

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

## **Implementation and Validation of the Particle Swarm Optimization Robotics Algorithm in Physical Systems**

Alex Daniel Maas Esquivel (maa17146@uvg.edu.gt)

Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica, Facultad de Ingeniería, Universidad Del Valle de Guatemala

### **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. Para tener una trayectoria suave y controlada tenemos el *Modified Particle Swarm Optimization* (MPSO) el cual toma en cuenta las dimensiones físicas del robot e implementa una serie de controladores buscando un movimiento controlado. Se buscó implementar y validar este algoritmo que funciona bien a nivel de simulación en un sistema físico el cual sea capaz de obtener resultados similares. Al no contar con una plataforma móvil operativa se desarrollan distintas técnicas para lograr la validación del algoritmo bajo ambientes controlados. Se llevan a cabo dos pruebas de validación para una correcta implementación. La primera consiste en una comparación directa con los resultados obtenidos a nivel de simulación haciendo uso del software webots. La segunda se llevó a cabo en la mesa de pruebas de la UVG donde el algoritmo indica las nuevas posiciones calculadas y de forma manual son seguidas por marcadores buscando una validación en tiempo real. Los resultados obtenidos en la primera prueba son aceptados con base a los porcentajes de error menores al 1%. Los resultados obtenidos en la mesa de pruebas son aceptados al obtener direcciones de movimiento esperadas según el conocimiento que se tiene de la meta.

**Palabras clave:** robótica de enjambre, PSO, iteraciones, punto de convergencia.

### **Abstract**

Swarm robotics employs patterns of collective behavior through interactions between robots and robots with their environment to achieve set goals or objectives. One of the commonly used swarm robotics algorithms is Particle Swarm Optimization (PSO), which is used as a trajectory planner. To have a smooth and controlled trajectory we have the Modified Particle Swarm Optimization (MPSO) which considers the physical dimensions of the robot and implements a series of controllers looking for a controlled movement.

We sought to implement and validate this algorithm that works well at the simulation level in a physical system which can obtain similar results. By not having an operating mobile platform, different techniques are developed to achieve algorithm validation under controlled environments. Two validation tests are carried out for a correct implementation. The first consists of a direct comparison with the results obtained at the simulation level using the webots software. The second was carried out on the UVG test table where the algorithm indicates the new calculated positions, and they are manually followed by markers seeking real-time validation. The results obtained in the first test are accepted based on the percentages of error less than 1%. The results obtained in the test table are accepted by obtaining expected directions of movement according to the knowledge of the goal.

**Keywords:** swarm robotics, PSO, iterations, convergence point.

## Introducción

La robótica de enjambre o *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 (Tortos 2013). El algoritmo PSO pertenece a las técnicas denominadas optimización inteligente y se clasifica como un algoritmo estocástico de optimización basado en población.

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 (Duarte y Quiroga 2010). 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* (Grandi y Melchiorri 2012).

Con estos datos es posible calcular los dos primeros factores principales de la ecuación del PSO

$$F_{cognitivo} = P_{local} - P_i \quad (1)$$

$$F_{social} = P_{global} - P_i \quad (2)$$

Donde  $P_i$  representa cada una de las soluciones/posiciones actuales en las ecuaciones 1 y 2. Para verificar que tan buena es la posición actual para cada partícula se usa la denominada función costo o función *fitness* 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 (Duarte y Quiroga 2010).

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  $C2$  en la ecuación 3. El factor de uniformidad corresponde a dos números aleatorios que van entre 0 y 1; se definen como  $r1$  y  $r2$  en la ecuación 3. El factor de constricción  $\varphi$  el cual se encarga de controlar la longitud de los pasos que cada partícula puede dar en cada iteración. El factor de inercia  $\omega$  el cual se encarga de controlar cuánta memoria puede almacenar cada partícula (Duarte y Quiroga 2010).

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

$$V_{i+1} = \varphi[\omega V_i + C1r1(F_{cognitivo}) + C2r2(F_{social})] \quad (3)$$

Donde  $V_i$  representa la velocidad actual 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

$$X_{i+1} = X_i + V_{i+1}\Delta t \quad (4)$$

Donde  $X$  representa la posición actual y  $X_{i+1}$  la nueva posición calculada para la partícula,  $\Delta t$  es el tiempo que le toma al algoritmo realizar cada iteración.

