el ordenador *Raspberry Pi* y el lenguaje de programación C y Python para esta implementación.

Al no contar con una plataforma móvil operativa se desarrollan distintas técnicas para lograr la validación del algoritmo bajo ambientes controlados. Se desarrollaron dos sistemas de comunicación, el primero que permita la obtención de la pose de cada agente en la mesa de pruebas de la UVG en tiempo real usando un algoritmo de visión por computadora desarrollado como otro proyecto de tesis; el segundo que permita el intercambio de información valiosa entre agentes que ayudan al propio algoritmo PSO.

Para ambas implementaciones se usó programación multihilos de forma que se la recepción de información se realice sin afectar la ejecución en paralelo del algoritmo PSO. Para los dos sistemas de comunicación usan un protocolo UDP usando una red local.

---

## Abstract

---

Swarm robotics employs patterns of collective behavior through interactions between robots and robots with their environment in order to achieve established objectives or goals. One of the most commonly used swarm robotics algorithms is Particle Swarm Optimization (PSO), which is used as a trajectory planner.

Modified Particle Swarm Optimization (MPSO) developed in phase II [1] was used in combination with the hyperparameters obtained of the application of neural networks in phase III [2]. Both works were carried out at the simulation level, so different types of controllers, platforms, computers, programming languages and mobile robots were evaluated for a correct implementation of the PSO in a physical system. The detrade study tool was used to select the Raspberry Pi computer and the C programming language for this implementation.

As there is no mobile platform, a communication system between agents was developed using a local network. Through a GPS module, the coordinates of each agent are obtained in real time. For both strategies, multithreaded programming is used so that both processes can be executed at the same time as the PSO.

# CAPÍTULO 1

---

## Introducción

---

El algoritmo de robótica de enjambre *Modified Particle Swarm Optimization* (el cual sera llamado MPSO en esta tesis) desarrollado en fases previas a este proyecto funciona bastante bien a nivel de simulación, donde se empleó el software webots. En este trabajo se presenta la implementación del algoritmo en un sistema físico así como pruebas de validación realizadas en una mesa de pruebas. Se busca replicar los resultados obtenidos a nivel de simulación de fases anteriores mientras el algoritmo es ejecutado en un sistema físico.

Por medio de la herramienta *Trade Study* se evaluó la mejor alternativa de microcontrolador, plataforma u ordenador y el tipo de lenguaje de programación a utilizar para realizar la implementación del algoritmo PSO en un sistema físico. Se integran los controladores desarrollados para el MPSO desarrollado en la fase [1] para tener una implementación mas apegada a la realidad, donde en un futuro se tenga una plataforma móvil completada para realizar mas pruebas. Para validar el sistema físico seleccionado se realiza la implementación del algoritmo PSO y se comparan los resultados obtenidos contra la aplicación del PSO desarrollada matlab de fases anteriores [1].

Se implemento programación multihilos para la ejecución de un proceso de comunicación entre agentes para intercambiar información útil para el algoritmo además de otro proceso de comunicación entre cada agente con la mesa de pruebas de la UVG. Gracias a un algoritmo de visión por computadora (el cual fue desarrollado como otro proyecto de tesis) se obtiene la pose de cada agente, estos datos son enviados a cada agente para tener una validación mas apegada a la realidad.

Se uso un protocolo de comunicación UDP para ambos procesos de comunicación, los cuales fueron validados por separado y en conjunto. Para la validación del algoritmo PSO primero se replican los resultados obtenidos en la fase [1], para esto se tomo la misma cantidad de agentes y las mismas condiciones iniciales buscando así obtener los mismos resultados, posición final y velocidad enviada a ambos motores, al momento de finalizar el algoritmo PSO el cual se ejecuta en cada dispositivo *Raspberry Pi*.

Buscando una validación aun mas contundente y pensando en futuras fases de este

proyecto se valido el algoritmo PSO en conjunto con el algoritmo de visión por computadora. Para esto se colocaron varios marcadores (los cuales simulan agentes) en la mesa de pruebas

# CAPÍTULO 2

---

## Antecedentes

---

### 2.1. Robotarium de Georgia Tech

El proyecto de *Robotarium* proporciona una plataforma de investigación robótica de enjambre de acceso remoto. En esta se permite a personas de todo el mundo hacer diversas pruebas con sus algoritmos de robótica de enjambre a sus propios robots con fines de apoyar la investigación y seguir buscando formas de mejorar las trayectorias definidas. Para utilizar la plataforma se debe descargar el simulador que se encuentra en la página del *Robotarium* el cual presenta la opción de descargarlo en Matlab o Python. Como siguiente paso se debe registrar en la página del *Robotarium* y esperar a ser aprobado para crear el experimento, puede tomar 2-3 días [3].

Se presenta una base plana con bordes que limitan el movimiento de los robots a utilizar estas cumplen la función de fronteras del espacio de búsqueda. La base es de un color blanco para facilitar el procesamiento de imágenes y se cuenta con una cámara ubicada en el techo apuntando a toda la base, con esta lograr capturar las posiciones de los diferentes robots de pruebas [3].



Figura 1: Mesa de Pruebas del *Robotarium* de Georgia Tech [3].

En el departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala se tuvo la iniciativa de investigar, aprender y trabajar con la robótica de enjambre (*Swarm Robotics*), gracias a esto surgieron las fases I, II y II del conocido “Megaproyecto Robotat”.

## 2.2. Megaproyecto Robotat

En la fase I se procedió con el diseño y la elaboración de una plataforma para el Robotat donde se implementó un algoritmo de visión de computadora para poder obtener la posición y orientación del robot desde una perspectiva planar (2 dimensiones). Posteriormente en la fase II desarrollada por Aldo Aguilar [1] se tomó el modelo estándar y existen del PSO y se procedió a modificarlo para que este tome en consideración las dimensiones de robots físicos y la velocidad a la que estos pueden moverse, además se implementaron diferentes tipos de controladores para buscar el robot pueda llegar al punto de meta realizando una trayectoria suave y controlada.

Se realizaron distintas pruebas buscando encontrar el mejor conjunto de hiperparámetros y probando distintas funciones objetivo, todo esto para hacer un algoritmo mucho más completo a nivel de simulación haciendo uso del programa Webots.

### 2.3. PSO

En la tercera fase, desarrollada por Eduardo Santizo [2] se mejoró el desempeño del algoritmo de optimización de partículas PSO haciendo uso del método denominado *PSO Tunner*, el cual consiste de una red neuronal recurrente la cual es capaz de tomar diferentes métricas de las partículas PSO y tornarlas a través de su procesamiento por medio de una red LSTM, GRU o BiLSTM y busca realizar una predicción de los hiper parámetros que debería emplear el algoritmo. Esta predicción es de carácter dinámico, lo que hace que en cada iteración se generan las métricas que describen al enjambre y use estos resultados para la siguiente iteración buscando así reducir el tiempo de convergencia y susceptibilidad a mínimos locales del PSO original [2].

En la misma fase se elaboró una alternativa al algoritmo de navegación alrededor de un ambiente conocido por medio de programación dinámica basándose en el ejemplo de programación dinámica *Gridworld*. Este nos presenta el caso donde un agente se mueve dentro de un espacio de estados representado en la forma de una cuadrícula y únicamente puede realizar 4 movimientos: Moverse hacia arriba, abajo, izquierda o derecha. De acuerdo a su estado actual y último movimiento es capaz de moverse a un nuevo estado y obtiene una recompensa, el agente busca obtener el máximo de recompensas posibles generando así una ruta óptima desde cada estado hasta la meta [2].

Finalmente, estos avances fueron compactados en un grupo de herramientas llamado *Swarm Robotics Toolbox* para ser usado en futuras fases. Debido a la pandemia ocasionada por el COVID-19, esta nueva propuesta de algoritmo únicamente fue probada a nivel de simulación, haciendo uso del programa Webots para las diversas pruebas realizadas con el robot E-puck.

### 2.4. Ant Colony

En otra tesis, desarrollada por Gabriela Iriarte, se buscó crear una alternativa al MPSO para su futuro uso en los *Bitbots* de la UVG, desarrollando así el algoritmo de planificador de trayectorias *Ant Colony*. Se implementó el algoritmo Simple *Ant Colony*, y posteriormente el algoritmo *Ant System* [4].

Se emplearon diversos controladores y se modificó el camino encontrado interpolándolo para crear más metas y lograr una trayectoria suave. Los parámetros encontrados se validaron por medio de simulaciones computarizadas permitiendo visualizar el comportamiento de colonia y adaptar los modelos del movimiento y de la cinemática a los Bitbots [4]. Lamentablemente por la pandemia ocasionada por el COVID-19 igual que el caso del PSO esta nueva propuesta de algoritmo se vio limitada a pruebas únicamente en simulación por medio del programa Webots.

# CAPÍTULO 3

---

## Justificación

---

La parte primordial de cada nueva propuesta de algoritmo es su correcta validación en una aplicación y ambiente real. La Universidad del Valle de Guatemala y su departamento de Ingeniería Macatrónica, Electrónica y Biomédica han buscado introducirse al mundo de la robótica de enjambre y poder desarrollar su propio laboratorio enfocado tanto para la enseñanza como para la investigación.

El enfoque principal de esta tesis, es dar un paso más hacia la creación de este laboratorio, tomando conocimientos ya existentes de fases anteriores y poder darles un cierre adecuado cumpliendo todos los objetivos previos propuestos para los algoritmos de robótica de enjambre pero aplicados en un ambiente real.

Para esta tesis se busca poder realizar la correcta migración del algoritmo MPSO el cual actualmente se encuentra en fase de simulación (Webots). Se busca por medio de diferentes pruebas poder verificar el correcto funcionamiento de este y validar la correcta migración del algoritmo hacia una plataforma y entorno real. Se buscará su futuro uso en robots físicos comerciales o desarrollados por el mismo departamento y poder usarlos en la mesa de pruebas ya existente en la Universidad del Valle de Guatemala.

# CAPÍTULO 4

---

## Objetivos

---

### Objetivo General

Implementar y validar el algoritmo de robótica de enjambre *Particle Swarm Optimization* desarrollado en años anteriores, a nivel de simulación, en sistemas físicos.

### Objetivos Específicos

- Evaluar distintas opciones de microcontroladores, sistemas embebidos, lenguajes de programación, entornos de desarrollo y robots móviles, para seleccionar los más adecuados para su uso en aplicaciones de robótica de enjambre, utilizando específicamente el PSO.
- Migrar el algoritmo de robótica de enjambre PSO hacia el microcontrolador del sistema físico seleccionado.
- Validar la migración del algoritmo de robótica de enjambre y verificar el desempeño de los sistemas físicos mediante pruebas simples en ambientes controlados.

## CAPÍTULO 5

---

### Alcance

---

El alcance de este trabajo de graduación consiste en la implementación y validación del algoritmo de robótica de enjambre *Particle Swarm Optimization* en sistemas físicos, para lo cual se toma el modelo PSO modificado [1] para la implementación de este algoritmo en un sistema físico. Se opta por realizar la implementación de este algoritmo haciendo uso del ordenador *Raspberry Pi* que entre sus muchas ventajas destaca su afinidad hacia los robots móviles.

Como consideraciones para el uso del algoritmo PSO se tiene la recepción y envío de información a otros agentes, la capacidad de obtener la posición actual de cada agente en tiempo real, conocer las dimensiones del robot móvil y contar con una cantidad adecuada de agentes para poder validar el correcto funcionamiento del algoritmo.

Por limitaciones de la pandemia COVID-19 y la falta de una plataforma móvil integrada con módulos de comunicación, llantas y sistema GPS, se plantean diversas técnicas adicionales para cumplir con estos requerimientos para demostrar el funcionamiento del algoritmo PSO. En futuras fases del proyecto se pueden implementar el código desarrollado, modulo GPS y sistema de comunicación en una plataforma móvil para observar su respuesta en la mesa de pruebas de la UVG.

# CAPÍTULO 6

---

Marco teórico

---

## 6.1. Robótica de Enjambre

La robótica de enjambre (*swarm robotics*) es una nueva aproximación a la coordinación de un gran número de robots relativamente simples. De forma que estos mismos robots sean capaces de llevar a cabo tareas colectivas que están fuera de las capacidades de un único robot [5].

Se supone que un comportamiento colectivo deseado surge de las interacciones entre los robots y las interacciones de los robots con el entorno. Este enfoque surgió en el campo de la inteligencia artificial de enjambres, así como en los estudios biológicos de insectos, hormigas y otros campos en la naturaleza [5].

Este campo de investigación tiene justamente su inspiración en el comportamiento observado en los insectos sociales, en los que se destacan, las hormigas, termitas, abejas o avispas; los cuales son ejemplos de cómo un gran número de individuos simples pueden interactuar para crear sistemas inteligentes colectivos [5].

## 6.2. Ventajas de la Robótica de enjambre

Los sistemas robóticos de enjambre son tolerantes a fallos y robustos, ya que pueden continuar con su misión ante el fallo de alguna unidad. Aprovechan al máximo el paralelismo ya que el conjunto de robots ejecuta más rápido cualquier tarea que un único robot, descomponiendo la tarea en subtareas y ejecútenlas de manera concurrente. Además que estos robots son bastante más baratos y el coste de cualquier reparación también es bastante menor en comparación con cualquier gran robot [5].

### 6.3. Particle Swarm Optimization PSO

El algoritmo de Optimización del enjambre de partículas (PSO) fue creado por el Dr. Russell Eberhart y el Dr. James Kennedy en el año de 1995. Este tiene origen la simulación de comportamientos sociales utilizando herramientas e ideas tomadas por gráficos computarizados e investigación sobre psicología social; además de una clara inspiración en el comportamiento social de las crías de aves y la educación de los peces [6].

El algoritmo PSO pertenece a las técnicas denominadas optimización inteligente y se clasifica como un algoritmo estocástico de optimización basado en población. A esta clasificación igualmente pertenecen los Algoritmos Genéticos (AG). PSO es intrínsecamente paralelo. La mayoría de algoritmos clásicos operan secuencialmente y pueden explorar el espacio de solución solamente en una dirección a la vez [6].

Este algoritmo es iniciado y simula un grupo aleatorio de partículas a las cuales se les asigna una posición y velocidad inicial, a estas partículas se les conoce como "soluciones", para luego proceder a actualizar las generaciones de estas para encontrar la solución óptima. En cada iteración cada partícula es actualizada por los siguientes 2 mejores resultados [7].

El primero de estos se le conoce como la mejor solución lograda hasta ahora por cada partícula y recibe el nombre de *local best*. El segundo mejor valor es rastreado por el PSO y este proceso se repite hasta que se cumpla con un número de iteraciones específica o se logre la convergencia del algoritmo. La convergencia se alcanza cuando todas las partículas son atraídas a la partícula con la mejor solución la cual recibe el nombre de *global best* [7]. La Figura 2 representa una exemplificación de como los varios *local best* deben de converger a un único *global best*.

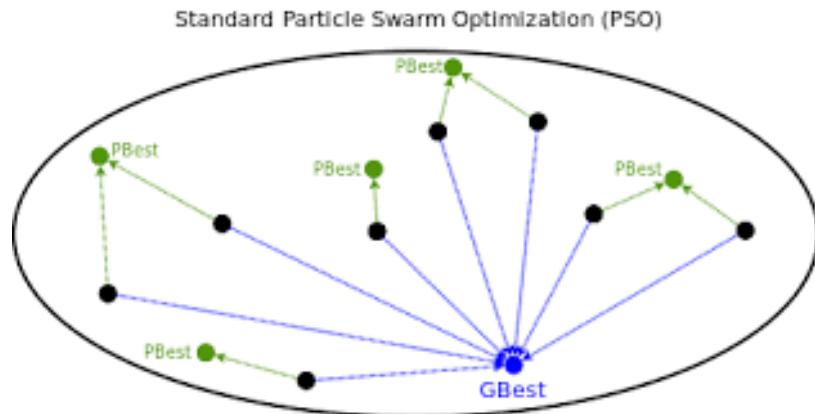


Figura 2: Ejemplo de convergencia [7]

Con estos datos es posible calcular los dos primeros factores principales de la ecuación del PSO para cada una de las partículas, los cuales son:

- Factor Cognitivo:  $P_{local} - P_i$
- Factor Social:  $P_{global} - P_i$

Donde  $P_i$  representa cada una de las soluciones/posiciones actuales. Para verificar que tan buena es la posición actual para cada partícula se usa la denominada función costo o *funcion fitness* que se da por  $f(P)$  [1], el valor escalar que genera como resultado se le denomina costo y el objetivo de las partículas es encontrar un conjunto de coordenadas que generen el valor de costo más pequeño posible dentro de una región dada.

Se toma en cuenta una diversidad de factores, entre ellos, el factor de escalamiento el cual consiste en hacer que las partículas tengan una mayor área de búsqueda o hacer que se concentren en un área más reducida, definidas como  $C1$  y  $C2$ . El factor de uniformidad corresponde a dos números aleatorios que van entre 0 y 1; se definen como  $r1$  y  $r2$ .

El factor de restricción  $\varphi$  el cual se encarga de controlar la longitud de los pasos que cada partícula puede dar en cada iteración [8]. Este parámetro es calculado de la forma:

$$\varphi = \frac{2}{2 - (C1 + C2) - \sqrt{(C1 + C2)^2 - 4(C1 + C2)}} \quad (1)$$

El factor de inercia  $w$  el cual se encarga de controlar cuanta memoria puede almacenar cada partícula y de acuerdo a [9] se tiene varias ecuaciones para calcular este parámetro, entre ellas:

**Inercia constante:**

$$0.8 < w < 1.2 \quad (2)$$

**Inercia Linear Decreciente:**

$$w = w_{max} - (w_{max} - w_{min}) * \frac{iter}{MAX_{iter}} \quad (3)$$

**Inercia Caótica:**

$$Z_i \in [0, \dots 1] \quad (4)$$

$$Z_{i+1} = 4 * Z_i * (1 - Z_i) \quad (5)$$

$$w = (w_{max} - w_{min}) * \frac{MAX_{iter} - iter}{MAX_{iter}} * w_{min} * Z_{i+1} \quad (6)$$

**Inercia Random:**

$$rand() \in [0, \dots 1] \quad (7)$$

$$w = 0.5 + \frac{rand()}{2} \quad (8)$$

**Inercia Exponencial:**

$$w = w_{min} + (w_{max} - w_{min}) * e^{\frac{-t}{\frac{MAX_{iter}}{10}}} \quad (9)$$

De forma que podemos armar la ecuación de velocidad brindada por el PSO.

$$V_{i+1} = \varphi[wV_i + C1r1(P_{local} - P_i) + C2r2(P_{global} - P_i)] \quad (10)$$

Donde  $V_i$  representa la velocidad actual de la partícula y  $V_{i+1}$  la nueva velocidad calculada para la partícula. Una vez actualizadas las velocidades de todas las partículas, se calcula las posiciones de cada una de estas:

$$X_{i+1} = X_i + V_{i+1} * \Delta t \quad (11)$$

Donde  $X_i$  representa la posición actual de la partícula y  $X_{i+1}$  la nueva posición calculada para la partícula,  $\Delta t$  es el tiempo que le toma al algoritmo realizar cada iteración. En la Figura 3 vemos un ejemplo de los diferentes parámetros y como estos tienen efectos sobre cada una de las partículas presentes en el algoritmo.

