
Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO)

Aldo Stefano Aguilar Nadalini



UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



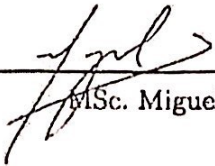
**Algoritmo Modificado de Optimización de
Enjambre de Partículas (MPSO)**

Trabajo de graduación presentado por Aldo Stefano Aguilar Nadalini
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

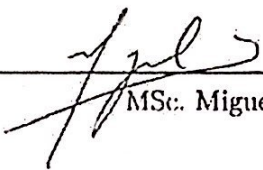
Guatemala,

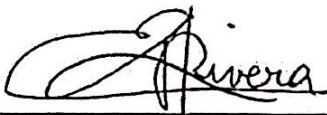
2019

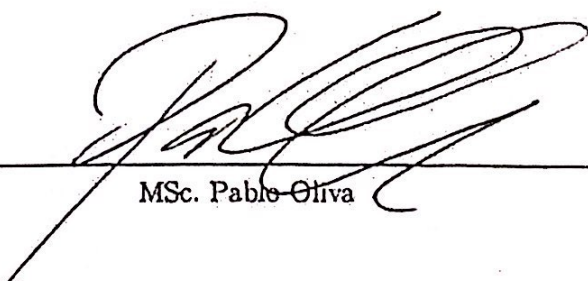
Vo.Bo.:

(f) 
MSc. Miguel Zea

Tribunal Examinador:

(f) 
MSc. Miguel Zea

(f) 
PhD. Luis Rivera

(f) 
MSc. Pablo Oliva

Fecha de aprobación: Guatemala, 03 de diciembre de 2019.

La elaboración de este trabajo escrito requirió de la utilización de conocimientos que abarcan diferentes áreas de investigación incluyendo robótica, sistemas de control y programación de algoritmos. Por esto mismo, no hubiera sido posible completar este trabajo sin todos los conocimientos que me fueron inculcados por profesores excelentemente preparados a lo largo de mi trayectoria en la Universidad del Valle de Guatemala. Este trabajo presenta al lector con la oportunidad de explorar una aplicación diferente de la optimización por inteligencia de enjambre implementada en robots diferenciales E-Puck. Este trabajo presenta un elevado contenido de fórmulas matemáticas, por lo que se asume que el estimado lector posee un conocimiento básico de aritmética, álgebra y cálculo diferencial.

Agradezco a mis padres por su constante apoyo durante estos 5 años de carrera y agradezco a Juan Pablo Cahueque, Fredy España y Antonio Ixtecoc por todas las horas de trabajo que realizamos como equipo durante toda la carrera. Sin lugar a dudas, mi trayectoria en la universidad no hubiera sido la misma sin su apoyo. Por último, agradezco a mi asesor de tesis Miguel Zea por la ayuda brindada para garantizar que este trabajo fuera de calidad, así como al Dr. Luis Rivera por prestar su tiempo y proveer retroalimentación útil para este trabajo.

Prefacio	III
Lista de figuras	IX
Lista de cuadros	X
Resumen	XI
Abstract	XII
1. Introducción	1
2. Antecedentes	3
2.1. Megaproyecto Robotat - Fase I	3
2.2. Particle Swarm Optimization (PSO)	3
2.3. Robotarium de Georgia Tech	6
2.4. Robot E-Puck	7
3. Justificación	8
4. Objetivos	9
4.1. Objetivo general	9
4.2. Objetivos específicos	9
5. Alcance	10
6. Marco teórico	11
6.1. Particle Swarm Optimization (PSO)	11
6.1.1. Generalidades del algoritmo	11
6.1.2. Estructura del algoritmo	11
6.1.3. Parámetros importantes del algoritmo	14
6.2. Funciones de costo	15
6.2.1. Función Schaffer F6	15
6.2.2. Función Sphere	16

6.2.3. Función Rosenbrock	17
6.2.4. Función Booth	17
6.2.5. Función Himmelblau	18
6.3. Robots diferenciales	19
6.3.1. Modelado de robots diferenciales	19
6.3.2. Difeomorfismo para transformación de cinemática de unicycle	21
6.4. Controladores de posición y velocidad de robots diferenciales	23
6.4.1. Control proporcional de velocidades con saturación limitada	23
6.4.2. Control PID de velocidad angular ω	23
6.4.3. Control proporcional de velocidad lineal v	23
6.4.4. Controlador de pose	24
6.4.5. Controlador de pose con criterio de estabilidad de Lyapunov	26
6.4.6. Controlador de lazo cerrado de direccionamiento de robot	28
6.4.7. Control por medio de regulador lineal cuadrático (LQR)	31
6.4.8. Controlador lineal cuadrático integral (LQI)	33
6.5. Cálculo de suavidad de curvas por método de energía de deflexión de vigas	34
6.5.1. Minimización de energía de curvas interpoladas por trazadores cúbicos	34
6.6. Simuladores por medio de software	35
6.6.1. MATLAB R2017b	35
6.6.2. WeBots R2019a	35
7. Estableciendo el valor correcto de parámetros para MPSO	36
7.1. Simulación de partículas para ajuste de parámetros PSO	36
7.1.1. Características de simulación dependiendo el parámetro de inercia	36
7.1.2. Características de simulación dependiendo los parámetros de escalamien-	40
to	40
7.1.3. Características de simulación dependiendo el parámetro de constricción	43
7.2. Selección final del parámetros PSO	47
8. Estructuración del algoritmo MPSO en simulación de Webots	48
8.1. Programación de algoritmo MPSO para simular convergencia en WeBots	48
8.1.1. Elementos del mundo simulado en WeBots	48
8.1.2. Diseño del algoritmo de controlador MPSO	49
8.1.3. Variables de controlador en WeBots	53
8.1.4. Cálculo de orientación de robot	53
8.1.5. Cálculo de posición de robot	55
8.1.6. Cálculo de errores de posición y orientación de robots	55
9. Planeación de trayectorias de robots por medio de PSO	56
9.1. Parámetros del planeador de trayectorias PSO	58
9.2. Ecuación modificada de actualización de velocidad PSO	59
9.3. Ciclo de ejecución de planeador PSO	59
10. Implementación de controladores a robots de la simulación	60
10.1. Control proporcional de cinemática transformada (TUC)	61
10.1.1. Diseño de controlador de cinemática transformada	61
10.2. Controlador PID de velocidad angular (TUC-PID)	64
10.3. Controlador simple de pose de robot diferencial (SPC)	67