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, este fue denominado *Modified Particle Swarm Optimization* (MPSO) (Nadalini, 2020).

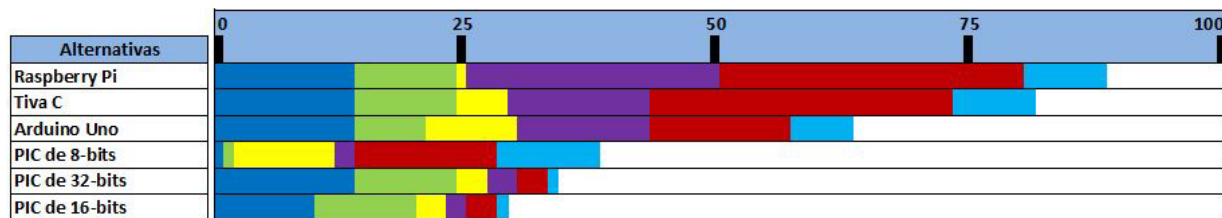
En este trabajo se presenta la implementación de este algoritmo en un sistema físico, la validación del sistema físico y la correcta migración de este. Se busca replicar de mejor forma los resultados obtenidos a nivel de simulación (webots) mientras el PSO es ejecutado en un sistema físico. Además, se realiza otra validación usando una mesa de pruebas en conjunto con un algoritmo de visión por computadora para lograr un paso más hacia una futura aplicación con robots móviles con ruedas.

## **Materiales y métodos**

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. Al tener distintas opciones de microcontroladores, plataformas de evaluación y ordenadores se realiza un *trade study*, las plataformas evaluadas fueron: Raspberry Pi, Tiva C, Arduino Uno, PIC-8bits, PIC-16-bits, PIC-32bits.

Entre los criterios usados están: capacidad de memoria, frecuencia de operación, adaptabilidad a robots móviles, disponibilidad, costo y previo uso. Se le asignó una calificación de 1 a 6 (siendo 6 la mejor calificación) a todas las posibles soluciones, las calificaciones se basaron en la literatura (Valdés y Areny, 2007; M. T. Inc., 2013; T. Instruments, 2013). En la figura 1 se observan los resultados y la mejor solución, siendo esta el ordenador Raspberry Pi el cual será llamado RPi en este artículo.



**Figura 1.** Trade Study sistema físico.

Criterio	Peso
Capacidad de memoria	15.000
Frecuencia de operación	10.000
Costo (Q)	10.000
Adaptabilidad a Robots móviles	25.000
Disponibilidad	30.000
Previo Uso	10.000

**Figura 2.** Criterios del Trade Study sistema físico.

Se evaluaron distintos lenguajes de programación de alto nivel para realizar la implementación del algoritmo MPSO. Las soluciones calificadas son los lenguajes: C, C++, Python, Java y Matlab. Los criterios usados fueron: disponibilidad de librerías, adaptabilidad y previo uso. Se le asignó una calificación de 1 a 5 (siendo 5 la mejor calificación) a todas las posibles soluciones. En la figura 2 se observan los resultados y la mejor solución, siendo el lenguaje C.



**Figura 3.** Trade Study lenguaje de programación.

Criterio	Peso
Previo Uso	30.000
Disponibilidad de librerías	40.000
Adaptabilidad	30.000

**Figura 4.** Criterios del Trade Study lenguaje de programación.

Como primera validación se implementó el algoritmo ordinario PSO en el ordenador RPi para comparar su funcionamiento con el desarrollado en Matlab para trabajos anteriores. Enfocando la implementación a robots móviles se implementó el algoritmo MPSO en cada dispositivo a utilizar en donde cada agente será capaz de ejecutar su propio algoritmo. 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. Para todas las demás pruebas se usaron los modelos RPi 3B, 3B+ y 4, por lo que estos resultados son replicables en cualquiera de estos modelos.