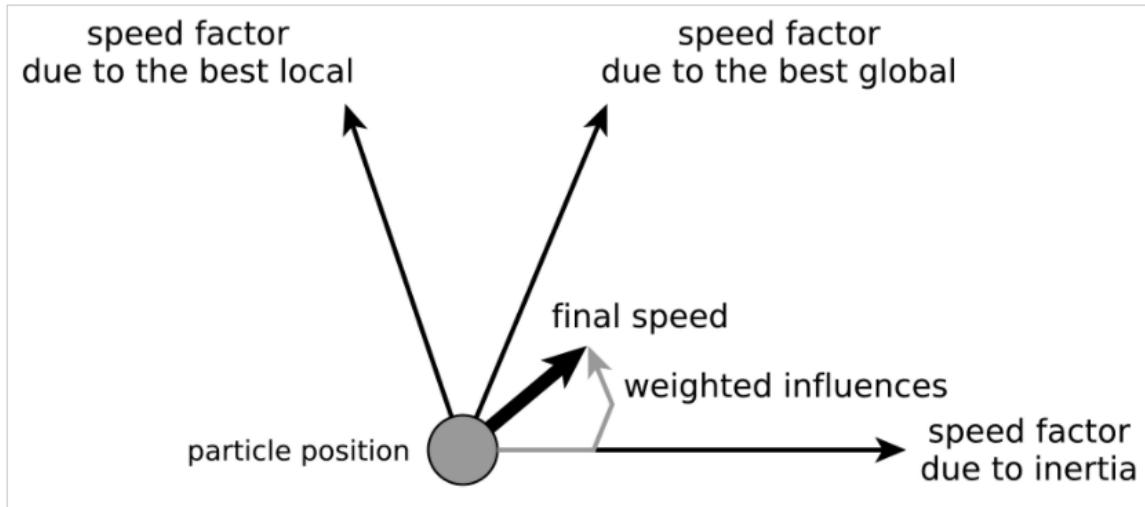


Figura 3: Efectos de los hiperparametros sobre una partícula [6].

## 6.4. Funciones de Costo o Fitness

Para verificar el correcto funcionamiento del algoritmo PSO se utilizan varias funciones las cuales contiene múltiples mínimos y máximos, al ser tan usadas para validación de este algoritmo se les conoce como *benchmark functions* o *Fitness functions*, en esta tesis se hará mención a ellas como funciones costo. Al usar estas funciones se tiene una mejor idea si los parámetros usados en el algoritmos PSO son correctos o no, dentro de las evaluaciones tenemos, velocidad de convergencia, capacidad de identificar mínimos o máximos, si la función presenta varios mínimos locales el algoritmos debe ser capaz de identificarlos y converger en el mínimo global.

### 6.4.1. Función *Sphere*

Es una de las funciones mas usadas, en su forma bidimensional es continua y convexa, no posee mínimos locales únicamente un minino global ubicado en  $[0,0]$  [10].

$$f(x) = \sum_{i=1}^N x_i^2 \quad (12)$$

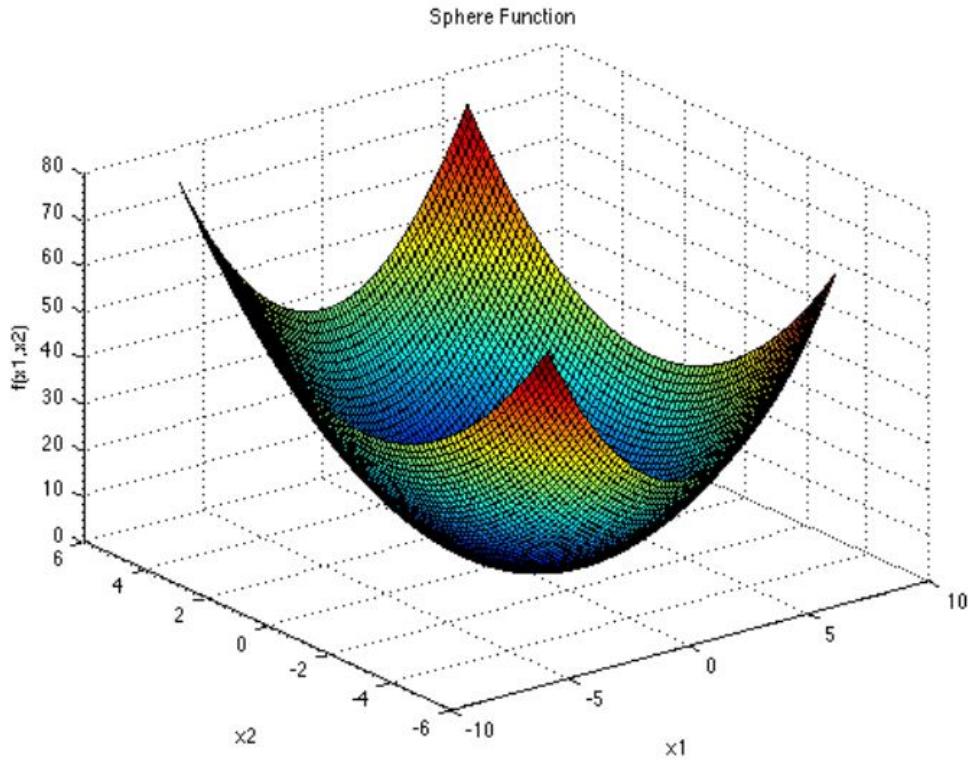


Figura 4: Representación de la función Sphere [10].

#### 6.4.2. Función *Rosenbrock*

Este tipo de función en su forma bidimensional es continua y posee un único mínimo global ubicado en  $[1,1]$  [11].

$$f(x) = \sum_{i=1}^{N-1} (1 - x_i)^2 + 100(x_{i+1} - x_i^2) \quad (13)$$

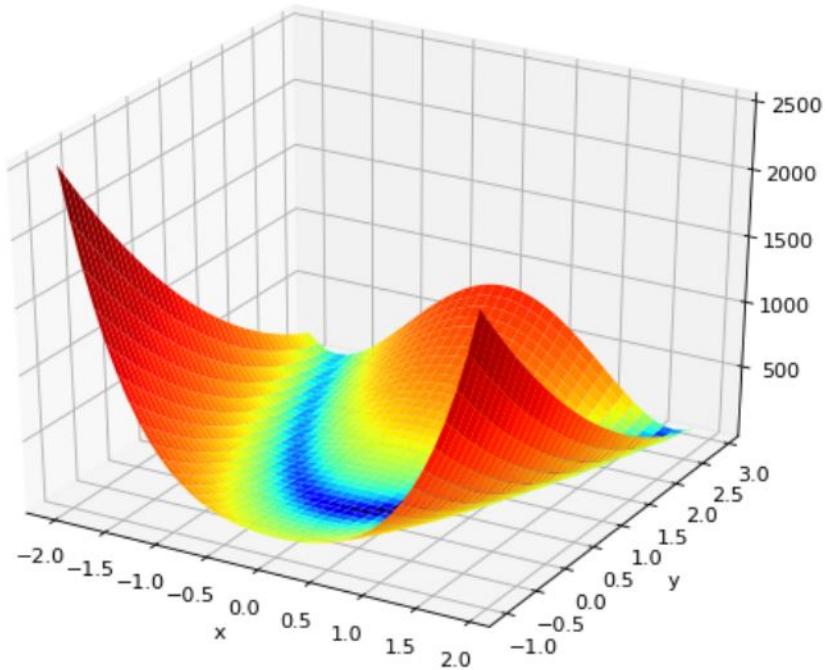


Figura 5: Representación de la función Rosenbrock [11].

#### 6.4.3. Función *Himmelblau*

Este tipo de función se caracteriza por tener múltiples mínimos del mismo valor, este hace que el algoritmo detecte múltiples mínimos globales. Es útil para determinar que tanto afecta la posición inicial de las partículas o agentes [11].

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \quad (14)$$

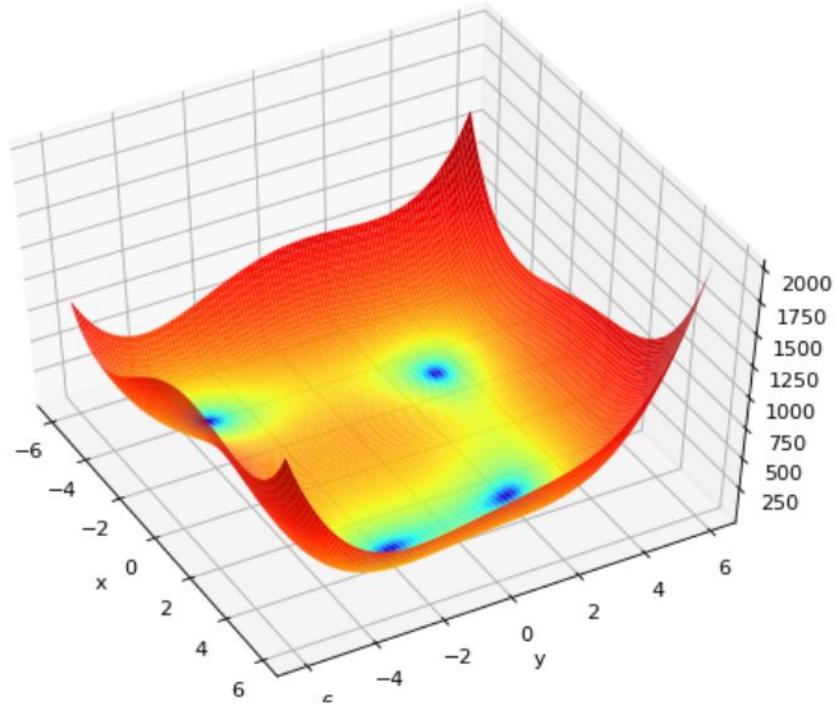


Figura 6: Representación de la función Himmelblau [11].

## 6.5. Programación Multihilos

En la programación Multihilos se hace referencia a los lenguajes de programación que permiten la ejecución de varias tareas de forma simultánea. Los hilos o *threads*, son pequeños procesos o piezas independientes de un gran proceso; de igual forma se puede decir, que un hilo es un flujo único de ejecución dentro de un proceso [12].

Un hilo no puede correr por si mismo, se ejecuta dentro de un programa, ya que requieren la supervisión de un proceso padre o un *main thread* para correr. Se pueden programar múltiples hilos de ejecución para que se corran simultáneamente en el mismo programa [12].

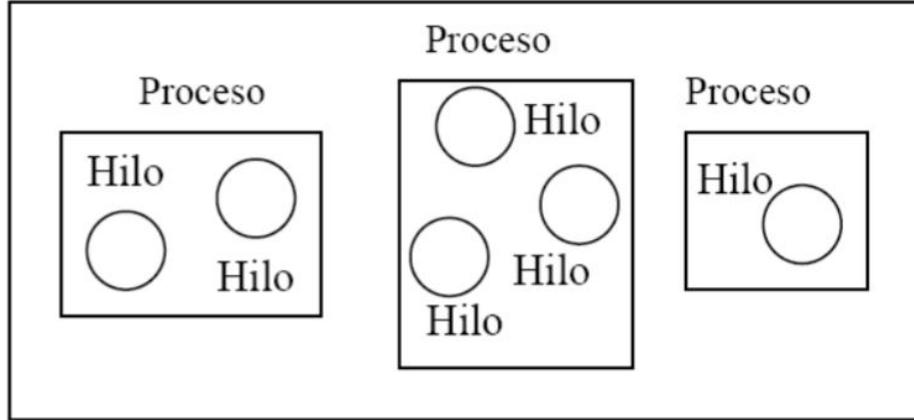


Figura 7: Ejemplo de los programas o *main thread* encargados de correr los diferentes hilos [12].

### 6.5.1. Casos en los que conviene usar programación Multihilos

El hecho de usar programación multihilos no siempre implica que vamos a tener una mejora de rendimiento y es decisión del programador cuándo conviene usar este tipo de programación. En términos generales, si se tiene una aplicación muy simple no tiene sentido plantearse este tipo de programación [13].

Es común el uso de programación multihilos cuando se desea realizar diversas tareas claramente diferenciadas con un alto coste computacional, se debe tener en cuenta que el resultado de las tareas que se ejecuten en diferentes hilos no deben depender del resultado de otras tareas ya que esto crea una dependencia que haría el proceso menos eficiente. El caso más común es cuando se prevea que pueda haber tareas retenidas o bloqueadas por estar esperando algún tipo de señal de activación [13].

Otra ventaja importante es el ahorro de tiempo al usar programación multihilos, ya que se tendrán diferentes programas ejecutándose a la vez y no se tendrá que esperar a que uno se termine para comenzar el siguiente proceso, como normalmente ocurre. Como se puede ver en la Figura 8 se tiene un *Main Thread* del cual se crean 4 *Thread* o 4 hilos cada uno ejecutando un programa diferente del otro, se ilustra el tiempo de ejecución de cada hilo y el programa principal tarda 20 segundos, siendo este el tiempo del *Thread 3*, el más largo, por lo que se tiene un ahorro de tiempo importante en comparación a ejecutar los 3 hilos de forma secuencial.

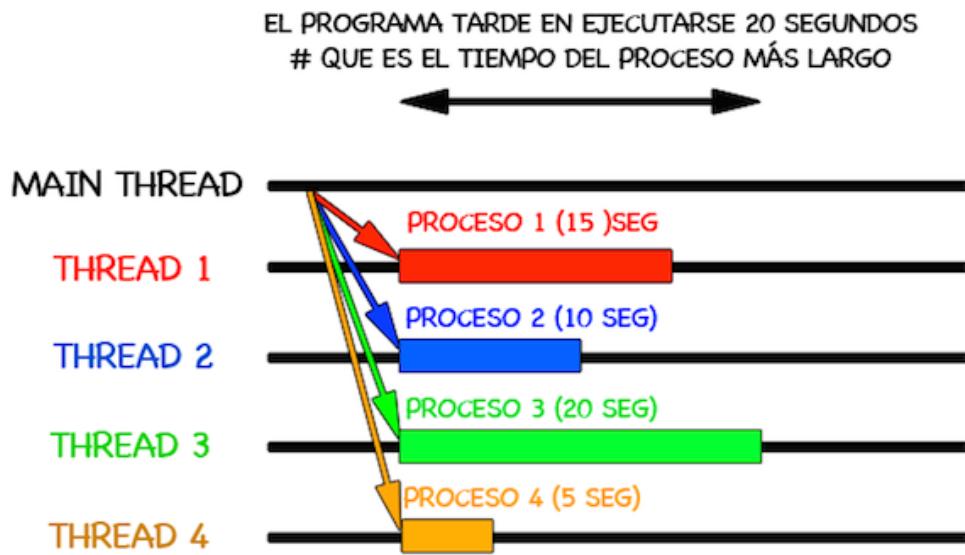


Figura 8: Ejemplo de un proceso Multihilos [13].

## 6.6. Protocolos de Comunicación a través de Internet

Internet (y, de forma más general, las redes TCP/IP) pone a disposición de las aplicaciones dos protocolos de transporte: UDP y TCP.

### 6.6.1. Protocolo de Control de Transmisión (TCP)

El modelo de servicio TCP incluye un servicio orientado a la conexión y un servicio de transferencia de datos fiable. Cuando una aplicación invoca TCP como su protocolo de transporte, la aplicación recibe ambos servicios de TCP [14].

- Servicio orientado a la conexión: El protocolo hace que el cliente y el servidor intercambien entre sí información de control de la capa de transporte antes de que empiecen a fluir los mensajes del nivel de aplicación. Este procedimiento denominado de negociación, de reconocimiento o de establecimiento de la conexión, alerta al cliente y al servidor, permitiéndoles prepararse para el intercambio de paquetes. Después de esta fase de negociación, se dice que existe una conexión TCP entre los sockets de los dos procesos. La conexión permite que los dos procesos puedan enviarse mensajes entre sí a través de la conexión al mismo tiempo.
- Servicio de transferencia de datos fiable: Los procesos que se están comunicando pueden confiar en TCP para entregar todos los datos enviados sin errores y en el orden correcto. Cuando un lado de la aplicación pasa un flujo de bytes a un socket, puede contar con

TCP para entregar el mismo flujo de bytes al socket receptor sin pérdida ni duplicación de bytes.

A diferencia de UDP, TCP es un protocolo orientado a la conexión. Esto significa que antes de que el cliente y el servidor puedan empezar a enviarse datos entre sí, tienen que seguir un proceso de acuerdo en tres fases y establecer una conexión TCP. Un extremo de la conexión TCP se conecta al socket del cliente y el otro extremo se conecta a un socket de servidor. Cuando creamos la conexión TCP, asociamos con ella la dirección del socket de cliente (dirección IP y número de puerto) y la dirección del socket de servidor (dirección IP y número de puerto). Una vez establecida la conexión TCP, cuando un lado desea enviar datos al otro lado, basta con colocar los datos en la conexión TCP a través de su socket. Esto es distinto al caso de UDP, en el que el servidor tiene que tener asociada al paquete una dirección de destino antes de colocarlo en el socket [14].

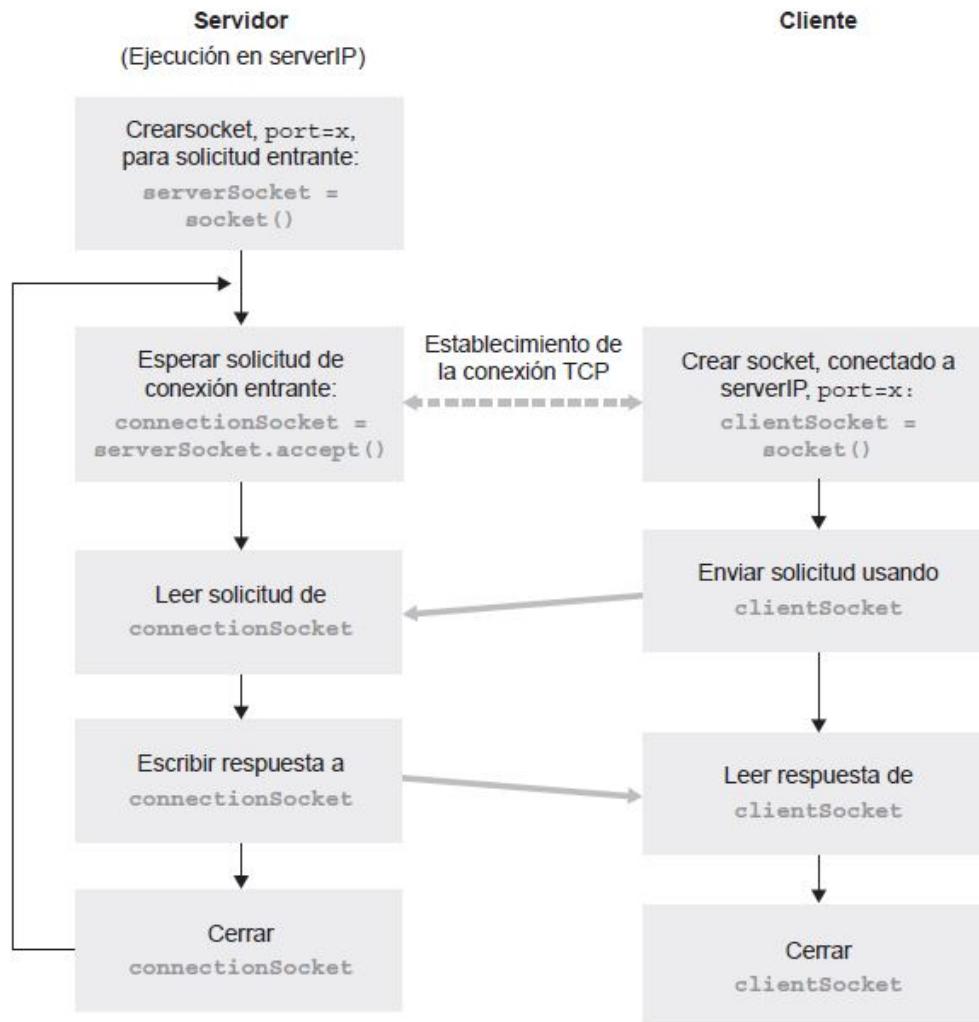


Figura 9: Aplicación cliente-servidor usando TCP [14]

### **6.6.2. Protocolo de Datagramas de usuario (UDP)**

UDP es un protocolo de transporte ligero y simple que proporciona unos servicios mínimos. No está orientado a la conexión, por lo que no tiene lugar un procedimiento de negociación antes de que los dos procesos comiencen a comunicarse. UDP proporciona un servicio de transferencia de datos no fiable; es decir, cuando un proceso envía un mensaje a un socket UDP, el protocolo UDP no ofrece ninguna garantía de que el mensaje vaya a llegar al proceso receptor. Además, los mensajes que sí llegan al proceso receptor pueden hacerlo de manera desordenada [14].