10.4. Controlador Lyapunov de pose de robot diferencial (LSPC)	70
10.5. Controlador de direccionamiento en lazo cerrado (CLSC)	73
10.6. Controlador regulador lineal cuadrático y unicycle transformado (TUC-LQR)	76
10.7. Controlador lineal cuadrático integral y unicycle transformado (TUC-LQI)	80
10.8. Comparación de desempeño de controladores	84
11. Conclusiones	87
12. Recomendaciones	89
13. Bibliografía	90
14. Anexos	92
14.1. Código y simulación utilizado en la investigación	92
14.2. Representación de agentes en simulación de Webots	93
14.3. Pseudocódigo de implementación digital de control PID	94
14.4. Capturas de pantalla de simulaciones en MATLAB	95

Lista de figuras

1. Controlador dinámico-cinemático de dos lazos [5]	5
2. Representación gráfica de topologías [6]	5
3. Detección de obstáculos con partículas [6]	5
4. Robotarium de Georgia Insitute of Technology [7]	6
5. Robot diferencial E-Puck [8]	7
6. Representación vectorial de ecuación de velocidad PSO [13]	13
7. Representación gráfica de función Schaffer F6 [15]	16
8. Representación gráfica de función Sphere [15]	16
9. Representación gráfica de función Rosenbrock [15]	17
10. Representación gráfica de función Booth [15]	17
11. Representación gráfica de función Himmelblau [15]	18
12. Modelo de robot diferencial [19]	20
13. Punto de transformación de cinemática [21]	22
14. Cinemática de robot y referencias de interés [17]	24
15. Trayectorias resultantes de convergencia con controlador de pose [22]	26
16. Estabilidad de Lyapunov vs. estabilidad asintótica [24]	27
17. Trayectorias de robot con controlador de direccionamiento ($k_1 = 1, k_2 = 3$) [26]	30
18. Trayectorias de robot con controlador de direccionamiento ($k_1 = 1, k_2 = 10$) [26]	30
19. Función de transferencia de sistema en representación LTI de espacio de estados	31
20. Estructura de controlador lineal cuadrático integral	33
21. Pantalla de inicio de MATLAB	35
22. Pantalla de inicio de WeBots	35
23. Dispersión de partículas utilizando parámetro de inercia caótica	37
24. Dispersión de partículas utilizando parámetro de inercia random	37
25. Dispersión de partículas utilizando parámetro de inercia constante	38
26. Dispersión de partículas utilizando parámetro de inercia lineal decreciente	38
27. Dispersión de partículas utilizando parámetro de inercia natural exponencial	39
28. Contornos de nivel de función de costo Himmelblau con sus cuatro mínimos	40
29. Dispersión de partículas utilizando $c_1 = 2$ y $c_2 = 2$	41
30. Dispersión de partículas utilizando $c_1 = 1$ y $c_2 = 5$	41

31. Dispersión de partículas utilizando $c_1 = 5$ y $c_2 = 1$	42
32. Dispersión de partículas utilizando $c_1 = 2$ y $c_2 = 10$	42
33. Dispersión de partículas utilizando ecuación (3) de φ	44
34. Dispersión de partículas utilizando $\varphi = 0.5$	44
35. Dispersión de partículas utilizando $\varphi = 1.0$	45
36. Dispersión de partículas utilizando $\varphi = 1.1$	45
37. Dispersión de partículas utilizando $\varphi = 1.3$	46
38. Movimiento resultante de partícula con parámetros elegidos (Función de prueba Sphere)	47
39. Mundo simulado en WeBots con E-Pucks implementados	49
40. Diagrama de flujo de configuración de controlador MPSO	50
41. Diagrama de flujo de controlador MPSO (Pt.1)	51
42. Diagrama de flujo de controlador MPSO (Pt.2)	52
43. Ejes de simulación vs. ejes de nodo <i>Compass</i>	53
44. Ejes de E-Puck y nodo <i>Compass</i>	54
45. Marcadores PSO para seguimiento de trayectoria	57
46. Ciclo de ejecución de MPSO	59
47. Trayectoria generada por marcadores PSO y controlador TUC	62
48. Velocidad lineal y angular de robot con controlador de cinemática transformada	63
49. Trayectorias resultantes de enjambre con controlador de cinemática transformada	64
50. Trayectoria generada por marcadores PSO y controlador TUC-PID	65
51. Velocidad lineal y angular de robot con controlador de cinemática y PID angular	66
52. Trayectorias resultantes de enjambre con controlador de cinemática y PID angular	67
53. Trayectoria generada por marcadores PSO y controlador SPC	68
54. Velocidad lineal y angular de robot con controlador simple de pose	69
55. Trayectorias resultantes de enjambre con controlador simple de pose	70
56. Trayectoria generada por marcadores PSO y controlador LSPC	71
57. Velocidad lineal y angular de robot con controlador de pose Lyapunov	72
58. Trayectorias resultantes de enjambre con controlador Lyapunov de pose	73
59. Trayectoria generada por marcadores PSO y controlador CLSC	74
60. Velocidad lineal y angular de robot con controlador de direccionamiento	75
61. Trayectorias resultantes de enjambre con controlador de direccionamiento	76
62. Trayectoria generada por marcadores PSO y controlador TUC-LQR	77
63. Velocidad lineal y angular de robot con controlador TUC-LQR	78
64. Trayectorias resultantes de enjambre con controlador TUC-LQR	79
65. Trayectoria generada por marcadores PSO y controlador TUC-LQI	81
66. Velocidad lineal y angular de robot con controlador TUC-LQI	82
67. Trayectorias resultantes de enjambre con controlador TUC-LQI	83
68. Suavidad de curvas de cada controlador calculada con energía de deformación	84
69. Porcentaje de saturación de actuadores durante simulaciones	85
70. Dispersión (X,Y) de robots E-Puck en espacio de búsqueda	86
71. Simulación de robot E-Puck dentro de mundo Webots	93
72. Simulación con función de costo Rosenbrock	95