### **Sistema físico**

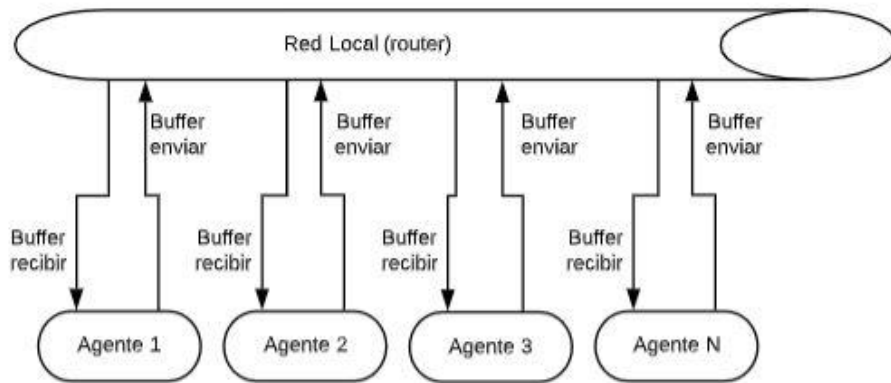
Tomando en consideración que cada agente representa un robot móvil se toma cada ordenador RPi como uno de estos. La combinación de todas las RPi, su comunicación entre ellas y su comunicación con la mesa de pruebas forman el conjunto del sistema físico donde se desea implementar y validar el algoritmo PSO.

### **Implementación física del PSO**

Con la limitante de no contar con una plataforma física operativa (robot móvil con ruedas) se buscaron alternativas para lograr una comunicación entre todos los agentes (parte fundamental del algoritmo) y que cada agente sea capaz de obtener su orientación y coordenadas actuales como entradas para el algoritmo. Para esto se usaron protocolos de comunicación a través de internet en combinación con programación multihilos para que estas tareas pueden ser ejecutadas de forma simultánea y no afecten la ejecución del algoritmo principal.

Como parte de otro proyecto de tesis se continuó con el desarrollo de un algoritmo de visión por computadora para el reconocimiento de la pose de agentes colocados en una mesa de pruebas. Se usó una red local y un protocolo de comunicación UDP para lograr una vía de comunicación entre este algoritmo desarrollado en Matlab y el algoritmo PSO desarrollado en lenguaje C. Se estableció que los agentes fueran los servidores mientras que el algoritmo de visión por computadora la de cliente, este último encargado de enviar a direcciones específicas las poses leídas.

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 usó 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, en la figura 5 se ejemplifica este proceso donde una cantidad N de agentes pueden intercambiar información entre sí.



**Figura 5.** Representación del envío y recepción de datos entre agentes.

### Ambientes Controlados

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 trabajos previos y la segunda fue una demostración usando la mesa de pruebas de la UVG la cual se puede ver en la figura 6.



**Figura 6.** Mesa de pruebas de la UVG.

### *Validación implementando Webots*

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 válida 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 almacena 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 válida con los resultados obtenidos en fases anteriores.

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 dadas 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.

#### *Validación implementando algoritmo visión por computadora*

Para la segunda validación se usó el algoritmo de visión por computadora desarrollado como otro trabajo de graduación para obtener las coordenadas y orientación de cada agente. Por las dimensiones de la mesa de pruebas y el tamaño de los marcadores se usaron únicamente cuatro agentes, el algoritmo de visión por computadora toma tiempo en procesar cada imagen por lo que cada iteración del PSO se hace al recibir una nueva coordenada.