UDP no incluye tampoco un mecanismo de control de congestión, por lo que el lado emisor de UDP puede introducir datos en la capa inferior (la capa de red) a la velocidad que le parezca. Si se usa UDP, antes de que un proceso emisor pueda colocar un paquete de datos en la puerta del socket, tiene que asociar en primer lugar una dirección de destino al paquete. Una vez que el paquete atraviesa el socket del emisor, Internet utilizará la dirección de destino para en rutar dicho paquete hacia el socket del proceso receptor, a través de Internet. Cuando el paquete llega al socket de recepción, el proceso receptor recuperará el paquete a través del socket y a continuación inspeccionará el contenido del mismo y tomará las acciones apropiadas [14].

Al incluir la dirección IP de destino en el paquete, los routers de Internet serán capaces de en rutar el paquete hasta el host de destino. Pero, dado que un host puede estar ejecutando muchos procesos de aplicaciones de red, cada uno de ellos con uno o más sockets, también es necesario identificar el socket concreto dentro del host de destino. Cuando se crea un socket, se le asigna un identificador, al que se denomina número de puerto. Por tanto, como cabría esperar, la dirección de destino del paquete también incluye el número de puerto del socket [14].

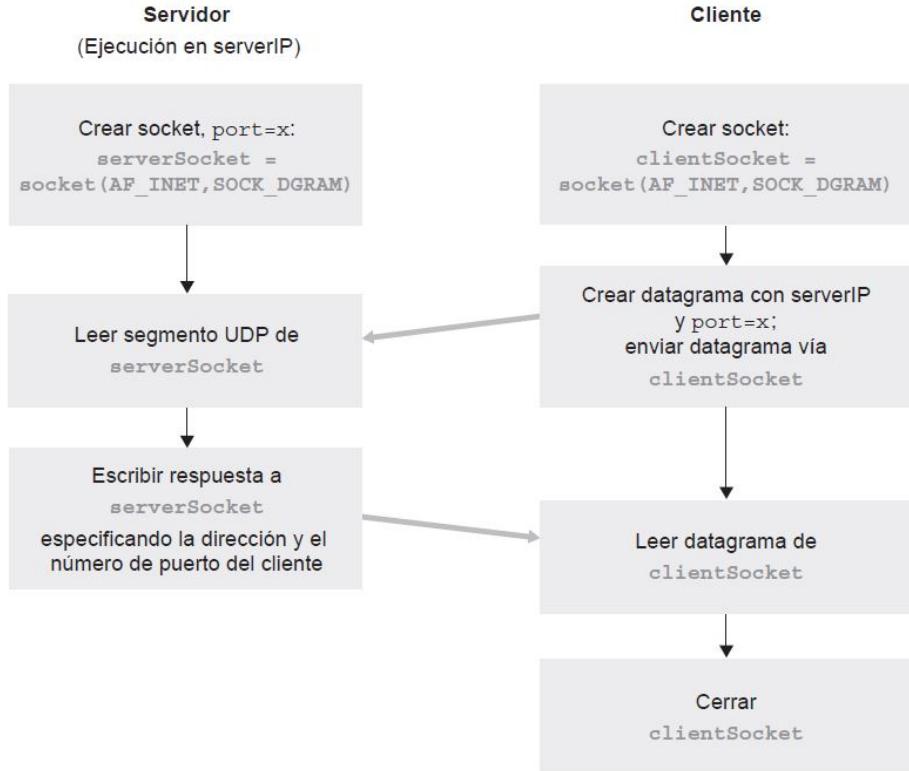


Figura 10: Aplicación cliente-servidor usando UDP [14]

## 6.7. Robots Móviles

La robótica móvil se considera actualmente un área de la tecnología avanzada manejadora de problemas de alta complejidad. Las propiedades características de los robots son la versatilidad y la autoadaptabilidad. La primera se entiende como la potencialidad estructural de ejecutar tareas diversas, lo cual implica una estructura mecánica de geometría variable. La autoadaptabilidad significa que un robot debe, por sí solo, alcanzar su objetivo a pesar de las perturbaciones imprevistas del entorno a lo largo de la ejecución de su tarea [15].

El proceso más básico para la Navegación de un robot móvil se basa en el modelo cinemático del sistema de propulsión. Este sistema es el que permite al robot moverse dentro de un determinado entorno. Uno de los sistemas más usuales se basa en el uso de ruedas de tracción diferencial [16].

### 6.7.1. Modelo Uniciclo

El modelo uniciclo consiste en una sola rueda y relaciona la velocidad angular de la rueda con su eje de rotación paralela al suelo y la velocidad lineal de la rueda con respecto al piso

[1]. Esto permite plantear las velocidades lineales de las ruedas como:

$$V_R = \phi_R r \quad (15)$$

$$V_L = \phi_L r \quad (16)$$

Donde la velocidad angular de ambos motores es  $\phi$ ,  $r$  es el radio de las ruedas y  $w$  es la velocidad angular del robot.

### 6.7.2. Modelo Diferencial

El modelo diferencial de un robot móvil considera ya 2 ruedas, se relacionan las velocidades lineales con el centro de masa del robot, al tener un robot uniforme la velocidad lineal del centro de masa es el promedio de las velocidades lineales de cada llanta, de forma que podemos plantear las ecuaciones de velocidad de ambas llantas en función de la velocidad lineal y angular [1].

$$V_R = \frac{v + wl}{r} \quad (17)$$

$$V_L = \frac{v - wl}{r} \quad (18)$$

Donde  $l$  es el radio del robot y  $r$  el radio de las llantas.

### 6.7.3. Difeomorfismo para la transformación de cinemática uniciclo

Debido a que el sistema dinámico de un robot diferencial es no lineal el aplicarle control no es una buena idea. Por lo que necesitamos realizar un ajuste llamado difeomorfismo buscando reducir la información que tenemos del robot móvil, con el difeomorfismo buscamos controlar el robot diferencial conociendo la velocidad y orientación de un punto [17].

Gracias a este ajuste es posible plantear las ecuaciones de velocidad lineal y angular, para ver el desarrollo algebraico completo se recomienda leer el capítulo 6 de [1] así como sus referencias.

$$v = u_1 \cos \phi + u_2 \sin \phi \quad (19)$$

$$w = \frac{-u_1 \sin \phi + u_2 \cos \phi}{l} \quad (20)$$

Donde  $u_1$  y  $u_2$  representan el vector de control y  $l$  la distancia entre el centro y el punto de difeomorfismo.

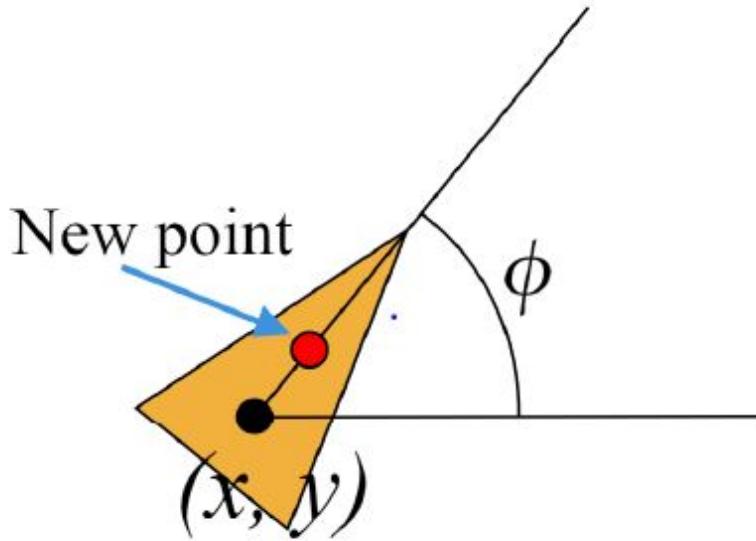


Figura 11: Ajuste del difeomorfismo [17].

## 6.8. Controladores de Posición y Velocidad de Robots Diferenciales

Estos controladores se plantean para asegurar que el robot logre llegar a la posición deseada, además que la trayectoria seguida sea suave, para ver un mayor desarrollo en estos controladores se recomienda leer el capítulo 6 de [1].

### 6.8.1. Control proporcional de velocidades con saturación limitada

Este controlador usa los errores de posición, orientación y las velocidad para acotar las velocidades del robot para ejecutar la trayectoria dada [16]. Las entradas de control se definen de la forma:

$$u_1 = I_x \tanh \frac{k_x(x_g - x)}{I_x} \quad (21)$$

$$u_2 = I_y \tanh \frac{k_y(y_g - y)}{I_y} \quad (22)$$

Donde  $(x_g$  y  $y_g)$  representan la meta y  $(x$  y  $y)$  representan la posición actual,  $I_x$  y  $I_y$  son las constantes de saturación y se encargan de evitar que las velocidades resultantes sean demasiado grandes cuando el error inicial de posición sea demasiado grande. Al aplicar estas entradas de control con el difeomorfista tenemos:

$$v = I_x \tanh \frac{k_y(y_g - y)}{I_y} \cos \phi + I_y \tanh \frac{k_y(y_g - y)}{I_y} \sin \phi \quad (23)$$

$$w = \frac{-I_x \tanh \frac{k_x(x_g - x)}{I_x} \sin \phi + I_y \tanh \frac{k_y(y_g - y)}{I_y} \cos \phi}{l} \quad (24)$$

### 6.8.2. Control PID de velocidad lineal y angular

En los robots móviles uno de los controladores mas usados es el PID, parte primordial de este controlador es la determinación de la constantes  $K_P$ ,  $K_I$  y  $K_D$ , si bien existen ciertas normas para la determinación de estas constantes calcularlas al tanteo resulta una buena práctica. Con estas constantes se busca reducir el error en estado estable así como mejorar los parámetros de rendimiento  $t_p$ ,  $t_s$  y  $M_p$  [1]. La ecuación general del controlador PID viene dada por:

$$PID(e(t)) = K_P e(t) + K_I \int_0^t e(t) dt + K_D \frac{de(t)}{dt} \quad (25)$$

Su implementación para la velocidad lineal y angular esta dada por el uso del error de orientación  $e_o$  y el error de posición  $e_p$ , estos son calculados:

$$e_o = \text{atan2}\left(\frac{\sin(\theta_g - \theta)}{\cos(\theta_g - \theta)}\right) \quad (26)$$

$$e_p = \sqrt{(x_g - x)^2 + (y_g - y)^2} \quad (27)$$

Donde  $\theta_g$  es al ángulo calculado desde el punto actual hasta la meta y  $\theta$  ángulo actual.

### 6.8.3. Control de pose

El control de pose toma en consideración la pose u orientación final que tendrá el robot al llegar a la meta, esta pose final depende la pose inicial que tenga el robot. Para profundizar en el desarrollo algebraico se recomienda leer el capítulo 6 de [1]. Las ecuaciones polares de coordenadas son:

$$\rho = \sqrt{(\Delta x)^2 + (\Delta y)^2} \quad (28)$$

$$\alpha = -\theta + \text{atan2}(\Delta y, \Delta x) \quad (29)$$

$$\beta = -\theta - \alpha \quad (30)$$

Mientras que las leyes de controlador de pose son:

$$v = k_p \rho \quad (31)$$

$$w = k_\alpha \alpha k_\beta \beta \quad (32)$$

Donde  $\rho$  es el error de posición este ayuda al robot a orientarse y seguir la linea entre el robot y la meta, mientras  $\beta$  se encarga de orientar dicha línea asegurándose que el robot logre la pose final al llegar a la meta.

#### 6.8.4. Control de pose de Lyapunov

Este controlador toma como base el criterio de estabilidad de Lyapunov para asegurar sea asintóticamente estable. Este criterio nos dice que si existe una solución lo suficientemente cerca de un punto de equilibrio  $X_o$  de una ecuación diferencial homogénea, esta se mantiene cerca para todo  $t > t_o$ , de igual forma este punto de equilibrio  $X_o$  es asintóticamente estable si posee estabilidad de Lyapunov además de ser un atractor de soluciones. Para un mayor desarrollo y desarrollo algebraico se recomienda leer el capítulo 6 de [1].

$$v = k_p \rho \cos \alpha \quad (33)$$

$$w = k_p \sin \alpha \cos \alpha k_\alpha \alpha \quad (34)$$

#### 6.8.5. Control Closed-Loop Steering

Este tipo de control de pose es de carácter similar a los anteriores variando en su cálculo de velocidad angular, para profundizar en este tipo de controlador se recomienda leer el capítulo 6 de [1].

$$v = k_p \rho \cos \alpha \quad (35)$$

$$w = \frac{2v}{5\rho} (k_2(\alpha + \text{atan}(-k_1\beta)) + (1 + \frac{k_1}{1 + (k_1\beta)^2}) \sin \alpha) \quad (36)$$

#### 6.8.6. Control por medio de Regulador Lineal Cuadrático (LQR)

El control LQR permite la estabilización de un sistema dinámico alrededor de un punto de operación, de igual forma este busca realizar control sobre un sistema de forma óptima, esto quiere decir usar la menor cantidad de control posible. Para un mayor detalle del desarrollo matemático se recomienda leer el capítulo 6 de [1]. Recordemos que el sistema de un robot diferencial es no lineal usamos el difeomorfismo para aplicar el control LQR, de forma que tenemos:

$$u = -K(x_g - x) + u_g \quad (37)$$

#### 6.8.7. Controlador Lineal Cuadrático Integral (LQI)

Debido a que el control LQR no es robusto contra las perturbaciones, y existe cierta incertidumbre entre el modelo del sistema y el sistema real, se busca compensar el error en estado estable agregando una parte integral. Recordemos que el sistema de un robot diferencial es no lineal usamos el difeomorfismo para aplicar el control LQI [1], de forma que tenemos:

$$u = -K_1 x + K_2 \sigma \quad (38)$$

## 6.9. Raspberry Pi

Se le conoce como *Raspberry Pi* a la serie de ordenadores de placa reducida desarrollados por el Reino Unido por la *Raspberry Pi Foundation*. Estos modelos fueron creados para la enseñanza de la informática en escuelas, a medida que su éxito creció se desarrollaron modelos más complejos dando lugar diversas versiones lo que dio lugar a la *Raspberry Pi Trading* encargada de la producción de las nuevas versiones [18].

### 6.9.1. Hardware

Todos estos modelos utilizan un arquitectura para el procesador ARM. Esta arquitectura es de tipo RISC (*Reduced Instruction Set Computer*), es decir, utiliza un sistema de instrucciones realmente simple lo que le permite ejecutar tareas con un mínimo consumo de energía [18].

### 6.9.2. Software

*Raspberry Pi OS* es el sistema operativo recomendado para el uso común en una *Raspberry Pi*, este sistema operativo es de uso gratuito basado en Debian, optimizado para el hardware *Raspberry Pi*. Este sistema operativo cuenta con mas de 35,000 paquetes y se puede descargar desde la pagina oficial: *Raspberry Pi OS*, se descarga una copia exacta del sistema operativo en una “imagen”, la cual contiene la estructura y los contenidos completos de un sistema operativo para luego ser copiada en una tarjeta SD para su uso en el hardware *Raspberry Pi*, el proceso de copiar la imagen en una SD se realiza mediante la aplicación gratuita *Raspberry Pi Imager* [19].

### 6.9.3. Modelos

Conforme el paso del tiempo la compañía *Raspberry Pi Trading* fue desarrollando diferentes modelos de acuerdo a las nuevas tecnologías disponibles, entre los cuales podemos encontrar:

- Raspberry Pi 1 modelo A (descontinuada)
- Raspberry Pi 1 modelo B (descontinuada)
- Raspberry Pi 2 modelo B
- Raspberry Pi 3 modelo B+
- Raspberry Pi 3 modelo A+
- Raspberry Pi 4 modelo B

A parte de los modelos normales, también se han desarrollado otra gama de placas denominadas *Raspberry Pi Zero*. Estas son mucho más pequeñas y menos potentes que sus hermanas, pero es precisamente su atractivo, menos gasto y un precio mucho menor [20].

## 6.10. Microcontroladores

Un microcontrolador es un circuito integrado que en su interior contiene una unidad central de procesamiento (CPU), unidades de memoria (RAM y ROM), puertos de entrada y salida y periféricos. Estas partes están interconectadas dentro del microcontrolador, y en conjunto forman lo que se le conoce como microcomputadora [21].

Los microcontroladores suelen clasificarse por familias o por el número de Bits que manejan (4, 8, 16 ó 32 bits). Lógicamente los microcontroladores de 16 y 32 bits son superiores a los de 4 y 8 bits en cuanto a funcionalidades, siendo que los microcontroladores de 8 bits dominan el mercado. Prácticamente la totalidad de los microcontroladores actuales se fabrican con tecnología CMOS 4 (Complementary Metal Oxide Semiconductor). Esta tecnología supera a las técnicas anteriores por su bajo consumo y alta inmunidad al ruido [21].

### 6.10.1. Arquitectura ARM

La arquitectura ARM tiene un conjunto de instrucciones simple pero eficiente que permite un tamaño de silicio compacto y ofrece alta velocidad de ejecución a bajo consumo. Cuentan con procesador RISC de 32 bits desarrollado por ARM Ltd. Debido a sus atributos de ahorro de energía, las unidades de procesamiento central ARM prevalecen en el mercado de la electrónica móvil, donde un menor gasto de energía es un objetivo de diseño vital [22].

### 6.10.2. Arquitectura AVR

El AVR también conocido como *Advanced Virtual RISC* es una arquitectura Harvard del tipo RISC de 8 bits con dos espacios de memoria completamente independientes: memoria de programa y memoria de datos. En la memoria de programas se encuentra el código a ejecutar. Es una memoria de 16 bits y la mayor parte de las instrucciones son de este tamaño. Algunas instrucciones necesitan dos posiciones de memoria [23].

La memoria de datos es de 8 bits y se divide en tres secciones. Existen instrucciones específicas para acceder a cada una de estas secciones de memoria, pero también hay instrucciones que pueden acceder a todo el espacio de memoria indistintamente. La parte más baja de esta memoria alberga 32 registros de 8 bits, seis de los cuales pueden agruparse de a pares para formar tres registros de 16 bits, usualmente usados como punteros. A continuación se encuentra el espacio de entrada/salida, con un total de 64 posiciones de 8 bits. El resto de la memoria es RAM [23].

### 6.10.3. Arquitectura PIC

La arquitectura PIC desarrollada por *Micro-chip Technology* para categorizar sus microcontroladores de chip solitarios. En esta arquitectura, la CPU se conecta de forma independiente y con buses distintos con la memoria de instrucciones y con la de datos y así permitir su acceso simultaneo [24].

La popularidad de estos micros radica en su alta disponibilidad en el mercado y bajo precio. Cuando se aprende a manejar un modelo específico, conociendo su arquitectura y su repertorio de instrucciones, es muy fácil emplear otro modelo.