73. Simulación con función de costo Sphere	95
74. Simulación con función de costo Booth	96
75. Simulación con función de costo Himmelblau	96
76. Simulación con función de costo Mínimos Múltiples	97
77. Simulación con función de costo Schaffer F6	97

1. Variación de parámetro de inercia [4]	4
2. Resultados de variación de parámetro de inercia [4]	4
3. Tabla de resultados: Parámetro de inercia	39
4. Mínimos de función Himmelblau	40
5. Tabla de resultados: Parámetros de escalamiento	43
6. Tabla de resultados: Parámetro de constricción	46
7. Parámetros de controlador TUC	62
8. Parámetros de controlador TUC-PID (Velocidad lineal)	65
9. Parámetros de controlador TUC-PID (Velocidad angular)	65
10. Parámetros de controlador simple de Pose (SPC)	68
11. Parámetros de controlador Lyapunov de Pose (LSPC)	71
12. Parámetros de controlador de direccionamiento (CLSC)	74
13. Parámetros de controlador con regulador lineal cuadrático (TUC-LQR)	77
14. Parámetros de controlador con integrador lineal cuadrático (TUC-LQI) No.1	80
15. Parámetros de controlador con integrador lineal cuadrático (TUC-LQI) No.2	81
16. Resultados de controladores	85

Un sistema multiagente de robots diferenciales requiere de un algoritmo definido para comportarse como un enjambre con habilidad de búsqueda de metas. En esta investigación, se propuso el uso del algoritmo Particle Swarm Optimization (PSO) para lograr que los agentes encontraran el camino óptimo hacia una meta dentro de una función de costo de manera colectiva. El algoritmo PSO clásico está diseñado para partículas sin masa ni dimensiones físicas; contrario a los robots diferenciales. Por lo tanto, el objetivo principal de esta investigación fue diseñar correctamente el algoritmo Modified Particle Swarm Optimization (MPSO) que tomara en cuenta las restricciones derivadas de las ecuaciones cinemáticas de los robots diferenciales y las dimensiones limitadas del espacio de búsqueda.

El algoritmo MPSO incluye un planeador de trayectorias utilizando el PSO clásico ajustado con múltiples parámetros optimizados para robots diferenciales, incluyendo el parámetro de inercia, el parámetro de constricción, dos parámetros de escalamiento y dos parámetros de uniformidad (para los factores cognitivo y social del PSO). Estos parámetros fueron configurados por medio de simulaciones de enjambres de 200 partículas en MATLAB evaluadas en diversas funciones de costo. Este algoritmo también incluye los controladores necesarios para rastrear las trayectorias computadas por el planeador PSO y generar velocidades suaves y continuas de robots diferenciales. Se evaluaron siete diferentes controladores incluyendo: Transformación de unicycle (TUC), Transformación de unicycle con PID (TUC-PID), Controlador simple de pose (SPC), Controlador de pose Lyapunov-estable (LSPC), Controlador de direccionamiento de lazo cerrado (CLSC), Transformación de unicycle con LQR (TUC-LQR), y Transformación de unicycle con LQI (TUC-LQI). El desempeño de cada uno de los controladores fue evaluado por medio de simulaciones con el software de Webots utilizando un enjambre de 10 robots E-Puck. El controlador TUC-LQI obtuvo los mejores resultados en cuanto a generar trayectorias suaves y velocidades continuas correctamente reguladas para los robots diferenciales.

A multi-agent E-Puck robotic system requires a definite algorithm to behave as a swarm with goal searching capabilities. The use of the Particle Swarm Optimization (PSO) algorithm was proposed to enable the agents to collectively find the optimal path to a goal in a cost function. The classic PSO algorithm is designed for particles with no mass or physical dimensions unlike E-Puck differential robots. Therefore, the main objective of this research was the correct design of a Modified Particle Swarm Optimization (MPSO) algorithm to take into account the restrictions derived from the kinematic equations of the E-Puck agents and the finite dimensions of the search space.

The MPSO includes a PSO trajectory planner using multiple known PSO parameters such as inertia weight, a constriction parameter, two scaling factors and two uniformity factors (for the cognitive and social components of the PSO) optimized for the E-Puck robots. The selection of these parameters was carried out using MATLAB simulations with swarms comprised of 200 particles evaluated in multiple cost functions. It also includes the necessary controllers to track the trajectories calculated by the PSO trajectory planner using smooth and continuous differential robot velocities. Seven different controllers were tested including: Transformed Unicycle (TUC), Transformed Unicycle with PID (TUC-PID), Simple Pose Controller (SPC), Lyapunov-stable Pose Controller (LSPC), Closed-loop Steering Controller (CLSC), Transformed Unicycle with LQR (TUC-LQR), and Transformed Unicycle with LQI (TUC-LQI). The controllers' performances were evaluated via Webots simulations using swarms of 10 E-Puck robots. The TUC-LQI was the controller that performed better in terms of achieving smooth trajectories and generating smooth continuous differential robot velocities.

El algoritmo de Particle Swarm Optimization modificado (MPSO) surgió primeramente de dos ecuaciones vectoriales del PSO clásico para la búsqueda de mínimos y máximos dentro de un espacio determinado. Estas ecuaciones evalúan los puntos de posición de una partícula en una función de costo. La primera ecuación de velocidad se compone del factor de memoria o velocidad pasada, el factor cognitivo siendo la diferencia entre el costo de la posición actual y la mejor posición personal encontrada por la partícula, y el factor social siendo la diferencia entre el costo de la posición actual y la mejor posición global encontrada por el enjambre. La segunda ecuación de posición es la actualización de la posición actual a una posición nueva, tomando la nueva velocidad de partícula multiplicada por un delta de tiempo específico.

Las modificaciones realizadas al algoritmo clásico de PSO surgieron a causa de la necesidad de implementar dicho algoritmo de búsqueda a robots que no se comportaban como partículas, sino que se veían limitados por sus características físicas (dimensiones y limitaciones de movimiento). Tanto las ecuaciones de cinemática de los robots diferenciales, como las restricciones del espacio de búsqueda tenían efecto sobre los parámetros numéricos ideales que se buscaban para la operación de estos. Se buscó que estos se comportaran como un enjambre capaz de rastrear y alcanzar una meta específica dentro del espacio de búsqueda de una manera óptima. Para la validación del diseño de dicho algoritmo, se realizó la simulación del comportamiento de los Bitbots utilizando softwares de MATLAB y WeBots que permiten emular el espacio de trabajo.