## **Resultados y discusión**

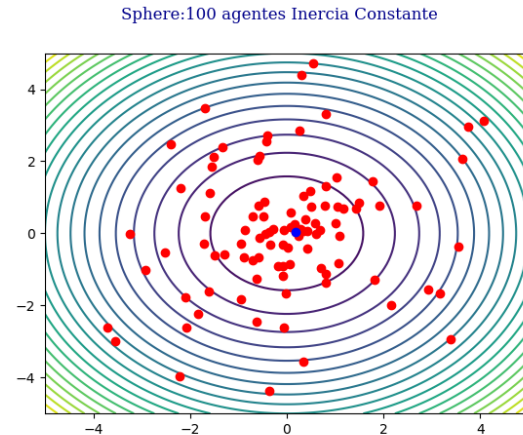
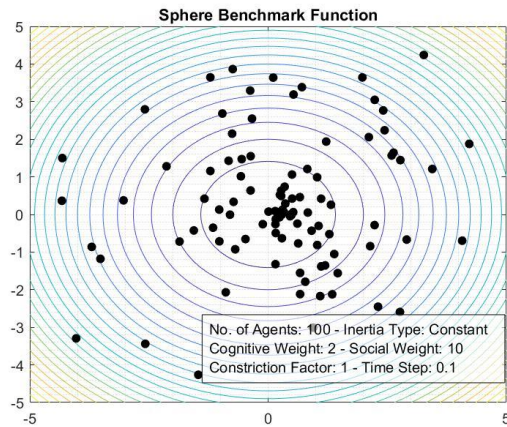
Para tener una comparación valida del algoritmo ordinario PSO (el ejecutado en RPi y matlab) ambos programas fueron ajustados con las mismas métricas, se probaron las funciones costo *Sphere* y *Himmelblau*, en combinación de un tipo de inercia constante y caótica. Se generaron 3 casos para cada combinación donde se variaron los agentes. En la tabla 1 se puede ver la mejor posición alcanzada en X y Y para el caso de 10, 50 y 100 agentes. Al no contar con dimensiones físicas, los agentes se modelan como partículas sin dimensiones.

**Tabla 1.** Resultados de la función costo *Sphere* y una inercia constante.

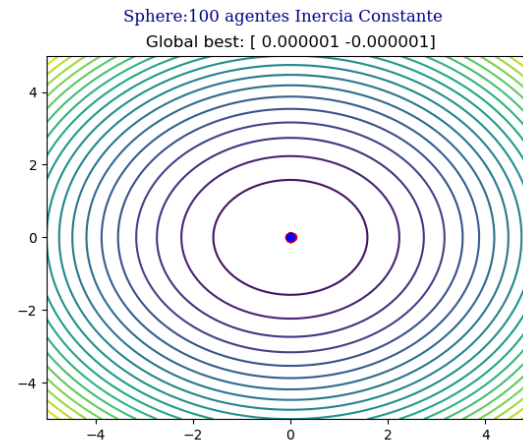
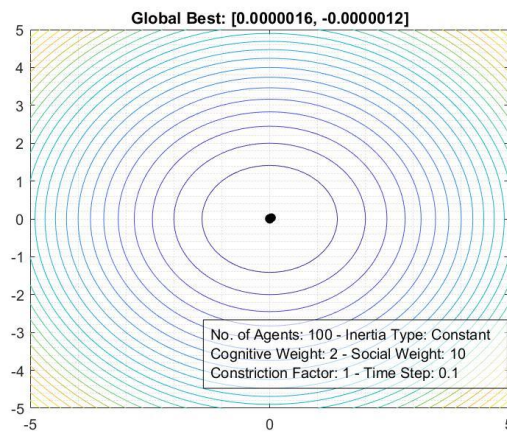
Cantidad de Agentes	Matlab		Raspberry Pi	
	X	Y	X	Y
10	0.000005	-0.000011	0.000005	-0.000010
50	-0.0000002	0.000000	-0.0000003	0.000000
100	0.000001	-0.000001	0.000001	-0.000001

En las figuras 7 y 8 se pueden ver las posiciones iniciales y finales de todos los agentes para el caso de la función costo *Sphere* y una inercia constante 100 agentes.