### Familia de 8-bits

Esta familia es la mas común de Microchip y destacan por su bajo precio. Poseen una dirección y un bus de datos separados. Lo que significa que también pueden estar separados en tamaño. Todos poseen un bus de datos de 8 bits de ancho, pero el bus del programa variará en tamaño dependiendo de la categoría [21]. La familia de 8 bits tiene cuatro categorías:

- Línea de base: número de pines escasos, hay micros de 6 a 14 pines.
- Rango medio: bajos coste y un número abundante de periféricos internos.
- Enhanced Mid-Range: varios periféricos juntos y listos para poder ser usados con sus respectivos pins.
- High End: cantidad de pines y periféricos considerables.

### Familia de 16-bits

Esta familia presenta dos opciones cuyo procesador es de 16-bits. Una de ellas es el PIC24, que sigue la línea de microcontroladores de 8-bits, y la otra es la denominada dsPIC. Ambas opciones fueron diseñadas en un intento de Microchip en introducirse en el mundo del procesamiento de señales [21].

### Familia de 32-bits

Esta familia presenta una única opción, la denominada PIC32, diseñados para aplicaciones embebidas que requieran una cantidad de memoria mayor, un procesado de la información mayor, y una cantidad considerable de periféricos [21].

## 6.11. Arduino Uno

Arduino es una plataforma de prototipos electrónica de código abierto basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Hay múltiples versiones de la placa Arduino, la mayoría usan el ATmega168 de Atmel. El microcontrolador de Arduino posee lo que se llama una interfaz de entrada, que es una conexión en la que podemos conectar en la placa diferentes tipos de periféricos. La información de estos periféricos que conectes se trasladará al microcontrolador, el cual se encargará de procesar los datos que le lleguen a través de ellos [25].



Figura 12: Arduino Uno [25]

## 6.12. Tiva C

La *Tiva C Series TM4C123G LaunchPad Evaluation Board* o como comúnmente se le conoce Tiva C es una plataforma de evaluación de bajo costo para microcontroladores *ARM Cortex -M4F*. Esta incluye botones de usuario programables y un LED RGB para aplicaciones personalizadas [26].

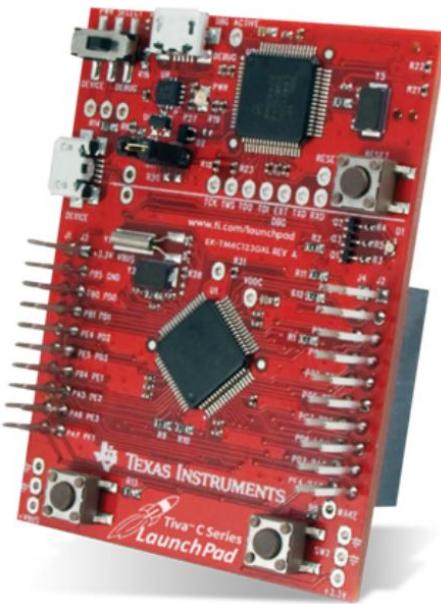


Figura 13: Tiva C [26]

## 6.13. Propuesta de Robots

### 6.13.1. Kilobot

El Kilobot es un sistema robótico de bajo costo, estos enjambres están inspirados en insectos sociales, como colonias de hormigas, que pueden buscar y encontrar eficientemente fuentes de alimento en grandes ambientes complejos, transportar colectivamente grandes objetos y coordinar la construcción de nidos y puentes en tales ambientes [27].

El Kilobot está diseñado para proporcionar a los científicos un banco de pruebas físicas para avanzar en la comprensión del comportamiento colectivo y realizar su potencial para ofrecer soluciones para una amplia gama de desafíos. Este tiene un máximo de 33 mm de diámetro [27].

Algunas de sus características físicas son:

- Diámetro: 33 mm
- Altura: 34 mm
- Costo: 12 dólares
- Max. distancia de comunicación : 7 cm
- Autonomía: 1 horas en movimiento



Figura 14: Kilobot [27].

### 6.13.2. E-puck

Este robot fue desarrollado por la Escuela Politécnica Federal de Lausana (EPFL) el cual fue creado para fines educativos y de investigación. Este pequeño robot es cilíndrico y cuenta con dos ruedas, equipado con una variedad de sensores cuya movilidad está garantizada por un sistema de accionamiento diferencial. Tanto su diseño como librerías y manuales de uso son de código abierto (*open source*) y todo se encuentra disponible en su página oficial [28].

Algunas de sus características físicas son:

- Diámetro: 70 mm
- Altura: 50 mm
- Peso: 200 g
- Max. velocidad: 13 cm/s
- Autonomía: 2 horas en movimiento

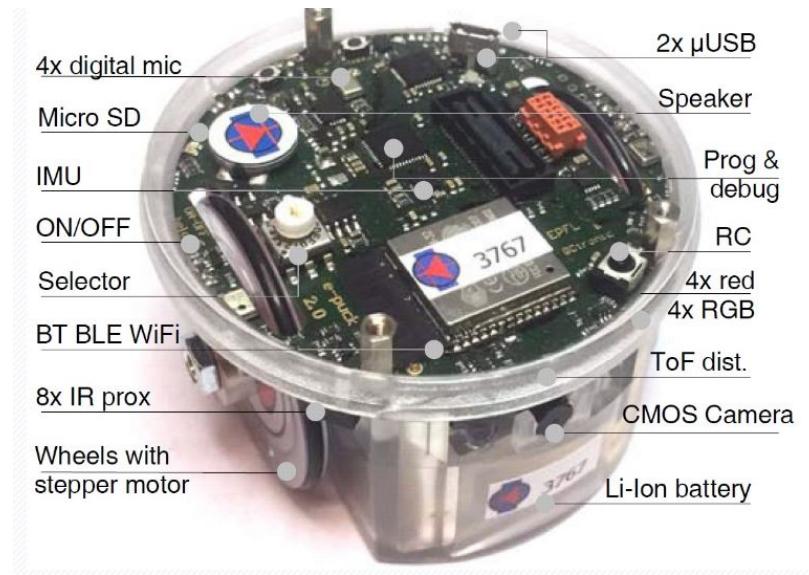


Figura 15: E-puck [28].

### 6.13.3. Pi-puck

El Pi-puck es una plataforma robótica desarrollada por el *York Robotics Laboratory* en la Universidad de York y GCtronic. Este básicamente es una extensión del e-puck y el e-puck2, el cual posee integrado una Raspberry Pi Zero agregando soporte Linux así como periféricos adicionales. Al ser de uso comercial se cuenta con una distribución de software YRL para el Pi-Puck en donde se incluye una imagen personalizada de Raspian y varios paquetes que permiten un completo control sobre el hardware y una serie de bibliotecas ya instaladas para hacer mas fácil el uso del robot [29].

La Raspberry Pi Zero se conecta al robot por medio de I2C, posee un micrófono digital y un altavoz. Consta de 2 baterías internas las cuales se cargan por medio de cable USB. Posee seis canales de I2C, dos entradas ADC así como múltiples leds para verificar funciones de encendido.

El precio de cada pi-puck es de Q 3,163.89 los cuales son distribuidos en Guatemala por *GCtronic*



Figura 16: Pi-puck [29].

# CAPÍTULO 7

---

## Validación de microcontrolador, sistemas embebidos, entorno de desarrollo, lenguaje de programación y robots

---

Para la implementación del algoritmo MPSO se evaluaron distintos tipos de microcontroladores, sistemas embebidos y entornos de desarrollo. Se selecciona el lenguaje de programación y el tipo de robot móvil que mejor se adapte a las necesidades del algoritmo.

### 7.1. Evaluación

Las arquitecturas de microcontroladores ARM y AVR son descartadas debido a que no son tan comunes en Guatemala por lo que se dificulta su obtención. Se opta por una arquitectura PIC (alta disponibilidad en el mercado y bajo precio) con la cual se tiene una previa experiencia de uso.

Dentro de la arquitectura PIC encontramos las familias de 8, 16 y 32 bits, a continuación estos se muestran en los Cuadros 1, 2 y 3. Se muestran las principales características de estas familias para lo cual se toma tres microcontroladores correspondientes a las tres familias mencionadas.

Microcontrolador	PIC16F877A
Voltaje Operativo (V)	5
Flash (bytes)	8 K
EPROM (bytes)	256
SRAM (bytes)	368
Frecuencia de operación (MHz)	20
Pines	40
Costo (Q)	79.0

Cuadro 1: Características del PIC16F877A [30].

Microcontrolador	PIC24F16KM202
Voltaje Operativo (V)	5
Flash (bytes)	16 K
EPROM (bytes)	512
SRAM (bytes)	2 K
Frecuencia de operación (MHz)	32
Pines	40
Costo (Q)	293.58

Cuadro 2: Características del PIC24F16KM202 [31].

Microcontrolador	PIC32MZ0512EFE064
Voltaje Operativo (V)	5
Flash (bytes)	160 K
EPROM (bytes)	128 K
SRAM (bytes)	512 K
Frecuencia de operación (MHz)	200
Pines	40
Costo (Q)	346.24

Cuadro 3: Características del PIC32MZ0512EFE064 [32]

Dentro de las principales plataformas de evaluación tenemos a la Tiva C y al Arduino Uno, a continuación se muestran en los Cuadros 4 y 5. Se muestran las principales características de estas dos plataformas.

Microcontrolador	ATmega328
Voltaje Operativo (V)	5
Flash (bytes)	32 K
EPROM (bytes)	1 k
SRAM (bytes)	2 k
Frecuencia de operación (MHz)	16
Pines	40
Costo (Q)	125.00

Cuadro 4: Características del Arduino Uno [25].

Microcontrolador	TM4C123GH6PM
Voltaje Operativo (V)	5
Flash (bytes)	256 KB
EPROM (bytes)	2 KB
SRAM (bytes)	32 KB
Frecuencia de operación (MHz)	80
Pines	40
Costo (Q)	260

Cuadro 5: Características de la Tiva C [26].

Dentro de los diferentes modelos con los que cuenta el ordenador *Raspberry Pi* tenemos el modelo 3B y el 4B. En el Cuadro 6 se pude ver una comparación entre ambos modelos.

Modelo	3B	4B
Procesador	Quad Core Cortex A-72 1,5 GHz	Quad Core Cortex A-53 1,4GHz
Memoria	1, 2, 4 GB LPDDR4	1 GB LPDDR2
GPU	Broadcom Videocore VI	Broadcom Videocore VI
Alimentación	USB Tipo-C	microUSB
Ethernet	Gigabit sin limitaciones	Gigabit hasta 300 Mbps
Precio(Q)	675	595

Cuadro 6: Características de la Raspberry Pi [18]

## 7.2. Selección del tipo de microcontrolador, plataforma u ordenador a utilizar

Al tener distintas opciones de microcontroladores, plataformas de evaluación y ordenadores se realiza un *trade study*, el cual es un estudio que ayuda a identificar una solución entre una lista de soluciones calificadas.

### 7.2.1. Criterios Usados

- **Capacidad de memoria:** además de la implementación del algoritmo PSO se agregan diversos controladores para ayudar con el suavizado y control de la trayectoria a seguir para llegar a la meta establecida. Se usaron varias librerías, funciones y espacio de memoria para la validación del PSO por lo que el espacio de memoria se toma en cuenta para la implementación.
- **Frecuencia de operación:** el PSO es un algoritmo que se basa en múltiples iteraciones para un óptimo resultado. Que tan rápido se hacen estas iteraciones es algo a tomar en cuenta para la implementación.
- **Adaptabilidad a Robots móviles:** teniendo en cuenta el principal objetivo de esta tesis se busca que el sistema físico seleccionado sea fácilmente adaptable a una plataforma o robot móvil. Debido a que los robots móviles cuentan con diversos sensores, módulos y motores se debe tomar en cuenta la cantidad de pines, dimensiones físicas y el voltaje operativo de cada microcontrolador, plataforma u ordenador. Otro aspecto importante tomado en cuenta es el ensamblaje en el robot móvil, si se necesita una placa o realizar algún tipo de soldadura.
- **Disponibilidad:** parte fundamental de la robótica de enjambre es el trabajo en conjunto de varios agentes. Se deben tener suficientes microcontroladores, plataformas u ordenadores para realizar pruebas que validen el correcto funcionamiento del PSO. Se toma en cuenta la facilidad de adquirir los microcontroladores, plataformas u ordenadores en el mercado local.
- **Costo (Q):** de la mano del criterio anterior, si no se poseen los microcontroladores, plataformas u ordenadores se deben de comprar. Por lo que se toma en consideración el costo unitario de estos, si fuera el caso de tener que comprarlos en otro país se toma en cuenta el costo del envío también.
- **Previo Uso:** se toma en consideración que la curva de aprendizaje no sea muy elevada. No es el objetivo de esta tesis aprender a usar un nuevo microcontrolador.

### 7.2.2. Peso por criterio

El peso asignado a cada criterio se respalda en ([30], [31], [32], [25], [26] y [18]). Se usan calificaciones de uno a seis (por tener 6 alternativas) donde seis representa la mejor calificación posible y uno la peor. A continuación se detalla el peso del total del *trade study* asignado a cada criterio así como la calificación de cada alternativa evaluada.

- **Adaptabilidad a Robots móviles:** el sistema físico debe ser fácil adaptable hacia una plataforma móvil teniendo en cuenta futuras implementaciones de esta tesis. Debido a esto se le da un peso del 25 %.

Raspberry Pi	6
Tiva C	4
Arduino Uno	4
PIC de 8-bits	2
PIC de 16-bits	2
PIC de 32-bits	2

Cuadro 7: Calificación por Adaptabilidad a Robots móviles

Al ordenador *Raspberry Pi* se le asigna la máxima calificación por tener una buena cantidad de pines y otros periféricos adicionales (puerto USB, HDMI). A pesar de ser físicamente más grande que otras opciones se tiene un precedente de robots móviles haciendo uso de este ordenador [29]. El Arduino Uno y Tiva C reciben una buena calificación al contar con una buena cantidad de pines y facilidad de implementación. La familia de PIC recibe una calificación baja ya que al implementarse o querer conectar cualquier modulo o motor se debe hacer mediante una placa adicional (fabricada o comprada) y se deben realizar soldadura.

- **Disponibilidad:** se debe contar con varios dispositivos para validar correctamente el PSO. Debido a esto se le da un peso del 30 %.

Raspberry Pi	6
Tiva C	6
Arduino Uno	3
PIC de 8-bits	3
PIC de 16-bits	1
PIC de 32-bits	1

Cuadro 8: Calificación Disponibilidad

Se le asigna la máxima calificación a la *Raspberry Pi* y a la Tiva C ya que el departamento cuenta con suficientes dispositivos para poder hacer experimentos y lograr la validación del algoritmo.. El Arduino Uno y el PIC de 8-bits reciben una calificación media ya que son fácilmente adquiribles en Guatemala, los PIC de 16 y 32 bits reciben la calificación minima al no poder conseguirlos en Guatemala (se investiga en electrónicas locales).

- **Costo (Q):** si la Universidad del Valle no posee suficientes dispositivos estos se deben de comprar. El peso asignado a este criterio es de 10 %.

Raspberry Pi	1
Tiva C	3
Arduino Uno	5
PIC de 8-bits	6
PIC de 16-bits	2
PIC de 32-bits	2

Cuadro 9: Calificación del Costo

El PIC de 8 bits recibe la máxima calificación al tener el mejor precio, seguido del Arduino Uno y la Tiva C. Los PIC de 16 y 32 bits deben ser comprados fuera del país por lo que su precio es elevado al tener que sumar el costo del envío, estos precios fueron obtenidos de: <https://www.mouser.com.gt>. El ordenador *Raspberry Pi* recibe la peor calificación al tener el costo más elevado.

- **Capacidad de memoria:** En base a las capacidades de memoria de cada dispositivo (cuadros 1, 2, 3, 4, 5 y 6) El peso asignado a este criterio es de 15 %.

Raspberry Pi	6
Tiva C	5
Arduino Uno	4
PIC de 8-bits	2
PIC de 16-bits	3
PIC de 32-bits	6

Cuadro 10: Calificación de la Capacidad de memoria

- **Frecuencia de operación:** De acuerdo con las pruebas realizadas en la fase anterior [1] se usó un tiempo de muestreo de 32 milisegundos. La frecuencia de operación de todos los dispositivos (cuadros 1, 2, 3, 4, 5 y 6) permite ejecutar este tiempo sin problemas por lo que todos los dispositivos reciben una buena calificación. El peso asignado a este criterio es de 10 %.

Raspberry Pi	6
Tiva C	6
Arduino Uno	5
PIC de 8-bits	4
PIC de 16-bits	6
PIC de 32-bits	6

Cuadro 11: Calificación de la Frecuencia de operación

- **Previo Uso:** Las calificaciones dadas se basan en experiencias personales con el uso de los dispositivos. El peso asignado a este criterio es de 10 %.

Raspberry Pi	4
Tiva C	5
Arduino Uno	3
PIC de 8-bits	5
PIC de 16-bits	1
PIC de 32-bits	1

Cuadro 12: Calificación del Previo Uso

La Tiva C, *Raspberry Pi* y PIC de 8 bits reciben una calificación alta ya que fueron usados en cursos previos. El Arduino Uno y los PIC de 16 y 32 bits reciben una baja calificación al tener una poca o nula experiencia con ellos.

### 7.2.3. Selección del sistema físico

La mejor opción es el ordenador *Raspberry Pi*, el resultado se puede ver en la figura 17. El modelo a utilizar sera el 3B, con el cual ya cuenta el departamento.

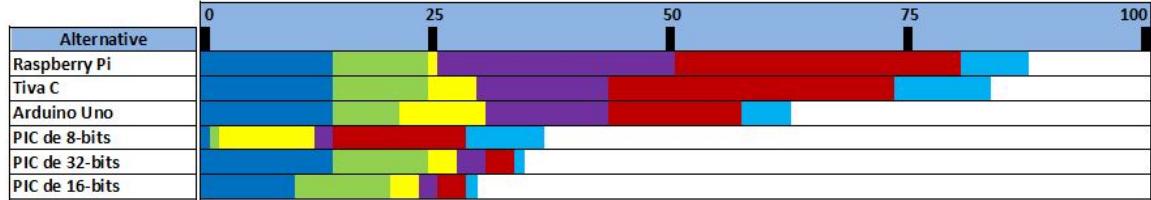


Figura 17: Resultado del *Trade Study* 1

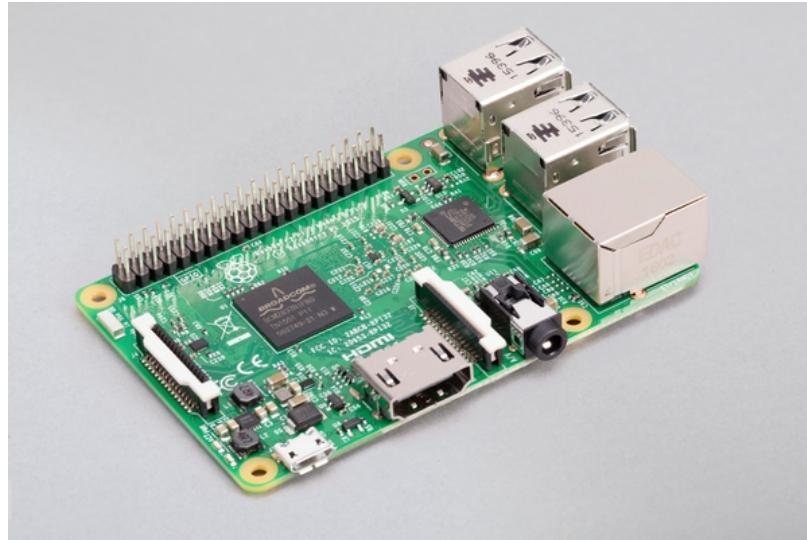


Figura 18: Raspberry Pi 3B [18].