Este trabajo escrito, primeramente, presenta la fase de selección de los parámetros óptimos de PSO por medio de MATLAB que lograron que un enjambre de partículas obtuviera un rango aceptable de exploración del espacio de búsqueda (mayor al 60% del espacio) y una tasa de convergencia rápida (menor a 30 segundos). Luego, se presenta la fase de estructuración del algoritmo MPSO en donde se presenta el diseño de las simulaciones en Webots utilizadas para validar el control de cinemática planteada de los robots diferenciales. Asimismo, se presenta el uso del PSO como planeador de trayectorias para robots diferenciales. Por último, se presentan los resultados de desempeño de siete diferentes controladores cine-

máticos que se estructuraron para seguir de manera suave las trayectorias generadas por el PSO.

2.1. Megaproyecto Robotat - Fase I

En la Fase 1 del Proyecto Robotat, se trabajó principalmente en el hardware que sería utilizado en los robots o agentes utilizados en el proyecto. Se construyó la plataforma del Robotat y se implementó un algoritmo de visión de computadora para poder obtener la posición y orientación de robots desde una perspectiva planar (dos dimensiones). Como se detalla en el trabajo escrito “Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre” del Ing. André Rodas [1], se utilizó una cámara sobre el Robotat para poder capturar los identificadores de los robots en el Robotat.

En esta misma fase, se trabajó en el módulo de conexión WiFi para poder establecer comunicación directa entre una computadora central y diferentes agentes Bitbot. El proceso de diseño de este protocolo de comunicación se detalla en el trabajo escrito “Diseñar e implementar una red de comunicación inalámbrica para la experimentación en robótica de enjambre” del Ing. Marlon Castillo [2]. Los resultados de estos trabajos permitieron observar hasta que punto de operabilidad se había llegado con los Bitbots y qué algoritmos hacían falta para que se pudieran operar múltiples agentes en conjunto dentro de la plataforma del Robotat por medio de técnicas de enjambre y de búsqueda de objetivos.

2.2. Particle Swarm Optimization (PSO)

En cuanto a la búsqueda de parámetros adecuados de PSO, investigaciones como [3] y [4], profundizan en los efectos de la variación de cada uno de dichos parámetros sobre el movimiento de las partículas. Particularmente en [4], se exponen los resultados de variar el tipo de parametro de inercia que se utilizaba en el algoritmo clásico de PSO evaluado en funciones de costo de prueba:

Cuadro 1: Iteraciones ejecutadas dependiendo parámetro de inercia y función de costo [4]

Tipo de parámetro de inercia	Sphere	Griewank	Rosenbrock	Rastrigin	Ackley
Constante	27612	3237	11512	3097	2854
Random	202	203	202	201	202
Adaptativa	306	282	320	297	292
Sigmoide creciente	469	265	419	319	352
Sigmoide decreciente	205	206	205	210	203
Lineal decreciente	2278	1460	1573	526	1254
Caótica	2404	1345	2469	1121	1012
Caótica random	202	202	202	201	202
Oscilante	13383	2207	26406	1535	1472
Global-local best	224	220	223	239	248
Reocido simulado	4245	821	2017	712	642
Natural exponencial (e1-PSO)	4390	1043	1231	928	790
Natural exponencial (e2-PSO)	1256	380	2248	531	332
Logarítmica decreciente	1573	822	1033	671	681
Exponencial decreciente	3627	870	1958	754	764

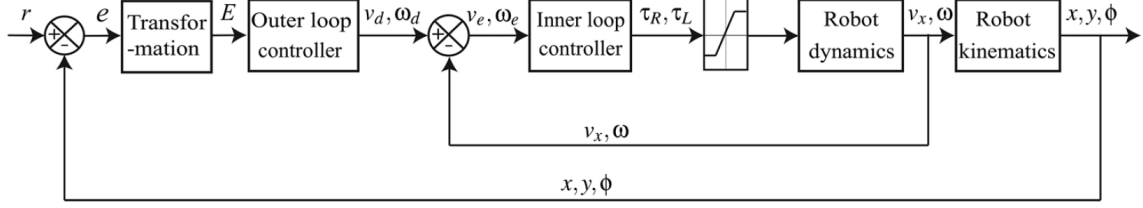
Los resultados de esta investigación arrojaron que utilizar un parámetro de inercia *random* causaba que la convergencia del algoritmo fuera la más rápida, mientras que utilizar un parámetro de inercia constante causaba la convergencia más lenta. También se evaluaron criterios como error promedio y mínimo error de convergencia como se observa en el siguiente cuadro:

Cuadro 2: Resultados de mejores y peores parámetros de inercia [4]

Criterio	Mejor estrategia de inercia	Peor estrategia de inercia
Error promedio	Inercia caótica	Inercia caótica <i>random</i>
Cantidad de iteraciones	Inercia <i>random</i>	Inercia constante
Error mínimo	Inercia lineal decreciente	Inercia caótica <i>random</i>

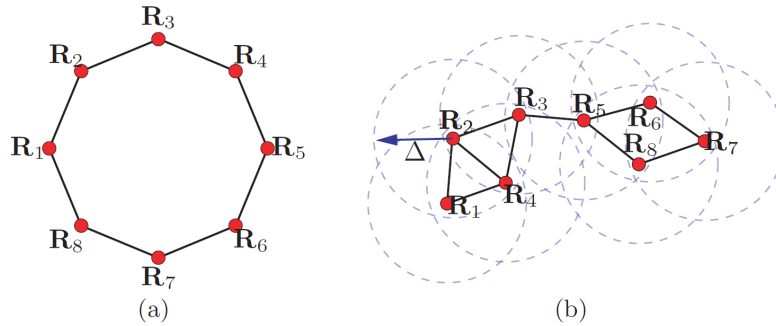
En publicaciones como [5], se determinan las ecuaciones adecuadas de cinemática de un modelo de robot diferencial. En dicha investigación, se utilizó el mismo modelo de robot que se utilizó en esta investigación. Sin embargo, en ella, se llevaron a cabo cálculos para establecer un controlador no solo cinemático, sino dinámico que tomaba en cuenta el torque de los motores y la fuerza de tracción de las llantas. Esto con el objetivo de verificar el efecto del deslizamiento de las llantas sobre el control de múltiples robots que siguen una trayectoria fija. Se utilizó el siguiente diseño de controlador:

Figura 1: Controlador dinámico-cinématico de dos lazos [5]



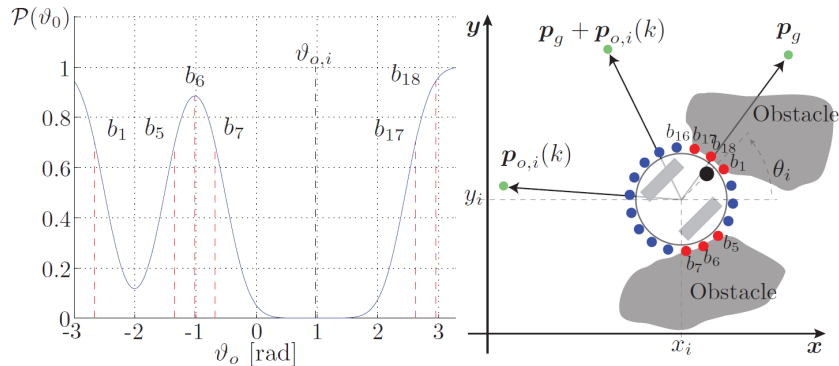
Por último, en investigaciones como [6], se profundiza más en las diferentes técnicas de navegación que se utilizan para aplicar algoritmos de PSO a robots con dimensiones físicas. El objetivo de esta investigación era determinar el efecto de la red o grafo de comunicación entre robots y la efectividad del algoritmo PSO clásico como tal. Se evaluaron dos topologías de transmisión de datos incluyendo un grafo de anillo (a) y un grafo de Δ -disco (b) como se observa en la siguiente figura:

Figura 2: Representación gráfica de topologías [6]



También se evaluó un algoritmo complementario para esquivar obstáculos y evitar colisiones. Para esto, colocaban las partículas PSO de tal manera que estuvieran alrededor de la circunferencia del robot y que estas evaluaran el entorno y detectaran obstáculos o localidades de costo muy alto en la función objetivo.

Figura 3: Detección de obstáculos con partículas [6]



2.3. Robotarium de Georgia Tech

El Robotarium es una plataforma que tiene como propósito ser utilizada para estudiar el comportamiento de robots en aplicaciones de seguimiento de trayectorias o comportamientos de enjambre. Esta plataforma fue originalmente diseñada por estudiantes del Instituto de Tecnología de Georgia en el 2016. El objetivo del Robotarium es ser un banco de pruebas accesible y fácil de ensamblar. La compra de un banco de pruebas para robots puede ser demasiado caro, por lo que un diseño más simple permite reducir costos. El diseño del Robotarium permite su fácil replicación a la hora de que estudiantes deban llevar a cabo proyectos de investigación [7].

El Robotarium se compone de una base plana con bordes que limitan el movimiento de los robots a utilizar en este y que son las fronteras del espacio de búsqueda o de tarea. El diseño original de la base es de color blanco para poder facilitar el procesamiento de imágenes del espacio de tarea. Se suele utilizar una cámara de alta resolución colocada directamente sobre el Robotarium para lograr capturar las posiciones de los diferentes robots de pruebas. Los robots utilizados en este banco de prueba poseen comunicación inalámbrica con el centro de cómputo del Robotarium para el envío de comandos o recepción de datos de la simulación de movimiento de dichos agentes. Los robots utilizados en estos bancos de prueba al estudiar aplicaciones de comportamientos de enjambre usualmente son diseños *open source* como el E-Puck presentado en la siguiente sección.

Figura 4: Robotarium de Georgia Institute of Technology [7]

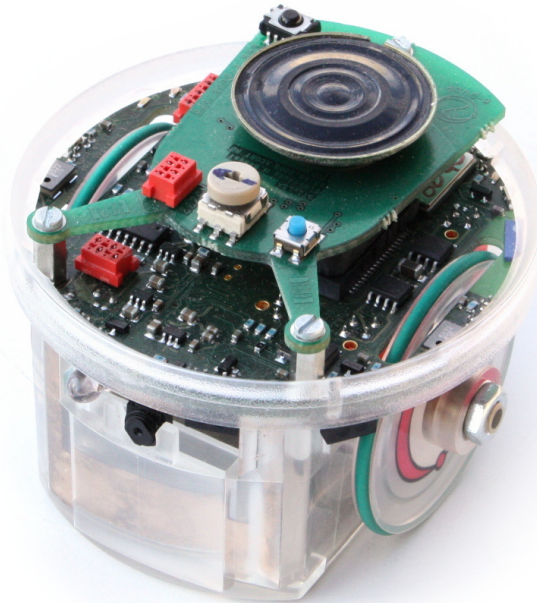


2.4. Robot E-Puck

Un ejemplo de robot diferencial es el E-Puck diseñado por Michael Bonani y Francesco Mondada en los Laboratorios ASL de la Universidad École Polytechnique Fédérale de Lausanne. El propósito de este robot es para educación en microingeniería. El diseño es *open hardware* y el software abordo también se puede obtener como *open source* [8]. Las características físicas de este robot son las siguientes.

- Diámetro de base: 70 mm
- Altura: 50 mm
- Radio de rueda: 20.5 mm
- Peso: 200 g
- Velocidad lineal máxima: 13 cm/s (angular 6.28 rad/s)

Figura 5: Robot diferencial E-Puck [8]



Los Bitbots diseñados en la Universidad del Valle de Guatemala como parte de la Fase I del Megaproyecto Robotat eran capaces de comportarse como un enjambre combinado de dimensiones reducidas. Estos podían ser localizados por medio de visión de computadora y eran capaces de seguir una trayectoria específica. Sin embargo, estos aún no eran capaces de llevar a cabo acciones como la determinación de mínimos en funciones de costo y la subsecuente búsqueda de estos puntos objetivo (metas) dentro del espacio de tarea utilizando las rutas óptimas. De aquí nació la necesidad de utilizar el algoritmo de Particle Swarm Optimization para darles a los robots diferenciales la capacidad de que, por medio del enjambre, pudieran llegar hacia metas definidas en rutas óptimas.