**Figura 7.** PSO con 100 agentes e inercia constante, posiciones iniciales.



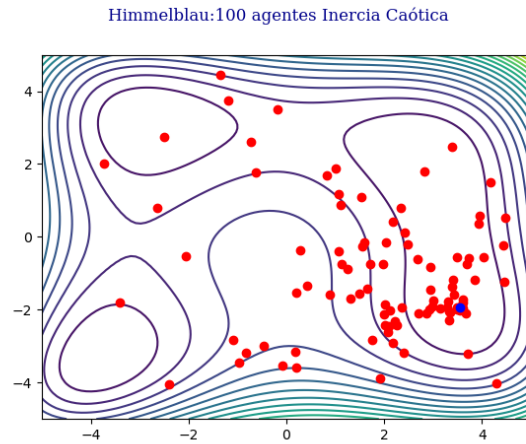
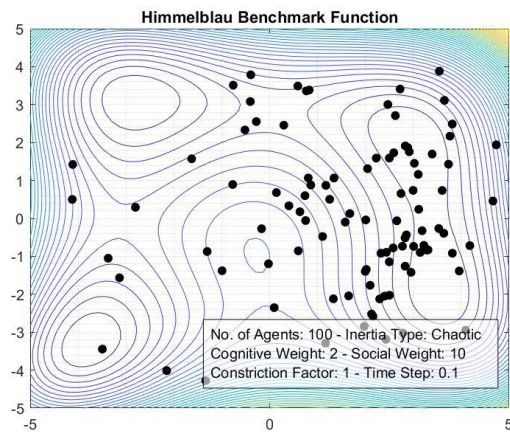
**Figura 8.** PSO con 100 agentes e inercia constante, posiciones finales.

En la tabla 2 se muestran los resultados de la función costo *Himmelblau* y una inercia caótica. Con esta combinación de parámetros se obtuvo un mejor resultado mostrando hasta 7 cifras significativas iguales.

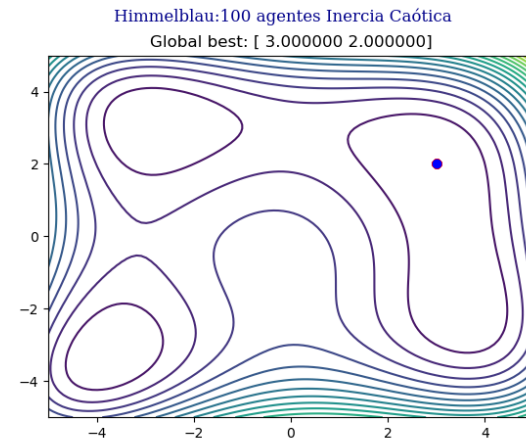
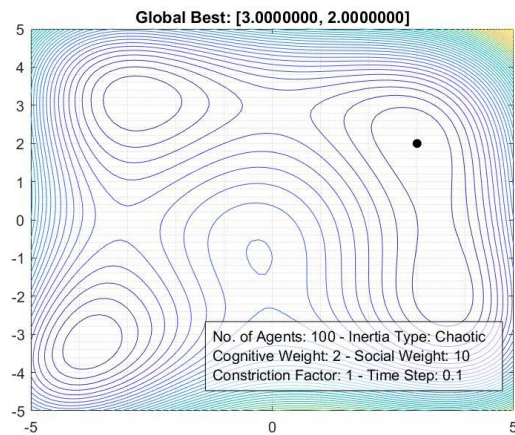
**Tabla 2.** Resultados de la función costo *Himmelblau* y una inercia caótica.

Cantidad de Agentes	Matlab		Raspberry Pi	
	X	Y	X	Y
10	3.5844283	-1.8481265	3.5844283	-1.8481265
50	3.5844283	-1.8481265	3.5844283	-1.8481265
100	3.0000000	2.0000000	3.0000000	2.0000000

En las figuras 9 y 10 se pueden ver las posiciones iniciales y finales de todos los agentes para el caso de la función costo *Himmelblau* y una inercia caótica 100 agentes.