## 7.3. Lenguaje de Programación

Los lenguajes de programación se utilizan para hacer funcionar las páginas web, las apps, el software y todo tipo de dispositivo que requiere programación informática y conocimientos de código [33]. De acuerdo a su finalidad, los lenguajes de programación se clasifican en:

- Lenguaje máquina: es el que usa cualquier máquina y se basa en un código binario [33].
- Lenguajes de programación de bajo nivel: pueden variar según el ordenador o máquina que se utilice [33].
- Lenguajes de programación de alto nivel: es un lenguaje que utiliza comandos y palabras (normalmente en inglés), las cuales es fácil de entender para un programador [33].

Se evaluaron distintos lenguajes de programación de alto nivel para realizar la implementación del algoritmo MPSO. Algunos lenguajes de programación no son tomados en cuenta principalmente porque fueron desarrollados para usos muy específicos como: JavaScrip (desarrollo de videojuegos), PHP (desarrollo de páginas web), R (enfocado al análisis estadístico) o Swift (desarrollo de aplicaciones para iOS y macOS).

### 7.3.1. Evaluación

- Lenguaje C: este lenguaje está orientado a la programación estructurada (código secuencial), con un conjunto de sentencias o instrucciones que se ejecutan una por una. Es independiente del hardware, por lo que se puede migrar a otros sistemas. Ofrece un control absoluto de todo lo que sucede en el ordenador y permite una organización del trabajo con total libertad [34].
- Lenguaje C++: es un lenguaje orientado a objetos. Una de sus principales características, el alto rendimiento que ofrece. Esto es debido a que puede hacer llamadas directas al sistema operativo, es un lenguaje compilado para cada plataforma, posee gran variedad de parámetros de optimización y se integra de forma directa con el lenguaje ensamblador [34].
- Lenguaje Python: es un lenguaje orientado a objetos, con una semántica dinámica integrada, principalmente para el desarrollo web y de aplicaciones informáticas. Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de números, archivos, etc. Además, existen muchas librerías que podemos importar en los programas para tratar temas específicos [35].
- Lenguaje Java: es un lenguaje orientado a objetos, independiente de la plataforma hardware donde se desarrolla, y que utiliza una sintaxis similar a la de C++ pero reducida. Es un lenguaje con una curva de aprendizaje baja y que dispone de una gran funcionalidad de base. Ofrece un código robusto, que ofrece un manejo automático de la memoria, lo que reduce el número de errores [34].
- Lenguaje Matlab: es un lenguaje interpretado de alto nivel, el cual permite organizar y estructurar el código permitiendo la implementación de algoritmos complejos o tareas respectivas. Este lenguaje permite operaciones de vectores y matrices, funciones, cálculo lambda, y programación orientada a objetos. [36].

### 7.3.2. Criterios Usados

- **Disponibilidad de librerías:** al buscar una implementación en un sistema físico y realizar la correcta implementación del PSO se evalúa la existencia de librerías usadas para el control de módulos, protocolos de comunicación, funciones matemáticas, etc.
- **Previo Uso:** se busca que la curva de aprendizaje del lenguaje de programación no sea muy elevada. No es objetivo de esta tesis aprender un nuevo lenguaje de programación.
- **Adaptabilidad:** al haber seleccionado el ordenador *Raspberry Pi* se evalúa la adaptabilidad del lenguaje de programación con este dispositivo. Es importante que sea

possible implementar el lenguaje de programación en diferentes dispositivos pensando en futuras pruebas con una plataforma móvil.

### 7.3.3. Peso por criterio

Para el peso asignado a cada criterio se usan calificaciones de uno a cinco (por tener 5 alternativas) donde cinco representa la mejor calificación posible y uno la peor. A continuación se detalla el peso total del *trade study* asignado a cada criterio así como la calificación de cada alternativa evaluada.

- **Disponibilidad de librerías:** que existan librerías bien documentadas facilita el proceso de implementación y validación del PSO. Se considera también el fácil acceso a estas librerías, fácil instalación y que no sean pagadas. El peso asignado a este criterio es de 40 %.

Lenguaje C	4
Lenguaje C++	4
Lenguaje Python	5
Lenguaje Java	3
Lenguaje Matlab	1

Cuadro 13: Calificación disponibilidad de Librerías

La mayor cantidad de librerías encontradas eran del lenguaje Python. Mucha documentación de la *Raspberry Pi* se encuentra en este lenguaje, debido a esto Python recibe la mayor calificación. Java, C y C++ cuentan con varias librerías y son de fácil acceso por lo que obtienen una calificación alta. Matlab tiene muchas librerías, sin embargo la mayoría son pagadas y debido a esto se le asigna la peor calificación.

- **Previo Uso:** estas calificaciones se basan al uso previo de cada lenguaje de programación. El peso asignado a este criterio es de 30 %.

Lenguaje C	5
Lenguaje C++	1
Lenguaje Python	1
Lenguaje Java	1
Lenguaje Matlab	5

Cuadro 14: Calificación por previo uso

El lenguaje C como Matlab reciben la máxima calificación posible al ser los más usados a lo largo de la carrera (cursos de Control, Robótica, Microcontroladores, etc.). Se tiene una experiencia nula en lenguaje C++, Python y Java por lo que se les asigna la peor calificación.

- **Adaptabilidad:** Ya que la *Raspberry Pi* no tiene un entorno de desarrollo exclusivo se tiene cierta libertad para seleccionar el lenguaje de programación. Todos los lenguajes

obtiene una calificación alta, exceptuando Matlab el cual trabaja con su propio entorno de desarrollo integrado y su acople a otros dispositivos conlleva un proceso más largo y complicado. El peso asignado a este criterio es de 30 %.

Lenguaje C	5
Lenguaje C++	5
Lenguaje Python	5
Lenguaje Java	5
Lenguaje Matlab	1

Cuadro 15: Calificación por adaptabilidad

#### 7.3.4. Selección del lenguaje de programación

Las mejores opciones son los lenguajes de programación C y Python, el resultado se puede ver en la figura 19. Ya que el ordenador *Raspberry Pi* no tiene un entorno de desarrollo exclusivo, se optó por trabajar en Visual Studio Code. Se aprovechó de herramientas de este editor como el resaltado de sintaxis y su finalización de código inteligente.

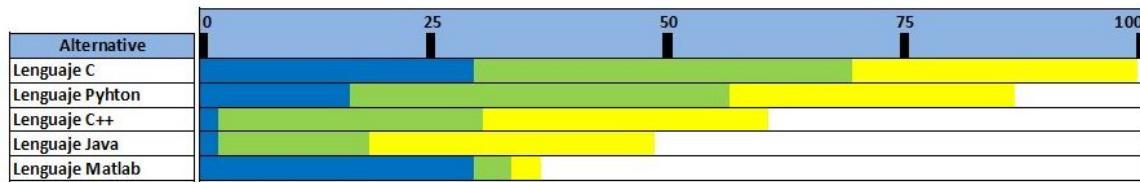


Figura 19: Resultado del *Trade Study 2*.



Figura 20: Visual Studio Code [37]

# CAPÍTULO 8

---

## Implementación del Particle Swarm Optimization

---

Para la implementación del algoritmo PSO en un sistema físico se toma como base los repositorios elaborados en las fases previas a esta tesis ([1] y [2]).

### 8.1. Validación del PSO

Como primera validación se implemento el algoritmo ordinario PSO en el ordenador *Raspberry Pi* para comparar su funcionamiento con el desarrollado en Matlab para fases anteriores [1].

Para que esta comparación sea valida ambos algoritmos fueron ajustados bajo las mismas métricas. Los agentes fueron modelados como partículas sin dimensiones físicas, el espacio de búsqueda fue limitado a [-5,5] en X y Y, además se estableció un máximo de 100 iteraciones para ejecutar el PSO. El factor de restricción y ambas constantes de escalamiento fueron las mismas para ambas aplicaciones ( $c_1=2$  y  $c_2 =10$ ) y se probaron dos tipos de inercia, la constante y la caótica. Se evaluaron dos funciones costo (*Sphere* y *Himmelblau*) mientras se vario la cantidad de agentes utilizados para cada prueba.

Se evaluó la capacidad de convergencia de ambas aplicaciones, por lo cual se comparo la mejor posición global obtenida por los agentes tomando hasta siete cifras significativas para hacer valida la comparación. Los resultados de estas comparaciones se pueden ver en las Figuras: 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 y 32

### 8.1.1. Función *Sphere*

#### Inercia Constante

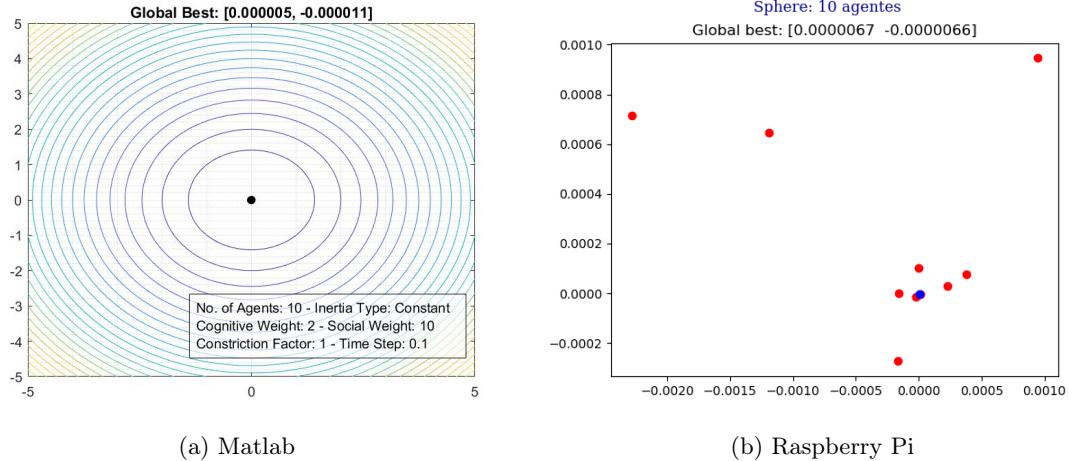


Figura 21: PSO con 10 agentes Inercia Constante

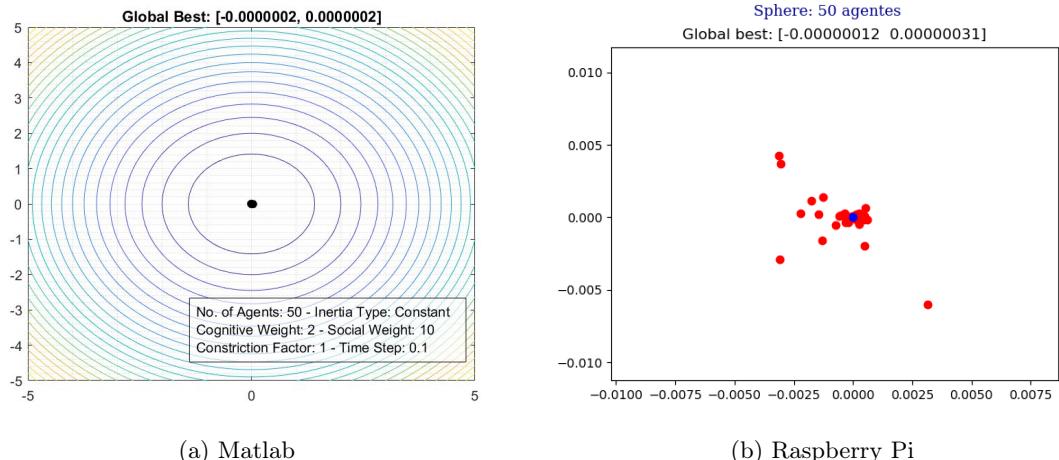


Figura 22: PSO con 50 agentes Inercia Constante

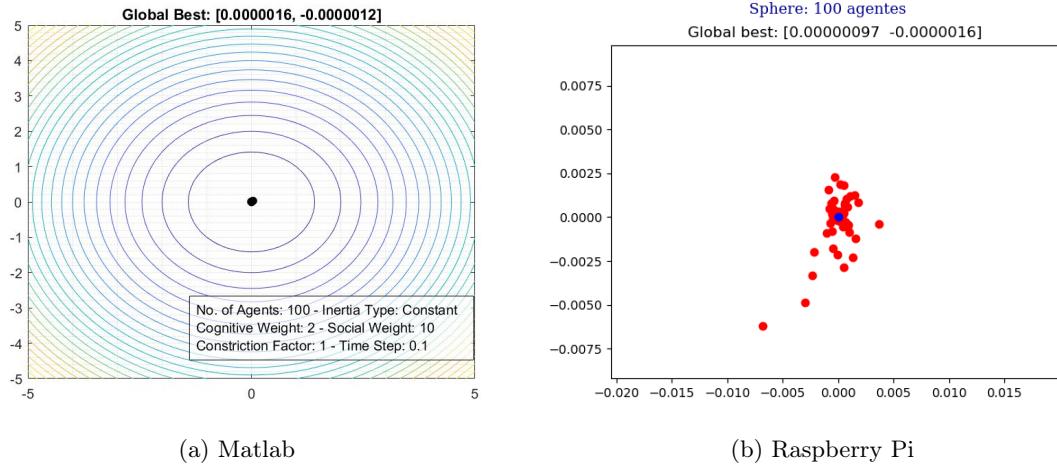


Figura 23: PSO con 100 agentes Inercia Constante

En el Cuadro 16, en la columna de matlab vemos la mejor posición alcanzada para esta aplicación para el caso de 10, 50 y 100 agentes; estos valores fueron tomados como teórico. En la columpa de *Raspberry Pi* vemos la mejor posición alcanzada para esta aplicación en donde se realizaron un total de 50 ejecuciones del código hasta alcanzar el menor porcentaje de error.

No. de agentes	Matlab		Raspberry Pi		% de Error	
	X	Y	X	Y	X	Y
10	0.0000050	-0.0000110	0.0000067	-0.0000066	34 %	40 %
50	-0.0000002	0.0000002	-0.0000001	0.0000003	40 %	55 %
100	0.0000016	-0.0000012	0.0000010	-0.0000016	39 %	33 %

Cuadro 16: Resultados de la comparación del PSO matlab vs PSO *Raspberry Pi* con Inercia Constante evaluando la función *Sphere*

## Inercia Caótica

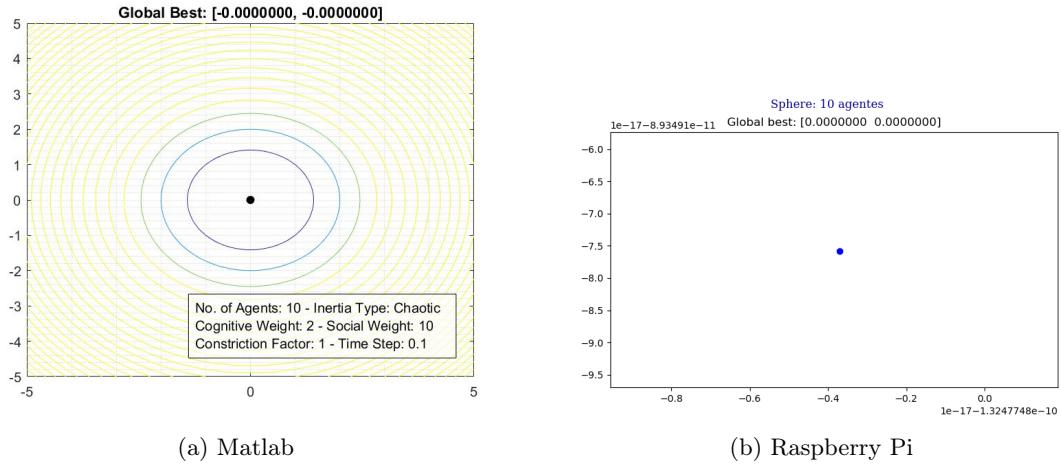


Figura 24: PSO con 10 agentes Inercia Caótica

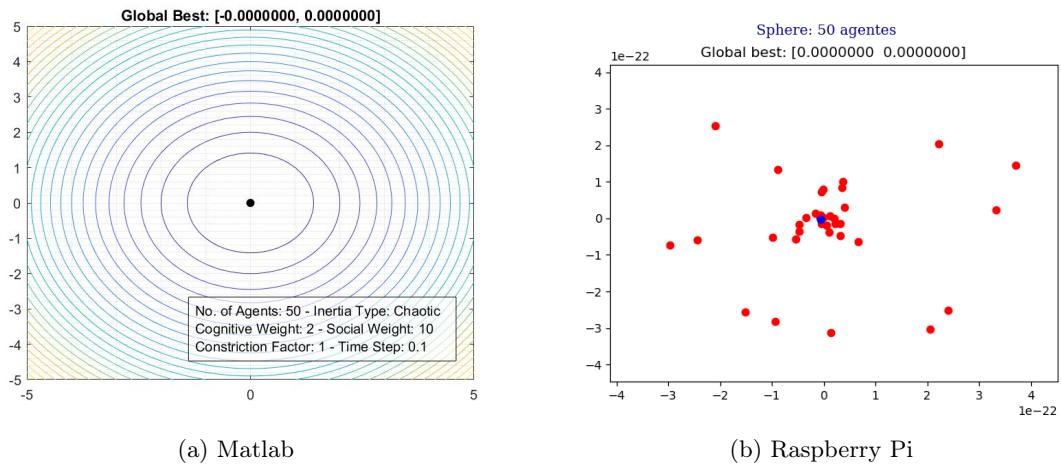


Figura 25: PSO con 50 agentes Inercia Caótica

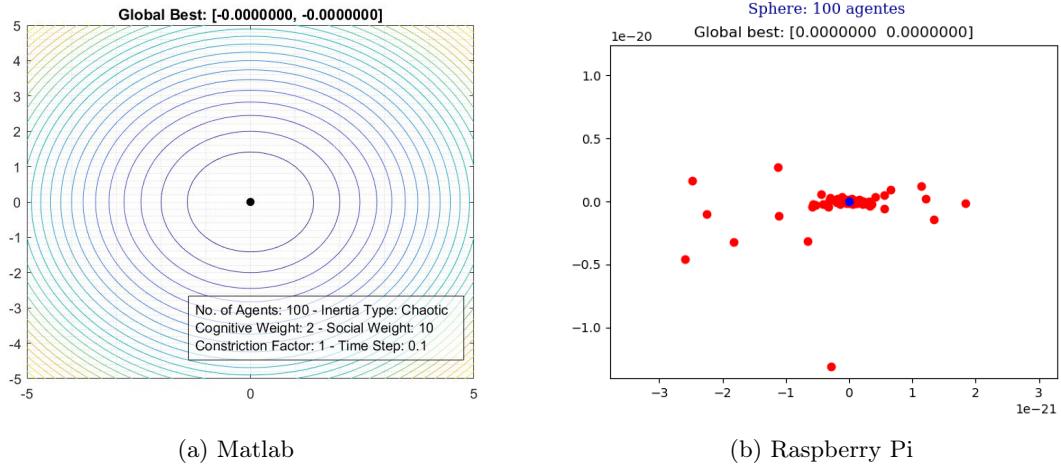


Figura 26: PSO con 100 agentes Inercia Caótica

No. de agentes	Matlab		Raspberry Pi		% de Error	
	X	Y	X	Y	X	Y
10	0.0000000	0.0000000	0.0000000	0.0000000	0 %	0 %
50	0.0000000	0.0000000	0.0000000	0.0000000	0 %	0 %
100	0.0000000	0.0000000	0.0000000	0.0000000	0 %	0 %

Cuadro 17: Resultados de la comparación del PSO matlab vs PSO *Raspberry Pi* con Inercia Caótica evaluando la función *Sphere*