Las modificaciones del MPSO se debieron realizar para que fuera posible la implementación del algoritmo de búsqueda, ya que los robots poseían dimensiones y características limitadas en el mundo físico. Era imposible utilizar directamente el PSO clásico de control de partículas, ya que los robots diferenciales debían cumplir con sus propias relaciones de cinemática. Se debió lograr modelar dichos robots diferenciales de tal manera que pudieran ser controlados teniendo únicamente las velocidades dadas por el algoritmo de partículas.

Se debió validar el algoritmo MPSO por medio de simulaciones por software para determinar si, efectivamente, con las modificaciones elegidas, se podía controlar a los robots diferenciales para que convergieran hacia una meta definida. Estas simulaciones, debieron realizarse en softwares que permitían la simulación del comportamiento dinámico de los robots físicos para asemejarlos lo más posible a los agentes reales. Los resultados de esta investigación abren las puertas a una futura implementación del controlador MPSO programado para el control de un enjambre más grande de una mayor cantidad de robots que puedan buscar metas puntuales dentro del Robotat.

4.1. Objetivo general

Implementar un algoritmo de búsqueda basado en la modificación del método de Particle Swarm Optimization (PSO) estableciendo los parámetros indicados para un mejor comportamiento del enjambre y las ecuaciones para adaptar dicho algoritmo a las dimensiones físicas y cinemática de los Bitbots de UVG.

4.2. Objetivos específicos

- Encontrar el valor de los parámetros de la ecuación de PSO que permitan a los Bitbots comportarse como un enjambre que, en conjunto, busca una meta definida y converge en ella en un tiempo finito.
- Validar las ecuaciones y parámetros PSO establecidos por medio de simulaciones computarizadas que permitan la visualización del comportamiento del enjambre de partículas.
- Establecer las ecuaciones necesarias para modelar el movimiento y la cinemática de los Bitbots y hacer posible la adaptación del algoritmo PSO en ellos.
- Validar las simulaciones del MPSO en Bitbots por medio de la implementación del algoritmo en robots físicos simulados en el software de WeBots.

El alcance de esta investigación fue lograr plantear la forma adecuada de trasladar el algoritmo de Particle Swarm Optimization (PSO) de un espacio de movimiento de partículas a un espacio tridimensional, tomando en consideración las limitaciones de agentes roboticos en el mundo físico. Se planteó validar el desarrollo matemático de la adaptación PSO a la cinemática de robots diferenciales por medio de simulaciones significativas en MATLAB y WEBOTS. En MATLAB se realizó la evaluación de efectos de los parámetros PSO en el comportamiento del enjambre de partículas y se seleccionaron los valores que lograron dar una mayor exploración del espacio de tarea y a su vez una convergencia rápida a la meta encontrada.

En WeBots, se programó el controlador MPSO en lenguaje C en donde se le implementaron los parámetros PSO elegidos a un planeador de trayectorias PSO. Asimismo, se determinaron las ecuaciones de cinemática y leyes de control necesarias para el movimiento adecuado de los robots siguiendo las trayectorias PSO. Como resultado positivo se obtuvo la exitosa búsqueda de metas específicas dentro del Robotat simulado a las que los robots convergían como enjambre en un tiempo finito. Se tomó en cuenta la importancia de evitar la saturación y sobreactuación de los actuadores de los robots para que las leyes de control fueran aplicables en un futuro a robots reales. En futuras fases del proyecto, se le puede implementar el código en C del controlador a los robots físicos que se lleguen a utilizar en el Robotat, siempre y cuando se tomen las mismas medidas de los agentes utilizados en las simulaciones realizadas en esta investigación.

6.1. Particle Swarm Optimization (PSO)

6.1.1. Generalidades del algoritmo

El algoritmo clásico de optimización de enjambre de partículas (PSO por sus siglas en inglés) fue desarrollado por el Dr. Russell Eberhart y el Dr. James Kennedy en 1995 como un algoritmo de optimización basado en comportamiento de poblaciones [9]. Este algoritmo es similar a técnicas evolutivas de computación como los algoritmos genéticos (AG) explicados más a detalle en [10]. El algoritmo se inicializa con una población de soluciones aleatorias de una función de costo o fitness function; estas soluciones son denominadas partículas. Este algoritmo busca la solución óptima a la función de costo por medio de actualización de generaciones de partículas.

A diferencia de los algoritmos genéticos, el PSO no toma en cuenta operadores evolutivos como *crossover* o mutación de partículas al actualizar cada generación. Las partículas del PSO simplemente se mueven a través de espacio de trabajo siguiendo las partículas óptimas actuales de cada generación. El PSO tiene la ventaja de que es fácil de implementar a comparación de los algoritmos genéticos, ya que posee una menor cantidad de parámetros que se deben ajustar. Esto ha permitido que dicho algoritmo haya sido implementado exitosamente en campos de estudio como entrenamiento de redes neuronales, control de sistemas con lógica difusa, optimización de funciones computacionales e incluso en aplicaciones de optimización de construcción de estructuras como se expone en [11].

6.1.2. Estructura del algoritmo

El cálculo del algoritmo comienza con cada partícula teniendo una solución actual particular dada por un vector

$$\mathbf{x}_i = [x_1, x_2, \dots, x_d],$$

siendo d la dimensión del espacio de trabajo. El valor o costo de dicha solución está dado por:

$$f(\mathbf{x}),$$

siendo f la función de costo o *fitness function* evaluada por el algoritmo.