**Figura 9.** PSO con 100 agentes e inercia caótica, posiciones iniciales.



**Figura 10.** PSO con 100 agentes e inercia caótica, posiciones finales.

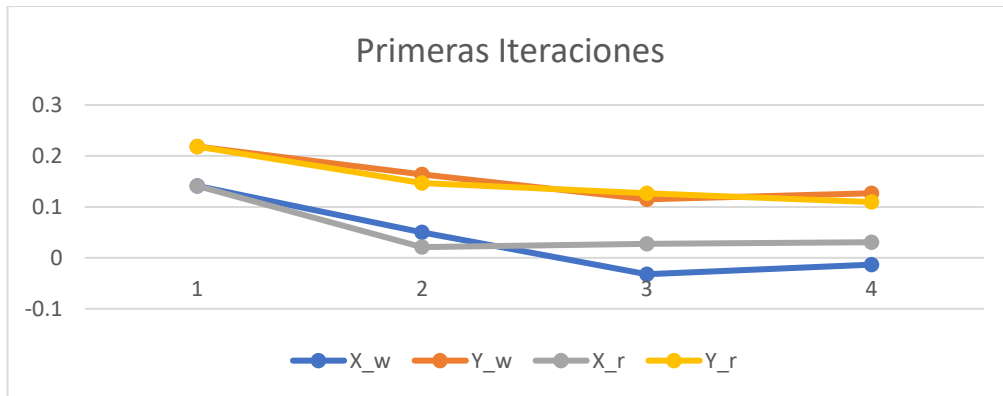
### Validación de la implementación de algoritmo en sistemas físicos

Para la primera validación se realizaron un total de cinco pruebas combinando funciones costo, controladores y tipo de inercia. En la tabla 3 se muestra las primeras iteraciones del algoritmo para una de las pruebas, en la cual se evaluó la función costo *Rosenbrock*, un tipo de inercia aleatoria y un controlador de pose. Estos resultados corresponden a uno de los siete agentes usados para esta prueba.

**Tabla 3.** Prueba 5 primeras iteraciones agente 7.

	Webots		Raspberry		% Error	
Nuevas posiciones (mm)						
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%

El error obtenido representa la resta del valor teórico (webots) menos el valor experimental (Raspberry) con relación a las dimensiones físicas del robot móvil, que para estas pruebas se toman las dimensiones del robot móvil E-puck, estas en milímetros. Como se ve en la figura 11 el agente comienza en el mismo punto en ambas pruebas y si bien toma las nuevas posiciones varían ligeramente presentan el mismo comportamiento.



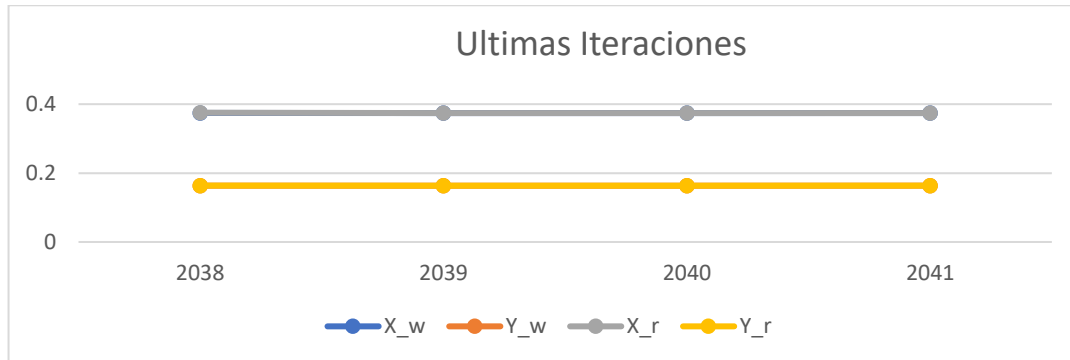
**Figura 11.** Gráfico de primeras iteraciones.

En la tabla 4 se muestran las iteraciones finales del algoritmo para la misma prueba y agente, esta prueba tomo 2041 iteraciones para detenerse.