### 8.1.2. Función *Himmelblau*

#### Inercia Constante

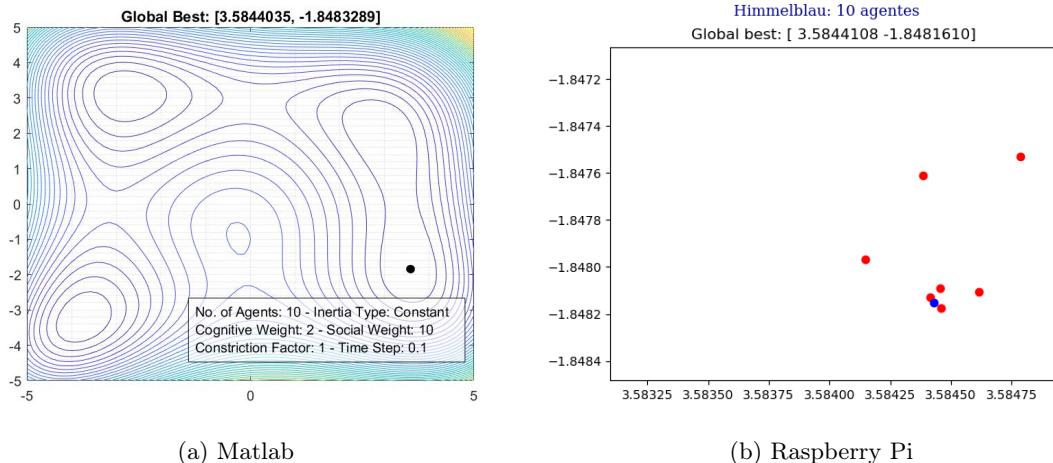


Figura 27: PSO con 10 agentes Inercia Constante

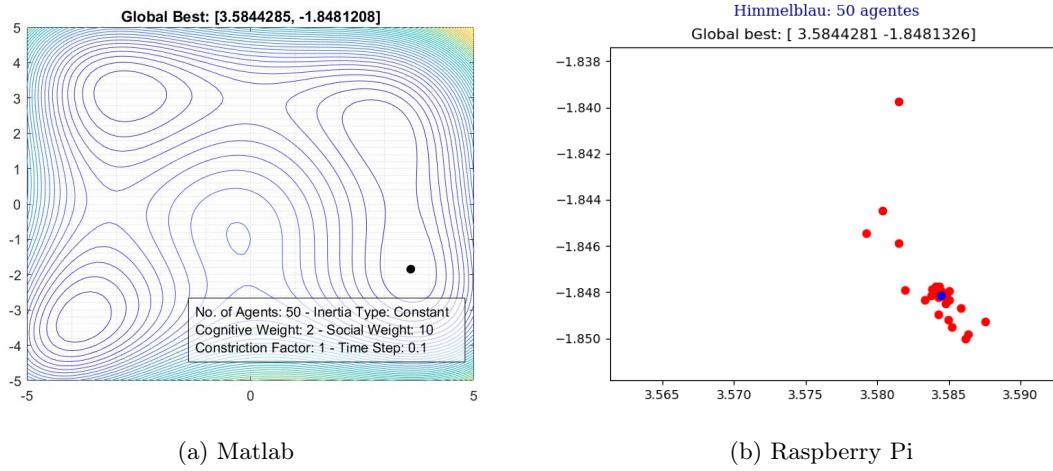


Figura 28: PSO con 50 agentes Inercia Constante

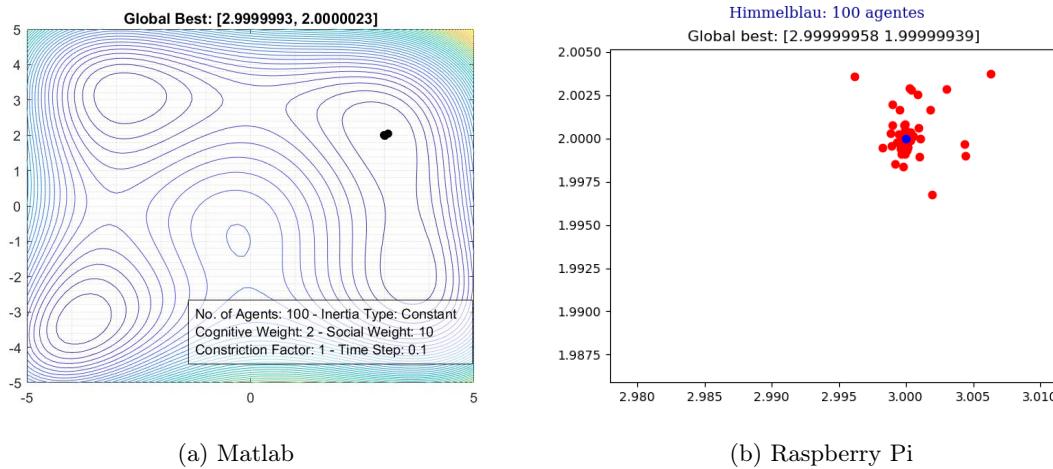


Figura 29: PSO con 100 agentes Inercia Constante

No. de agentes	Matlab		Raspberry Pi		% de Error	
	X	Y	X	Y	X	Y
10	3.5844035	-1.8483289	3.5844108	-1.8481610	0 %	0 %
50	3.5844285	-1.8481208	3.5844281	-1.8481326	0 %	0 %
100	2.9999993	2.0000023	2.9999998	1.9999994	0 %	0 %

Cuadro 18: Resultados de la comparación del PSO matlab vs PSO *Raspberry Pi* con Inercia Constante evaluando la función *Himmelblau*

## Inercia Caótica

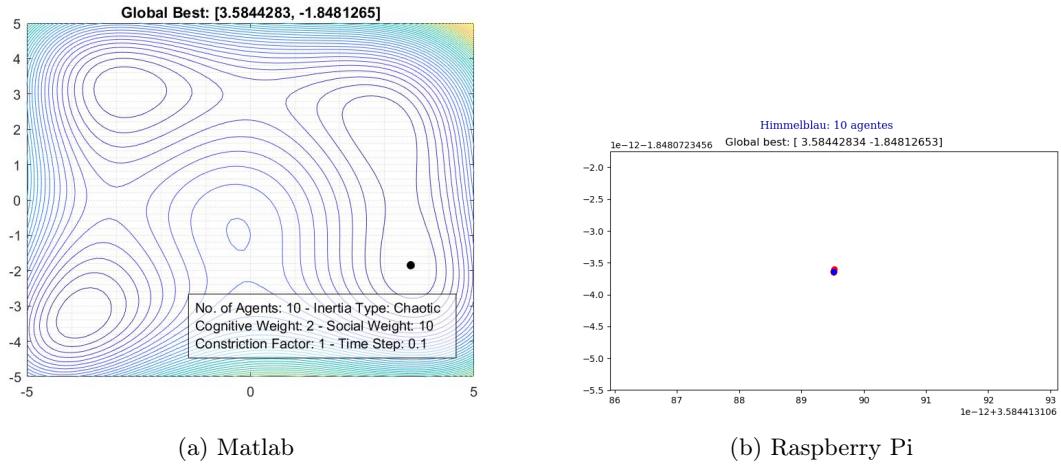


Figura 30: PSO con 10 agentes Inercia Caótica

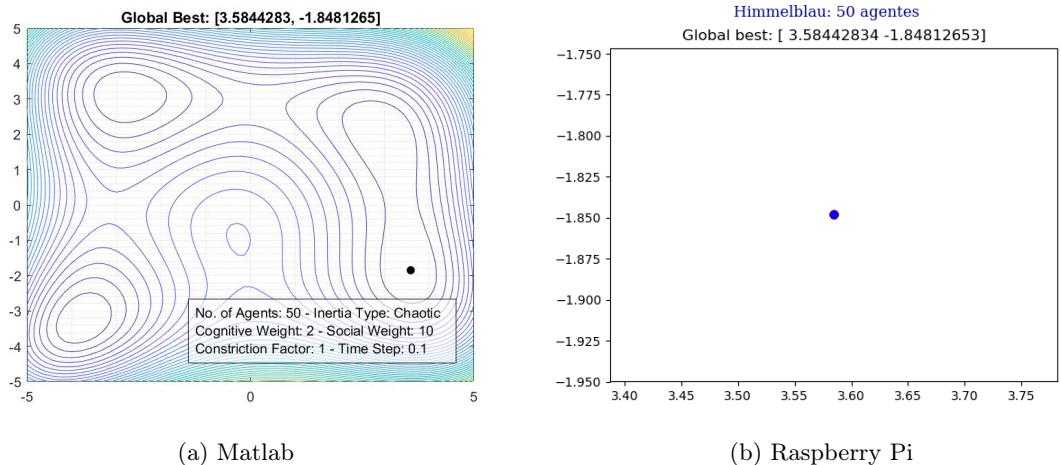


Figura 31: PSO con 50 agentes Inercia Caótica

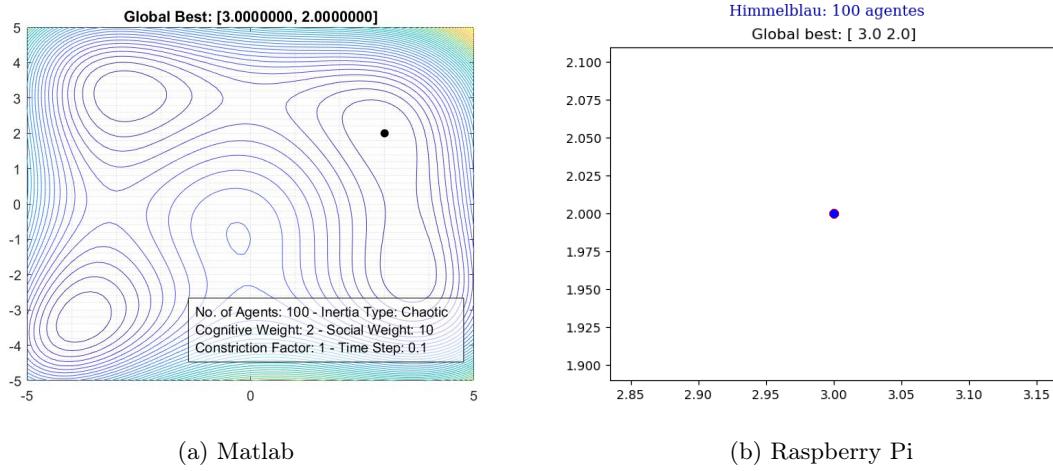


Figura 32: PSO con 100 agentes Inercia Caótica

No. de agentes	Matlab		Raspberry Pi		% de Error	
	X	Y	X	Y	X	Y
10	3.5844283	-1.8481265	3.5844283	-1.8481265	0 %	0 %
50	3.5844283	-1.8481265	3.5844283	-1.8481265	0 %	0 %
100	3.0000000	2.0000000	3.0000000	2.0000000	0 %	0 %

Cuadro 19: Resultados de la comparación del PSO matlab vs PSO *Raspberry Pi* con Inercia Caótica evaluando la función *Himmelblau*

## 8.2. PSO enfocado a Robots Móviles

Se implemento el algoritmo en cada dispositivo a utilizar en donde cada agente será capaz de ejecutar su propio MPSO. Cada agente tendrá la capacidad de recibir sus coordenadas y orientación actual así como comunicarse con otros agentes para tomar una decisión en conjunto y proceder con el cálculo de una nueva posición.

### 8.2.1. Creación de funciones costo

Se procedió a crear las diferentes funciones *benchmark* usadas normalmente para evaluar el correcto funcionamiento del PSO. Estas fueron: *Sphere*, *Rosenbrock* y *Himmelblau*.

El proceso de evaluación consistió en evaluar la posición de cada agente ( $X, Y$ ) en la función costo seleccionada. Este resultado es usado para determinar el costo de posición por cada agente. Durante la implementación se trabajara con 3 tipos de costo, los cuales son:

- Fitness actual: esta evaluación toma en consideración la posición actual de cada agente.
- Fitness local: esta evaluación toma en consideración la posición inicial de cada agente y es el costo que se comunica a los demás agentes. Este valor se actualiza si el *fitness*

actual posee un menor costo.

- Fitness global: esta evaluación toma en consideración la posición inicial de cada agente y solo se actualiza si el *fitness* local posee un menor costo (el propio o el enviado por otro agente).

### 8.2.2. Creación y ejecución de hilos de programación

Como se menciono en secciones previas, se usaron varias estrategias para validar el funcionamiento del PSO debido a la falta de una plataforma o robot móvil. Para estas estrategias se uso programación multihilos para que estas tareas puedan ser ejecutadas al mismo tiempo que el algoritmo PSO, esto sera mejor explicado en el capitulo 9. En la Figura 33 se observa el diagrama que ejemplifica este proceso.

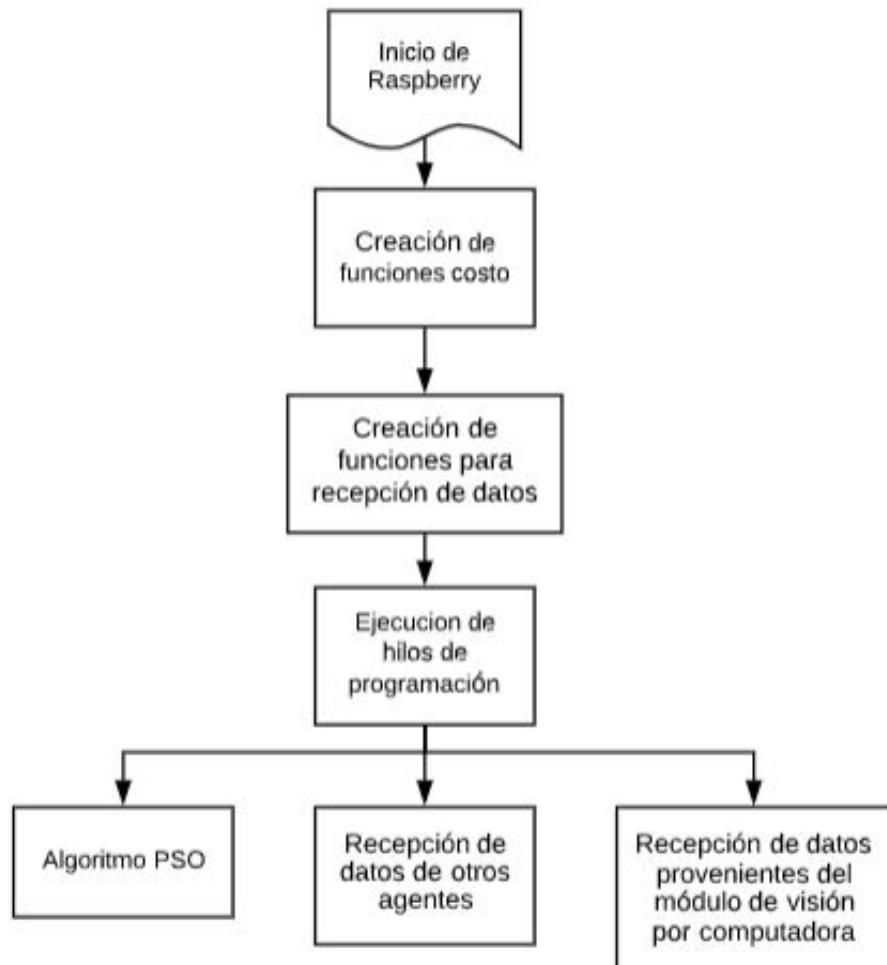


Figura 33: Ejecución de Hilos de Programación.

### **8.2.3. Configuración inicial del PSO**

Esta configuración se hace al inicio del programa y solo ocurre una vez, tiene como objetivo registrar la posición inicial de cada agente y asignar este valor al *best local* y *best global* así como evaluar la posición inicial para el calculo del costo del *fitness local* y *fitness global*.

### **8.2.4. Obtención de posición y orientación del robot**

Con la posición inicial ya establecida, se registra la posición y orientación actual en cada iteración del PSO. Para la obtención de las coordenadas y orientación de cada agente se usa un algoritmo de visión por computadora el cual fue desarrollado como parte de otro proyecto de tesis. En este algoritmo se implementa una interfaz de matlab para capturar la pose de marcadores los cuales simularan los agentes usados.

Se creo un socket usando una red local y un protocolo de comunicación UDP para enviar la información desde matlab a cada agente. Cada marcador es identificado con un numero específico para enviar las coordenadas y orientación a una IP especifica correspondiente al marcador para asegurar que cada agente reciba únicamente su posición X, Y y orientación respecto a la mesa de pruebas de la UVG.

### **8.2.5. Actualización del *local best* y *global best***

Con la posición actual de cada iteración ya registrada se calcula el valor actual de la función de costo seleccionada para determinar el costo actual y proceder con la actualización del *local best* y *global best*. Para esto se deben de poder comunicar entre si todos los agentes para el intercambio de información. La Figura 34 muestra un diagrama que ilustra el proceso de actualización de estos valores. Los cuadros en amarillo representan su ejecución en otro hilo de programación.

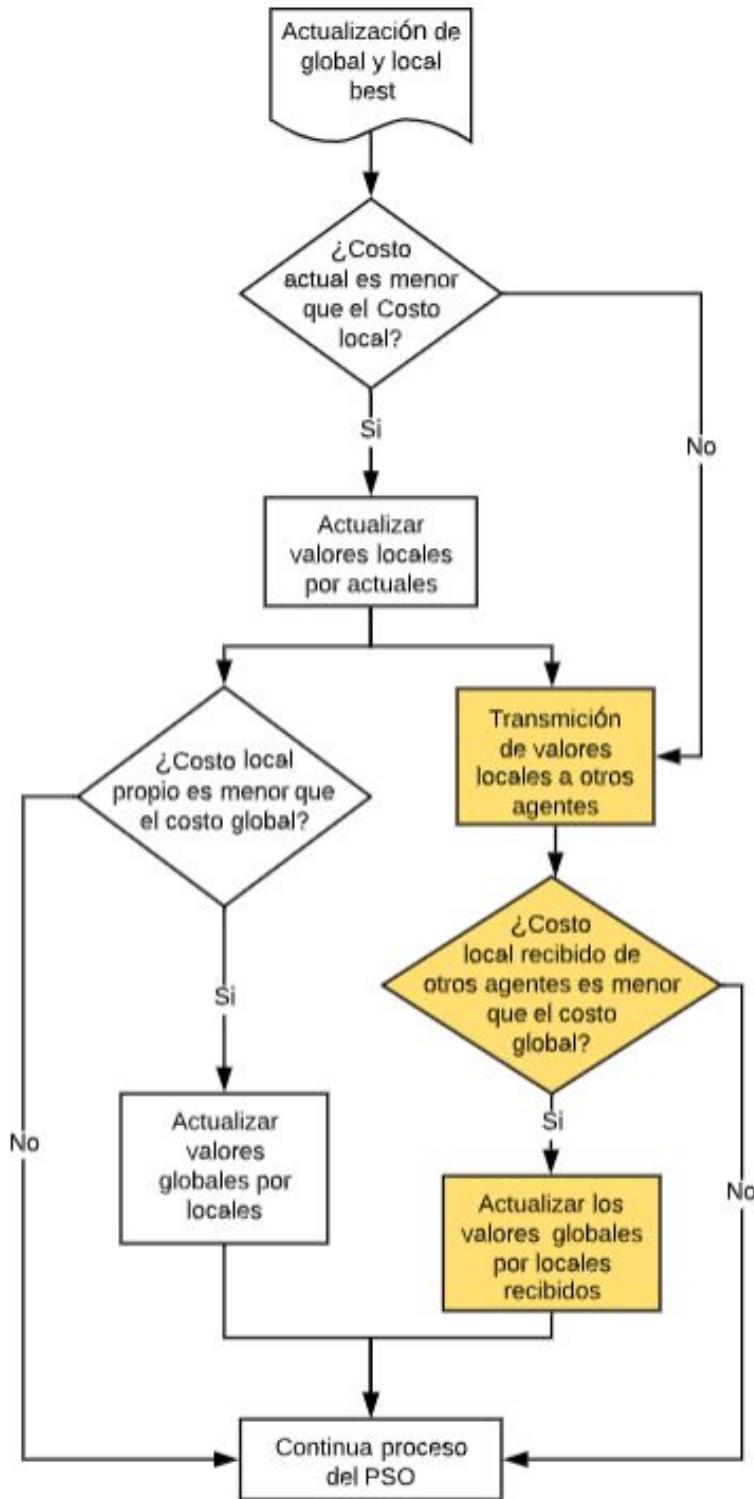


Figura 34: Actualización del local y global best.