**Tabla 4.** Prueba 5 ultimas iteraciones agentes 7.

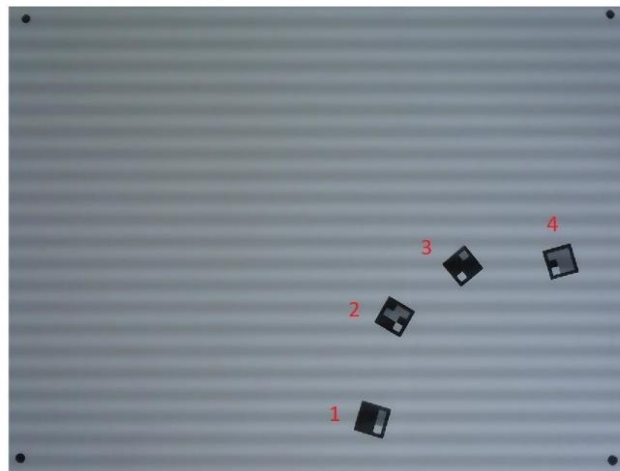
	Webots		Raspberry		% Error	
Nuevas posiciones (mm)						
iteración	X_w	Y_w	X_r	Y_r	X	Y
2038	0.37453	0.163496	0.375238	0.163597	0.001%	0.000%
2039	0.374411	0.163480	0.374498	0.163492	0.000%	0.000%
2040	0.37428	0.163462	0.374293	0.163463	0.000%	0.000%
2041	0.374204	0.16345	0.374241	0.163452	0.000%	0.000%

Como se ve en la figura 12 el agente calcula las mismas nuevas posiciones en ambas implementaciones, esto en las iteraciones finales, lo que muestra una buena convergencia del agente 7 para esta prueba.



**Figura 12.** Gráfico de iteraciones finales.

Para la segunda validación se evaluó la función costo *Shpere*, un tipo de inercia exponencial y un controlador PID. En la figura 13 se muestran las posiciones iniciales de los 4 marcadores, las dimensiones de los marcadores son de 5 cm cuadrados mientras que la mesa mide 1x0.8 m



**Figura 13.** Posición inicial de los marcadores.

En la figura 14 se ven los resultados del algoritmo PSO ejecutado en 4 RPi, la nueva posición calculada para X para los agentes 1, 2 y 3 nos indica que se deben mover a la derecha mientras que la coordenada Y nos indica que se deben de mover hacia abajo. Las nuevas posiciones del agente 4 presentan el mismo movimiento, pero en menor cantidad ya que este se encuentra más pegado al límite de la mesa de pruebas.

```
pi@IEUVG: ~/Documents
Coordenadas agente 1 son: 154.280300, 46.071400, 154.798828.
Comienza PSO No. de iteracion 26
Nuevas Posiciones del agente 1 son: -3370.560445, 2860.417746.
Velocidad del agente 1 son: 2.000000, 2.000000.

Coordenadas agente 1 son: 150.389700, 45.035500, 155.555908.
Comienza PSO No. de iteracion 27
Nuevas Posiciones del agente 1 son: -3405.629063, 2891.564373.
Velocidad del agente 1 son: 2.000000, 2.000000.

Coordenadas agente 1 son: 145.861600, 46.366500, 155.224854.
Comienza PSO No. de iteracion 28
Nuevas Posiciones del agente 1 son: -2948.350108, 2535.355093.
Velocidad del agente 1 son: 2.000000, 2.000000.

pi@IEUVG: ~/Desktop
Coordenadas agente 2 son: 126.717000, 107.101400, 149.743530.
Comienza PSO No. de iteracion 26
Nuevas Posiciones del agente 2 son: -2643.316086, 1424.996470.
Velocidad del agente 2 son: 2.000000, 2.000000.

Coordenadas agente 2 son: 124.107000, 104.333700, 150.255005.
Comienza PSO No. de iteracion 27
Nuevas Posiciones del agente 2 son: -2677.266179, 1398.983767.
Velocidad del agente 2 son: 2.000000, 2.000000.

Coordenadas agente 2 son: 119.948600, 101.486500, 150.255005.
Comienza PSO No. de iteracion 28
Nuevas Posiciones del agente 2 son: -2317.811938, 1233.056915.
Velocidad del agente 2 son: 2.000000, 2.000000.

pi@IEUVG: ~/Desktop
Coordenadas agente 3 son: 89.262700, 165.261200, 123.023865.
Comienza PSO No. de iteracion 25
Nuevas Posiciones del agente 3 son: -2149.188884, -11.877909.
Velocidad del agente 3 son: 2.000000, 2.000000.

Coordenadas agente 3 son: 88.441800, 161.620200, 122.471191.
Comienza PSO No. de iteracion 26
Nuevas Posiciones del agente 3 son: -1610.568848, 17.892825.
Velocidad del agente 3 son: 2.000000, 2.000000.

Coordenadas agente 3 son: 84.649000, 160.112100, 120.963745.
Comienza PSO No. de iteracion 27
Nuevas Posiciones del agente 3 son: -1629.151058, -44.410149.
Velocidad del agente 3 son: 2.000000, 2.000000.

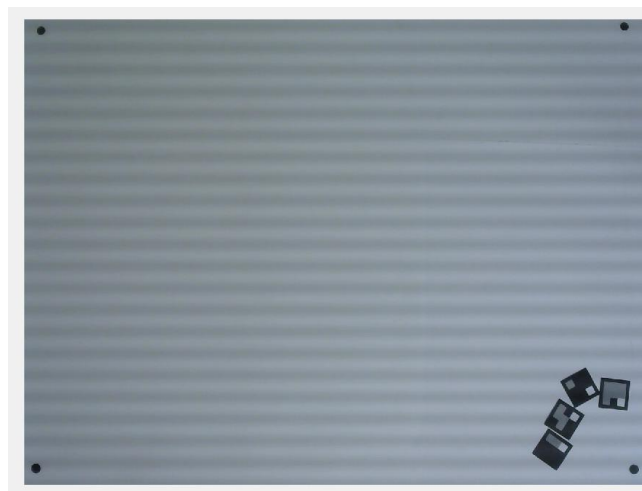
pi@IEUVG: ~/Desktop
Coordenadas agente 4 son: 25.529500, 151.611500, 80.537659.
Comienza PSO No. de iteracion 22
Nuevas Posiciones del agente 4 son: 25.495546, 151.587931.
Velocidad del agente 4 son: -2.000000, -2.000000.

Coordenadas agente 4 son: 23.534000, 147.261400, 83.659790.
Comienza PSO No. de iteracion 23
Nuevas Posiciones del agente 4 son: 23.509219, 147.244198.
Velocidad del agente 4 son: -2.000000, -2.000000.

Coordenadas agente 4 son: 23.534000, 147.261400, 83.659790.
Comienza PSO No. de iteracion 24
Nuevas Posiciones del agente 4 son: 23.515914, 147.248845.
Velocidad del agente 4 son: -2.000000, -2.000000.
```

**Figura 14.** Proceso iterativo.

En la figura 8 se muestran las posiciones finales de los agentes aproximadamente en la interacción 35, siendo esta posición la última que el algoritmo de visión por computadora logra capturar, luego de esto deja de reconocer los marcadores.



**Figura 8.** Posición final de los marcadores.

## Conclusiones

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 gráficas 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 cálculo de nuevas posiciones que hace sentido de acuerdo con la posición actual de los marcadores.

Al evaluar distintas plataformas y microcontroladores y de acuerdo con la validación del PSO ordinario y su comparación con su versión en Matlab el ordenador Raspberry Pi es la mejor opción para la implementación del PSO en un sistema físico.

## Bibliografía

S. A. Nadalini, “Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO)”, Tesis de Licenciatura, Universidad del Valle de Guatemala, 2020.

C. Duarte y C. J. Quiroga, *PSO algorithm*. Ciudad Universitaria, Santander, Colombia: Santander, 2010.

F. Valdes y R. Areny, *Microcontroladores Fundamentos y Aplicaciones con PIC*. España: Marcombo, 2007

L. S. Tortosa, “Agentes y enjambres artificiales: modelado y comportamientos parasistemas de enjambre robóticos,” phdthesis, Universidad de Alicante, España, 2013

R. F. R. Grandi y C. Melchiorri, *A Navigation Strategy for Multi-Robot SystemsBased on Particle Swarm Optimization Techniques*. Dubrovnik, Croatia: Dubrovnik, 2012.

M. T. Inc, General Purpose, 16-Bit Flash Microcontrollers with XLP Technology Data Sheet, Microchip Technology Inc, september 2013. 68

M. T. Inc, PIC16F87XA Data Sheet, Microchip Technology Inc, september 2013.

M. T. Inc, PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Family, Micro-chip Technology Inc, september 2013.

T. Instruments, Tiva C Series TM4C123G LaunchPad Evaluation Board, Texas Instruments, 2013.