### **8.2.6. Algoritmo PSO**

Para el cálculo de los parámetros de uniformidad, las ecuaciones del tipo de inercia a utilizar, el parámetro de restricción y el valor del V\_Scaler necesarios para la ejecución del algoritmo PSO, se hace uso de las ecuaciones desarrolladas en la fase II [1], si se desea una mayor descripción ver repositorios adjuntos de github. Se realiza el cálculo para una nueva velocidad (10) y una nueva posición (11) para cada agente.

### **8.2.7. Cálculo de la velocidad lineal y velocidad angular mediante varios tipos de controladores**

Para el cálculo de ambas velocidades se usaron distintos tipos de controladores descritos en el capítulo 6, así como los desarrollados en la fase II [1].

Los controladores usados son:

- Control proporcional de velocidades con saturación limitada
- Control PID de velocidad lineal y angular
- Control de pose
- Control de pose de Lyapunov
- Control Closed-Loop Steering
- Control por medio de Regulador Lineal Cuadrático (LQR)
- Controlador Lineal Cuadrático Integral (LQI)

### **8.2.8. Ajuste de velocidad de los motores**

Se usó el modelo diferencial (17) y (18) para calcular las velocidades que serán enviadas a los respectivos motores. Sin embargo estos valores deben ser evaluados para asegurarse que no sobrepasen el límite dado por los motores a usar en la plataforma móvil. Se establece como máxima velocidad posible al límite dado por la hoja de datos del motor a usar. Este proceso se describe en la Figura 35.



Figura 35: Proceso de truncar velocidades máximas.

Ambas velocidades son desplegadas en la ventada de comandos para darles una interpretación en espera de la plataforma móvil.

# CAPÍTULO 9

---

## Validación e Implementación física del PSO

---

Se crearon distintas pruebas para validar la correcta implementación del algoritmo PSO en un sistema físico. En este capítulo estas son explicadas y se interpretan sus resultados.

### 9.1. Programación Multihilos

Como se menciona en el capítulo 6, la programación multihilos permite la ejecución de varias tareas de manera simultanea. Esta estrategia se uso para ejecutar el algoritmo PSO, intercambiar información entre agentes y recibir información del algoritmo de visión por computadora. .

#### 9.1.1. Recepción del modulo de visión por computadora

Como parte de otro proyecto de graduación se continuo el trabajo desarrollado en [38], donde se elaboró un algoritmo de visión por computadora para el reconocimiento de la pose de agentes. Este algoritmo cuenta con una versión en matlab, python y C; se usó un protocolo de comunicación UDP para que fuera posible enviar las coordenadas y orientación en tiempo real de marcadores (simulando agentes) colocados en la mesa de pruebas de la UVG y que los agentes (dispositivos *Raspberry Pi*) sean capaces de recibir esta información en un lenguaje C.

El proceso consistió en la creación de sockets de comunicación donde el algoritmo de visión por computadora (aplicación desarrollada en matlab) cumpla la función de cliente mientras los agentes cumplen la función de servidores. De esta forma es posible enviar diferente información a cada agente al incluir la dirección de destino, siendo la IP de cada *Raspberry Pi*.

La primera prueba para validar este proceso de comunicación fue la creación de varias cadenas de caracteres en matlab las cuales fueron enviadas a direcciones específicas. En la Figura 36 se puede ver como a los cuatro agentes les llega un mensaje distinto.

The figure displays four terminal windows from a Linux system (Ubuntu 4.19.66-v7+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l) showing the interaction between a Matlab server and four agents (Agent 1, Agent 2, Agent 3, and Agent 4). The server sends a message to each agent, which then prints it to the terminal. The messages are as follows:

- Agent 1:** Mensaje recibido: Hola Agente 1
- Agent 2:** Mensaje recibido: Hola Agente 2
- Agent 3:** Mensaje recibido: Hola Agente 3
- Agent 4:** Mensaje recibido: Hola Agente 4

```

pi@IEUVG: ~/Documents
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: wed Oct 27 13:06:59 2021

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@IEUVG:~$ cd Documents
pi@IEUVG:~/Documents $ gcc serverudp.c
pi@IEUVG:~/Documents $ gcc serverudp.c -o server
pi@IEUVG:~/Documents $ ./server
Mensaje recibido: Hola Agente 1

pi@IEUVG: ~/Desktop
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.0.5' (ED25519) to the list of known hosts.
Linux IEUVG 4.19.66-v7+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Oct 12 17:20:04 2021
pi@IEUVG:~$ cd Desktop
pi@IEUVG:~/Desktop $ ls
agentes pruebas4 serverudp.c
pi@IEUVG:~/Desktop $ gcc serverudp.c
pi@IEUVG:~/Desktop $ gcc serverudp.c -o server
pi@IEUVG:~/Desktop $ ./server
Mensaje recibido: Hola Agente 3

pi@IEUVG: ~/Desktop
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.0.6' (ED25519) to the list of known hosts.
Linux IEUVG 4.19.66-v7+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Oct 12 17:10:28 2021
pi@IEUVG:~$ cd Desktop
pi@IEUVG:~/Desktop $ ls
agentes pruebas4 serverudp.c
pi@IEUVG:~/Desktop $ gcc serverudp.c
pi@IEUVG:~/Desktop $ gcc serverudp.c -o server
pi@IEUVG:~/Desktop $ ./server
Mensaje recibido: Hola Agente 4

```

Figura 36: Prueba 1 de la comunicación Matlab con C.

La segunda prueba fue el envío de valores numéricos en un formato específico. Para un correcto funcionamiento del PSO es necesario brindarle la información actual de cada agente (coordenada X, Y y orientación), esta información es ordenada gracias al algoritmo de visión por computadora de forma específica antes de ser enviada. Cuando la información es recibida primero se descompone y luego se almacenada para su uso en el algoritmo PSO. El orden consiste: *coordenada X, coordenada Y, orientación*; cada dato es separado por una coma.

De igual forma se envió un mensaje específico a cada dirección, en la Figura 37 se pueden ver los valores numéricos recibidos por cada agente.

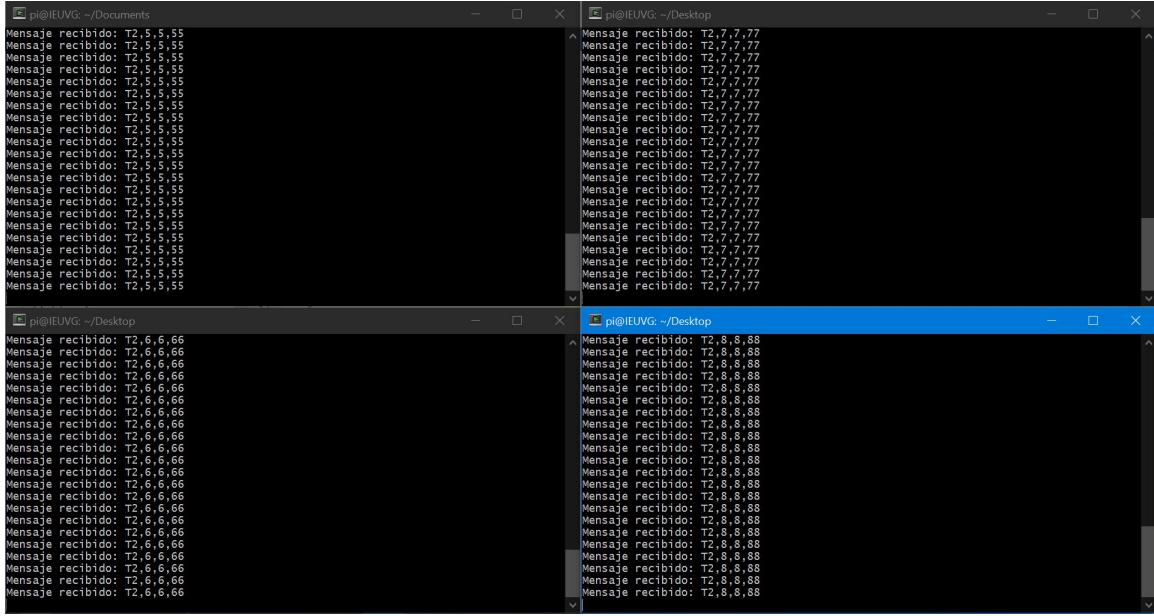


Figura 37: Prueba 2 de la comunicación Matlab con C.

Con estas pruebas se validó el protocolo de comunicación encargado de transferir la pose de los agentes desde algoritmo de visión por computadora hacia los dispositivos *Raspberry Pi*. Para este proceso se usó programación multihilos, el único propósito de este hilo de programación es la constante recepción de los datos provenientes de matlab.

### 9.1.2. Envío y recepción de información entre agentes

Una parte fundamental del correcto funcionamiento del PSO es el poder comunicar la posición actual así como el costo actual a los otros agentes para tomar una decisión sobre el recorrido a tomar. El intercambio de información debe permitir a todos los agentes enviar tanto como recibir datos, con esto en cuenta una comunicación del tipo cliente-servidor no sería eficiente entre agentes. Se uso un protocolo de comunicación UDP del tipo *broadcast* en la cual todos los agentes conectados a la red pueden enviar y recibir la misma información.

Como se puede ver en la Figura 38 los N agentes que estén conectados a la red local serán capaces de enviar y recibir información al mismo tiempo. Para facilidad de interpretación se crearon 2 variables, una para el envío: *buffer enviar* y otra para la recepción: *buffer recibir*.

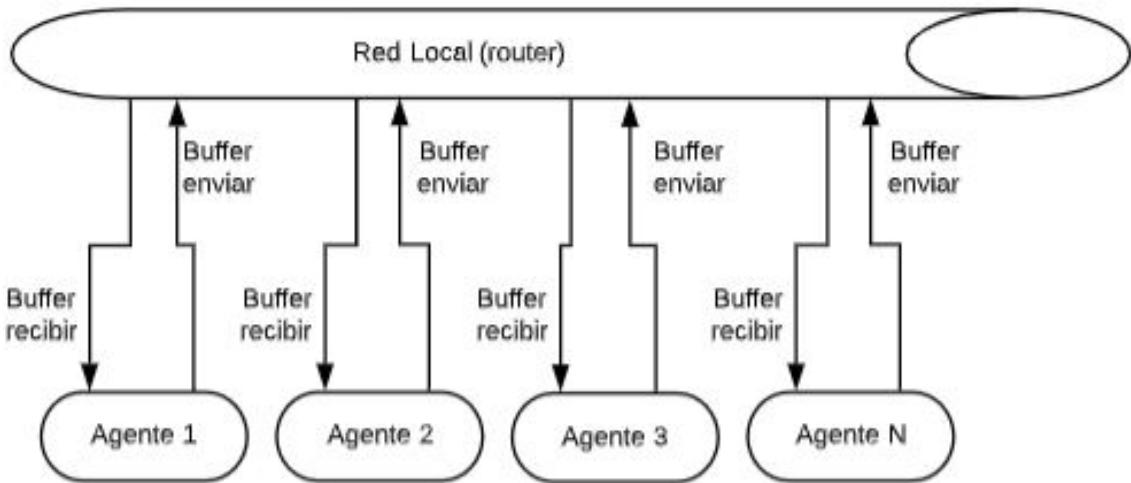


Figura 38: Representación del envío y recepción de datos entre agentes.

La información a intercambiar es ordenada de forma conveniente antes de ser enviada, los datos enviados corresponden a los valores locales de posición para X y Y además del costo local que se tiene por cada posición del agente. Igual que el caso anterior cuando la información es recibida primero se descompone y luego se almacena para su uso en el PSO. El orden consiste en: *mejor posición local X, mejor posición local Y, costo local*; cada dato es separado por una coma.

Para este proceso se usa programación multihilos, el único propósito de este hilo de programación es recibir la información de otros agentes y comparar el costo global propio con el costo local recibido por parte de los otros agentes. El envío y ordenamiento de la información se realiza en el programa principal, mismo donde se ejecuta el resto del algoritmo PSO.

En la Figura 39 se puede ejemplificar este proceso. Los cuadros en amarillo representan su ejecución en otro hilo de programación.

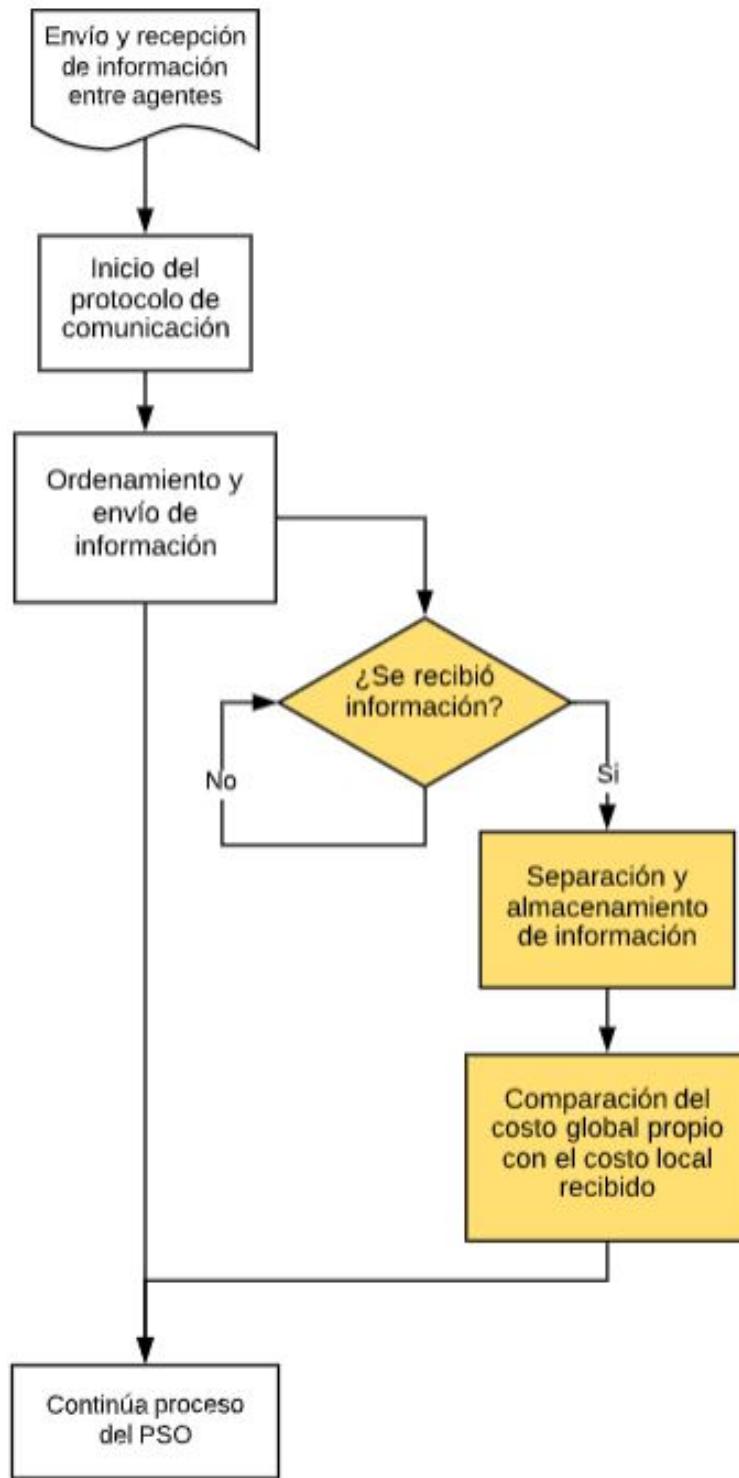


Figura 39: Diagrama de envío y recepción de información entre agentes.

Para validar el protocolo de comunicación y la programación multihilos se elaboró un código sencillo, el cual tiene como objetivo ordenar la mejor posición local X, Y y el mejor costo local (el cual es introducido manualmente junto con el número de agente). En el código principal se establece el protocolo de comunicación, se envía la información y se crea el hilo de programación que se encarga de recibir la información de todos los agentes conectados a la red local y comparar el costo global (presente en cada agente) con el costo local (enviado por cada agente) para tomar una decisión sobre la mejor posición.

Para esta prueba se usaron 8 dispositivos, cada uno con un costo local y global diferente. Los resultados se pueden en el Cuadro 20.

Agente	costo local inicial	costo global inicial	costo global final
1	1.0	1.0	1.0
2	2.0	2.0	1.0
3	3.0	3.0	1.0
4	4.0	4.0	1.0
5	5.0	5.0	1.0
6	6.0	6.0	1.0
7	7.0	7.0	1.0
8	8.0	8.0	1.0

Cuadro 20: Prueba de convergencia de agentes

Para lograr visualizar el cambio, se imprime en la consola el numero de agente y si este logra actualizar el valor global o no. En las Figuras 40 y 41 se observan los resultados de esta pruebas.

Figura 40: Pruebas del envío y recepción de datos 1.

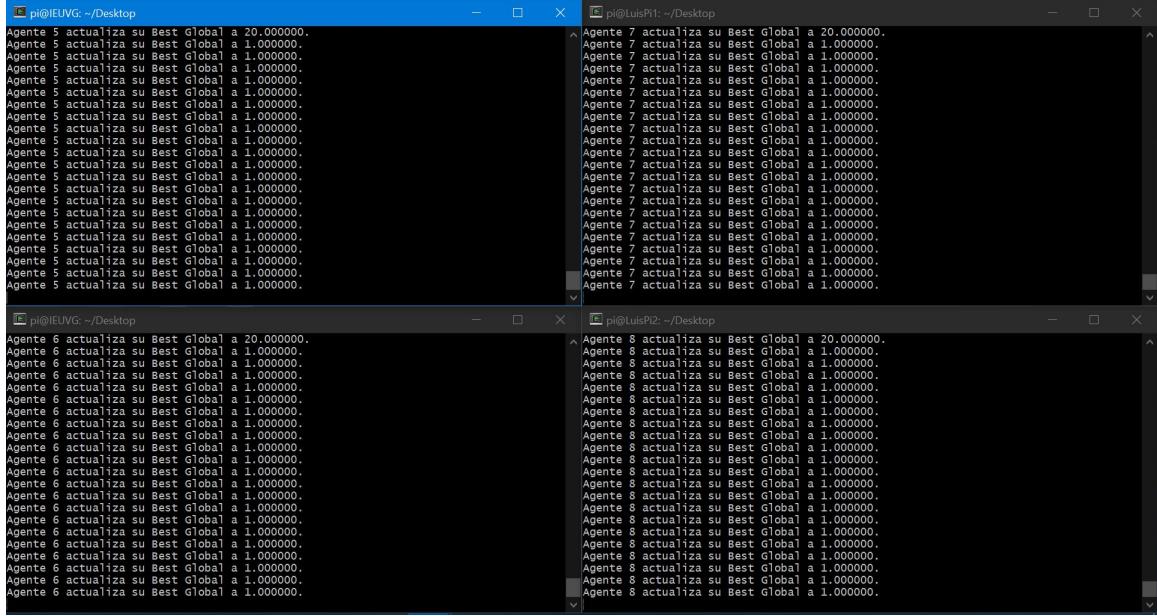


Figura 41: Pruebas del envío y recepción de datos 2.

Los 8 agentes lograron enviar su información para luego ser procesada e identificar el menor costo disponible. Como se puede ver en el Cuadro 20 este costo corresponde al agente 1 con un costo global de 1.0, valor que lograron identificar todos los agentes.

### 9.1.3. Combinación de ambos protocolos y programación multihilos

Se combinaron ambos protocolos de comunicación desarrollados anteriormente, cada uno cuenta con su propio hilo de programación encargado de la recepción de información. Para esta validación se colocaron 4 marcadores en la mesa de pruebas, uno por cada dispositivo *Raspberry Pi*. El objetivo de la prueba consistió en recibir la información de los marcadores provenientes del algoritmo de visión por computadora y al mismo tiempo intercambiar información entre los 4 agentes para converger al menor costo detectado.

En la Figura 42 se puede ver el resultado del algoritmo de visión por computadora, la interfaz y los valores correspondientes de los agentes usados en esta prueba.

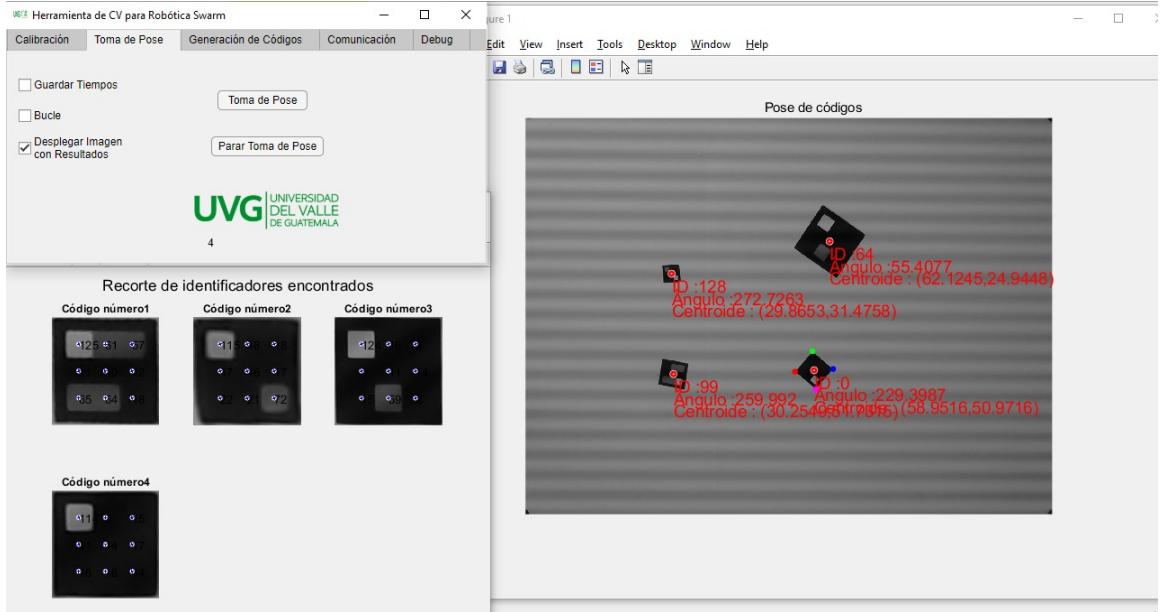


Figura 42: Poses detectadas de 4 agentes.

En la Figura 43 se observa como los 4 agentes logran converger al menor costo detectando y al mismo tiempo recibir específicamente su pose actual. De esta forma se logró validar los dos hilos de programación.

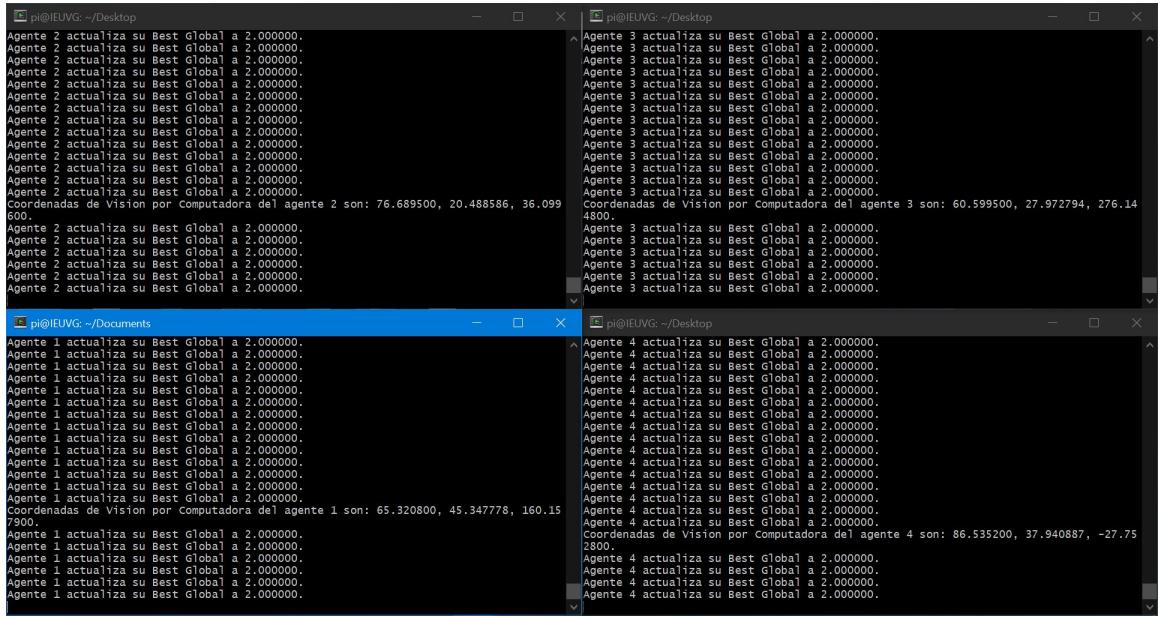


Figura 43: Validación de programación multihilos.

## 9.2. Pruebas del algoritmo PSO

La validación del algoritmo PSO en un sistema físico se realizó de dos formas, la primera fue una comparación directa con los resultados del mismo algoritmo evaluado en webots en fases anteriores y la segunda fue una demostración usando la mesa de pruebas de la UVG.

### 9.2.1. Ambientes controlados

Para la primera validación se usó webots para obtener las coordenadas y orientación de cada agente, se usó un total de siete agentes para las distintas pruebas bajo ambientes controlados. Para tener una comparación valida entre ambos algoritmos se trabajó bajo las mismas métricas, las cuales fueron: posición inicial, cantidad de agentes, función costo, tipo de controlador, velocidad de los motores, dimensiones físicas, tiempo de muestreo, inercia y demás parámetros internos del PSO.

La pose de los agentes con respecto a webots se almaceno en un archivo de texto el cual se ordena y posteriormente se envía a cada dirección específica mediante matlab. Con esto se obtiene la trayectoria esperada que deben de realizar los agentes y permite realizar una comparación valida con los resultados obtenidos en fases anteriores.

#### Primera validación del algoritmo

Se probaron distintas combinaciones de funciones costo y parámetros de inercia, los cuales permiten verificar el costo actual de cada agente y las nuevas posiciones dada por el algoritmo PSO. Se probaron varios tipos de controladores para verificar que el valor enviado a los motores (en una futura plataforma móvil) haga sentido con relación a las nuevas posiciones previamente calculadas.

Para la primera prueba se uso un controlador PID (ecuación 25), la función costo *Shpere* y una inercia exponencial (ecuación 9).

La segunda prueba se uso un controlador LQR (ecuación 37), la función costo *Himmelblau* y una inercia linear (ecuación 3).

La tercera prueba se uso un controlador LQI (ecuación 37), la función costo *Rosenbrock* y una inercia constante (ecuación 2).

La cuarta prueba se uso un controlador de pose de Lyapunov (ecuaciones 33 y 34), la función costo *Shpere* y una inercia caótica (ecuación 6).

La quinta prueba se uso un controlador de pose (ecuaciones 31 y 32), la función costo *Rosenbrock* y una inercia random (ecuación 8).

	Webots		Raspberry		% Error	
	Nuevas posiciones					
iteración	X_w	Y_w	X_r	Y_r	X	Y
1	0.141206	0.218473	0.14119	0.21849	0.000 %	0.000 %
2	0.049871	0.163887	0.021058	0.146722	0.041 %	0.025 %
3	-0.032152	0.11489	0.027807	0.126922	0.086 %	0.017 %
4	-0.013096	0.126416	0.030789	0.109509	0.063 %	0.024 %

Cuadro 21: Resultados de la quinta prueba para el agente 7 primeras iteraciones

	Webots		Raspberry		% Error	
	Nuevas posiciones					
iteración	X_w	Y_w	X_r	Y_r	X	Y
2038	0.37453	0.163496	0.375238	0.163597	0.001 %	0.000 %
2039	0.374411	0.163480	0.374498	0.163492	0.000 %	0.000 %
2040	0.37428	0.163462	0.374293	0.163463	0.000 %	0.000 %
2041	0.374204	0.16345	0.374241	0.163452	0.000 %	0.000 %

Cuadro 22: Resultados de la quinta prueba para el agente 7 iteraciones finales

### 9.2.2. Mesa de Pruebas

Para la segunda validación se usó el algoritmo de visión por computadora para obtener las coordenadas y orientación de cada gente, se usó un total de cuatro agentes.

# CAPÍTULO 10

---

## Conclusiones

---

- El ordenador *Raspberry Pi* es la mejor alternativa para realizar la implementación del algoritmo PSO en un sistema físico.
- Todos los datos enviados por medio de la red local son recibidos exitosamente por los agentes conectados a la misma red.
- El proceso de ordenar, enviar, recibir y descomponer la información de cada agente es comprobado al verificar la convergencia de todos los agentes a las coordenadas del agente que posee el mejor costo.
- Los resultados obtenidos de la comparación directa entre webots y el ordenador Raspberry Pi en cuanto al cálculo de nuevas posiciones dadas por el PSO son aceptadas en base a los porcentajes de error y graficas generadas, por lo que se obtiene una implementación exitosa del algoritmo PSO en sistemas físicos.
- Los resultados obtenidos en la mesa de pruebas nos indican un correcto movimiento de los agentes colocados sobre la mesa, el algoritmo PSO en sistemas físicos se adapta a las posiciones iniciales de los marcadores y realiza un calculo de nuevas posiciones que hace sentido de acuerdo con la posición actual de los marcadores.

# CAPÍTULO 11

---

## Recomendaciones

---

- Se recomienda realizar implementación con algún otro tipo de robot móvil para hacer uso de su sistema GPS integrado y sus protocolos de comunicación entre agentes.
- Se recomienda probar otros protocolos comunicación entre los agentes para verificar que la información sea recibida por todos los agentes en uso.
- Integrar el sistema de captura de movimiento que se tiene en el laboratorio de robótica del CIT para tomar la pose de los agentes.

## CAPÍTULO 12

---

### Bibliografía

---

- [1] A. S. A. Nadalini, “Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO),” Tesis de licenciatura, Universidad del Valle de Guatemala, 2019.
- [2] E. A. S. Olivet, “Aprendizaje Reforzado y Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre,” Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.
- [3] Jason Maderer, *Robotarium: A Robotics Lab Accessible to All*, <https://www.news.gatech.edu/features/robotarium-robotics-lab-accessible-all>, Accessed: 2021-03-27, 2017.
- [4] G. I. Colmenares, “Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica,” Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.
- [5] L. S. Tortosa, “Ajentes y enjambres artificiales: modelado y comportamientos para sistemas de enjambre robóticos,” phdthesis, Universidad de Alicante, España, 2013.
- [6] R. F. R. Grandi **and** C. Melchiorri, *A Navigation Strategy for Multi-Robot Systems Based on Particle Swarm Optimization Techniques*. Dubrovnik, Croatia: Dubrovnik, 2012.
- [7] C. Duarte **and** C. J. Quiroga, *PSO algorithm*. Ciudad Universitaria, Santander, Colombia: Santander, 2010.
- [8] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon **and** A. Abraham, “Inertia Weight strategies in Particle Swarm Optimization,” **in** *2011 Third World Congress on Nature and Biologically Inspired Computing*, 2011, **pages** 633–640.
- [9] Y. Shi **and** R. Eberhart, “A modified particle swarm optimizer,” **in** *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 1998, **pages** 69–73.
- [10] D. Bingham and S. Surjanovic, *Virtual Library of Simulation Experiment*, <https://www.sfu.ca/~ssurjano/spheref.html>, Accessed: 2021-08-13, 2013.
- [11] D. Bingham, *Project Homepage DEAP*, <https://deap.readthedocs.io/en/master/api/benchmarks.html#deap.benchmarks.sphere>, Accessed: 2021-09-13, 2021.

- [12] C. CABALLERO GONZÁLEZ, *Programación con lenguajes de guión en páginas web*. S.A.: Ediciones Paraninfo, 2015.
- [13] J. Lajara **and** J. Pelegri, *LabVIEW: Entorno gráfico de programación*. S.A.: Marcombo, 2011.
- [14] J. F. Kurose **and** K. W. Ross, *Redes de computadoras Un enfoque descendente*. Madrid: PEARSON EDUCACIÓN, S. A., 2017.
- [15] R. MÓVILES, G. BERMÚDEZ, Universidad Distrital Francisco José de Caldas. Colombia, **december** 2001.
- [16] A. M. J. Valencia **and** L. Rios, *MODELO CINEMÁTICO DE UN ROBOT MÓVIL TIPO DIFERENCIAL Y NAVEGACIÓN A PARTIR DE LA ESTIMACIÓN ODOMÉTRICA*, Universidad Tecnológica de Pereira, **may** 2009.
- [17] M. Egerstedt, *Control of Mobile Robots, Introduction to Controls*, Georgia Institute of Technology, **may** 2014.
- [18] Raspberry Pi, *Raspberry Pi*, <https://www.raspberrypi.org>, Accessed: 2021-06-04, 2021.
- [19] ———, *Raspberry Pi OS*, <https://www.raspberrypi.org/software/>, Accessed: 2021-06-04, 2021.
- [20] ———, *Raspberry Pi Products*, <https://www.raspberrypi.org/products/>, Accessed: 2021-06-04, 2021.
- [21] A. C. Estrada, *Tipos y fabricantes de Microcontroladores*, Instituto Tecnológico de Estudios Superiores de Uruapan, 2013.
- [22] S. Caprile, *Desarrollo con microcontroladores ARM Cortex-M3*. Buenos Aires: Punto-libro, 2012.
- [23] S. E. Tropea **and** D. M. Caruso, *MICROCONTROLADOR COMPATIBLE CON AVR, INTERFAZ DE DEPURACIÓN Y BUS WISHBONE*, Instituto Nacional de Tecnología Industrial Buenos Aires, Argentina, 2015.
- [24] F. Valdes **and** R. Arely, *Microcontroladores Fundamentos y Aplicaciones con PIC*. España: Marcombo, 2007.
- [25] Arduino, *Arduino Uno*, <https://arduino.cc/arduino-uno/>, Accessed: 2021-09-24, 2021.
- [26] T. Instruments, *Tiva C Series TM4C123G LaunchPad Evaluation Board*, Texas Instruments, 2013.
- [27] K-Team, *KILOBOT*, <https://www.k-team.com/mobile-robotics-products/kilobot>, Accessed: 2021-02-25, 2017.
- [28] GCtronic, *e-puck education robot*, <http://www.e-puck.org>, Accessed: 2021-03-27, 2018.
- [29] University of York, *Pi-puck*, <https://pi-puck.readthedocs.io/en/latest/>, Accessed: 2021-06-04, 2020.
- [30] M. T. Inc, *PIC16F87XA Data Sheet*, Microchip Technology Inc, **september** 2013.
- [31] ———, *General Purpose, 16-Bit Flash Microcontrollers with XLP Technology Data Sheet*, Microchip Technology Inc, **september** 2013.

- [32] ——, *PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Family*, Microchip Technology Inc, **september** 2013.
- [33] C. E. Barcelona, *¿Cuántos lenguajes de programación existen?* Epitech, 2021.
- [34] P. J. D. Harvey M. Deitel, *Como programar en C/C++y Java*. México: Pearson Educación, 2004.
- [35] S. Chazallet, *Python 3: los fundamentos del lenguaje*. España: Ediciones ENI, 2016.
- [36] O. Reinoso **and** L. Jimenes, *MATLAB: conceptos básicos y descripción gráfica*. España: Universidad Miguel Hernandez del Elche, 2018.
- [37] Microsoft, *Visual Studio Code*, <https://code.visualstudio.com>, Accessed: 2021-09-20, 2021.
- [38] J. P. G. Jordán, “Algoritmos de Visión por Computadora para el Reconocimiento de la Pose de Agentes Empleando Programación Orientada a Objetos y Multihilos,” Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.
- [39] Cyberbotics, *Webots*, <https://cyberbotics.com>, Accessed: 2021-08-25, 2021.

# CAPÍTULO 13

---

## Anexos

---

### 13.1. Código

Creación de funciones Costo:

```
// Funciones costo
double funcion(double x, double y){
    double f=0;
    if (funcion_costo==0){
        f=pow(x,2)+pow(y,2);                                // Sphere
    }else if (funcion_costo==1){
        f=pow(1-x,2)+100*pow(y-pow(x,2),2);              // Rosenbrock
    }else if (funcion_costo==2){
        f=pow(x+2*y-7,2)+pow(2*x+y-5,2);                // Booth
    }else if(funcion_costo==3){
        f=pow(x*x+y-11,2)+pow(x+y*y-7,2);              // Himmelblau
    }
    return f;
}
```

Creación de función receiving:

```
void *receiving(void *ptr)
{
    int *sock, n;
    int i = 0;
    sock = (int *)ptr;                                     // socket identifier
    unsigned int length = sizeof(struct sockaddr_in); // size of structure
    struct sockaddr_in from;
```

```

while (1){
    memset(buffer_recibir, 0, MSG_SIZE); // "limpia" el buffer
    // receive message
    n = recvfrom(*sock, buffer_recibir, MSG_SIZE, 0,
    (struct sockaddr *)&from, &length);
    if (n < 0){
        error("Error: recvfrom");
    }
    i = 0;
    //----- Se descompone el buffer_recibir-----
    token = strtok(buffer_recibir, ",");
    recepcion[i] = atof(token);
    while ((token = strtok(NULL, ",")) != NULL){
        i++;
        recepcion[i] = atof(token);
    }
    // -----Actualizar global best -----
    if (recepcion[2] < fitness_global){
        printf("Global actualizado.\n");
        best_global[0] = recepcion[0];
        best_global[1] = recepcion[1];
        fitness_global = recepcion[2];
    }
    else{
        printf("No se actualiza el Global.\n");
    }
}
pthread_exit(0);
}

```

Creación de función gps:

```

void *gps(void *ptr){
    // Open the communication
    gps_init();
    loc_t data;
    while (1) {
        gps_location(&data);
        // Visualizacion de la informacion
        printf("\%lf \%lf\n", data.latitude, data.longitude);
        posicion_robot[2]= data.longitude;
        posicion_robot[0]= data.latitude;
        rad = data.course;
    }
    pthread_exit(0);
}

```

# CAPÍTULO 14

---

## Glosario

---

**Visual Studio Code** Visual Studio Code es un editor de código fuente que permite trabajar con diversos lenguajes de programación, admite gestionar tus propios atajos de teclado y refactorizar el código [37]. 41

**Webots** Es un simulador de robots 3D gratuito y de código abierto, usado para simular robots móviles se suele utilizar con fines educativos [39].. 4