Luego, cada partícula posee en memoria la solución con menor costo que ha encontrado durante n iteraciones. A este dato se le denomina el **local best** de cada partícula. Asimismo, cada partícula transmite su solución óptima encontrada y se determina cual de todas las soluciones es la mejor del enjambre completo. A esta solución colectiva se le denomina **global best**. Con estos datos, cada partícula calcula los dos primeros factores principales de la ecuación de PSO:

Factor cognitivo:

$$\mathbf{p}_{local} - \mathbf{x}_i$$

Factor social:

$$\mathbf{p}_{global} - \mathbf{x}_i$$

El factor cognitivo es la resta vectorial entre la solución de *local best* y la solución actual. A su vez, el factor social es la resta vectorial entre la solución de *global best* y la solución actual. Teniendo estos dos factores, ya se puede estructurar una ecuación para la actualización de la velocidad de las partículas:

$$\mathbf{V}_{i+1} = \mathbf{V}_i + (\mathbf{p}_{local} - \mathbf{x}_i) + (\mathbf{p}_{global} - \mathbf{x}_i)$$

Esta se compone por ambos factores anteriormente descritos sumados a la velocidad actual de cada partícula. La eliminación de este nuevo factor agregado de memoria (velocidad actual) podría ser eliminado para simplicidad del algoritmo. Sin embargo, J. Kennedy y R. Eberhart descubrieron que eliminar dicho factor causa que el algoritmo de PSO sea ineficiente para encontrar una solución óptima de la función de costo [9]. Kennedy y Eberhart también descubrieron que se le debe aplicar cierta dinamicidad estocástica a dichos factores para que el enjambre de partículas no se mueva únicamente en una dirección común, sino que explore adecuadamente el espacio de búsqueda. Para esto, se utilizan parámetros de uniformidad para cada factor.

Parámetros de uniformidad:

$$\rho_{1,2} = rand() \in [0, \dots, 1]$$

Asimismo, descubrieron que darle un factor de escalamiento a los factores cognitivo y social cambiaba el comportamiento del enjambre. Un mayor peso al factor cognitivo resulta en una mayor dispersión de las partículas aumentando la capacidad de exploración del enjambre. Un mayor peso al factor social resulta en una mayor convergencia del enjambre hacia mínimos locales aumentando la capacidad de explotación de un punto específico [12].

Parámetros de escalamiento:

$$c_1, c_2 \in \mathbb{R}_{\geq 0}$$

En [9], Kennedy y Eberhart dieron una ponderación igual a ambos factores para tener un balance entre ambas características del enjambre. Sin embargo, no en todas las aplicaciones es útil dicha ponderación como se presenta en las siguientes subsecciones de esta investigación.

En la siguiente investigación de Particle Swarm Optimization llevada a cabo por Y. Shi y R. Eberhart en 1998 [3], también se introdujo un factor de ponderación ω denominado *inercia* para el componente de memoria. Se descubrió que este parámetro determinaba cuanto memoria retenía la partícula de su velocidad anterior y se observó que este parámetro variaba el comportamiento de convergencia del enjambre en la solución óptima. A su vez, se aplicó un parámetro de constricción φ para limitar los aumentos posibles de la velocidad y así acotar los valores posibles de esta (Nota: En las siguientes subsecciones se desarrolla más acerca de estos parámetros). Ya teniendo los parámetros de ponderación a utilizar, la ecuación se estructura finalmente como:

Ecuación de velocidad PSO:

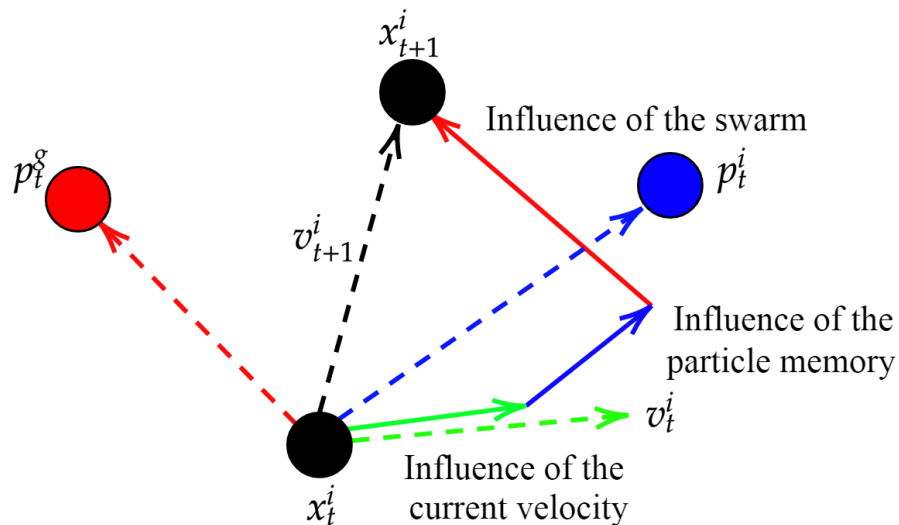
$$\mathbf{V}_{i+1} = \varphi[\omega \mathbf{V}_i + c_1 \rho_1(\mathbf{p}_{local} - \mathbf{x}_i) + c_2 \rho_2(\mathbf{p}_{global} - \mathbf{x}_i)]. \quad (1)$$

Teniendo la ecuación de actualización de velocidad de partícula, se puede estructurar la ecuación de actualización de posición para que las partículas converjan a la solución en donde la función de costo se minimiza o maximiza dependiendo la aplicación:

Ecuación de posición PSO:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{V}_{i+1} \quad (2)$$

Figura 6: Representación vectorial de ecuación de velocidad PSO [13]



6.1.3. Parámetros importantes del algoritmo

Parámetro de constricción

Este parámetro denotado por la letra φ , es una ponderación que se le da a la ecuación completa de velocidad PSO. El resultado de la suma de todos los factores ponderados cognitivo, social y de memoria se multiplican por este parámetro. Este sirve para ajustar la longitud de los pasos que cada partícula puede dar en cada iteración. La fórmula más popular para calcular el parámetro de constricción planteada en [14] es la siguiente:

$$\varphi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}; \quad \phi = c_1 + c_2; \quad \phi > 4 \quad (3)$$

Parámetro de inercia

Este parámetro denotado por la letra ω , es una ponderación que se le da al factor de memoria de la ecuación de velocidad PSO. Y. Shi y R. Eberhart investigaron acerca de este nuevo parámetro en [3] y determinaron que su valor puede cambiar la velocidad de convergencia del enjambre en el mínimo/máximo de la función de costo. En la experimentación que realizaron, se descubrió que valores de $\omega > 0.8$ favorecían una convergencia más lenta del enjambre dándole así más capacidad de exploración. Con valores de $\omega < 0.8$, se obtenía una convergencia más rápida hacia mínimos locales dándole al enjambre mayor capacidad de explotación. En [3], se plantea un parámetro de inercia decreciente en el tiempo para que al aumentar el número de iteraciones del algoritmo, este cambie el carácter explorativo inicial del enjambre a un carácter más convergente y de explotación. En [4], se plantean las siguientes nuevas fórmulas para calcular los diferentes valores del parámetro de inercia:

Inercia constante: Con este rango de valores de inercia, el enjambre aumenta su velocidad de convergencia y también modera la dispersión de las partículas asegurando la convergencia del algoritmo. Con estos valores se tiene mejor equilibrio entre exploración de mayor espacio y convergencia rápida hacia un mínimo.

$$0.8 < \omega < 1.2$$

Inercia lineal decreciente: La inercia comienza con un valor de 1.4 hasta disminuir a 0.5. Con esto, se le da mayor dispersión al enjambre en el inicio de la búsqueda para garantizar encontrar el mínimo global y luego se acelera la convergencia del enjambre hacia ese punto, haciendo más eficiente el algoritmo.

$$w = w_{max} - (w_{max} - w_{min}) * \frac{iter}{MAX_{iter}} \quad (4)$$

Inercia caótica: Primeramente, se elige un valor Z entre $[0,1]$ y se hace un mapeo de función logística. Con este valor, ya se calcula la inercia tomando en cuenta valores de ω_{min} y ω_{max} , así como el número de iteraciones que lleva ejecutadas el algoritmo. Esta función logra darle más precisión al enjambre a la hora de encontrar el mínimo global. Sin embargo, no brinda la convergencia más rápida:

$$Z_i \in [0, \dots, 1] \quad (5)$$

$$Z_{i+1} = 4 * Z_i * (1 - Z_i) \quad (6)$$

$$\omega = (\omega_{max} - \omega_{min}) * \left(\frac{MAX_{iter} - iter}{MAX_{iter}} \right) * \omega_{min} * Z_{i+1} \quad (7)$$

Inercia random: Se elije un número aleatorio y se suma a 0.5 para determinar la inercia de manera aleatoria. Esta asignación de inercia mejora la habilidad del enjambre de salir de mínimos locales y reduce el número total de iteraciones necesarias para que el algoritmo converja:

$$\omega = 0.5 + \frac{rand()}{2}; \quad rand() \in [0, \dots, 1] \quad (8)$$

Inercia natural exponencial: Esta estrategia de ecuación exponencial reduce el error promedio para que el enjambre logre llegar al punto mínimo y también permite convergencia en un número de iteraciones regular. Permite que al inicio de la ejecución del algoritmo se tenga una buena exploración del espacio de búsqueda y se tenga una aceleración exponencial de la velocidad de convergencia al aumentar las iteraciones.

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

6.2. Funciones de costo

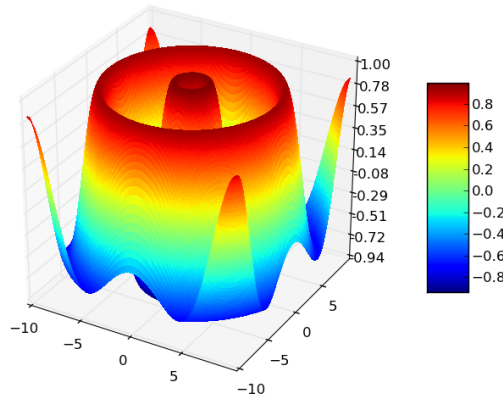
Para la evaluación de efectividad y eficiencia del algoritmo PSO se utilizan funciones de prueba o *benchmark functions* que presentan topologías con múltiples mínimos y máximos. Estas son útiles para determinar si los parámetros elegidos para el algoritmo son los adecuados. Se pueden elegir funciones simples con un solo mínimo para evaluar velocidades de convergencia o se pueden elegir funciones con varios mínimos locales y un mínimo absoluto para determinar la capacidad de exploración del algoritmo para encontrar el máximo absoluto sin converger erróneamente a algún mínimo local.

6.2.1. Función Schaffer F6

En [9], J. Kennedy y R. Eberhart utilizaron la función de test *Schaffer F6* que consta de múltiples oscilaciones y que permite evaluar la capacidad de exploración del PSO.

$$f(x, y) = 0.5 + \frac{\sin^2(\sqrt{x^2 + y^2}) - 0.5}{[1 + 0.001(x^2 + y^2)]^2} \quad (10)$$

Figura 7: Representación gráfica de función Schaffer F6 [15]



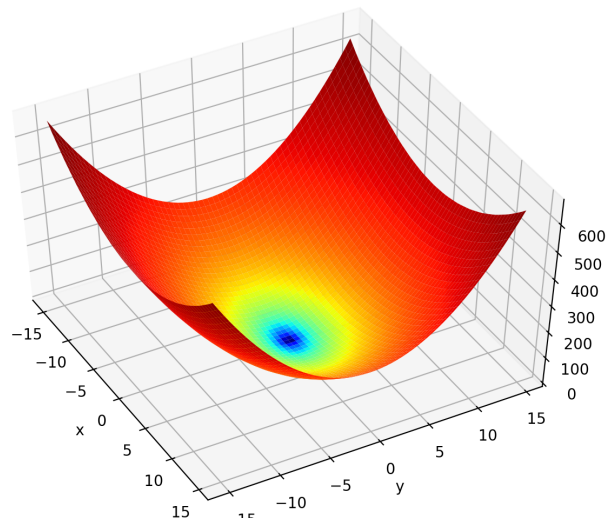
En la investigación de J. Bansal et al. [4], se utilizaron diferentes funciones de costo para evaluar el rendimiento del parámetro de *inercia* (ω). Las funciones que mejor evalúan la rapidez de convergencia y capacidad de exploración del PSO se presentan en las siguientes subsecciones.

6.2.2. Función Sphere

Esta función de costos, en su versión bidimensional, posee un mínimo absoluto en $[0,0]$. Es útil para evaluar velocidad de convergencia del enjambre por su simplicidad.

$$\text{Min}f(x_1, x_2, \dots, x_n) = \sum_{i=1}^n x_i^2 \quad (11)$$

Figura 8: Representación gráfica de función Sphere [15]

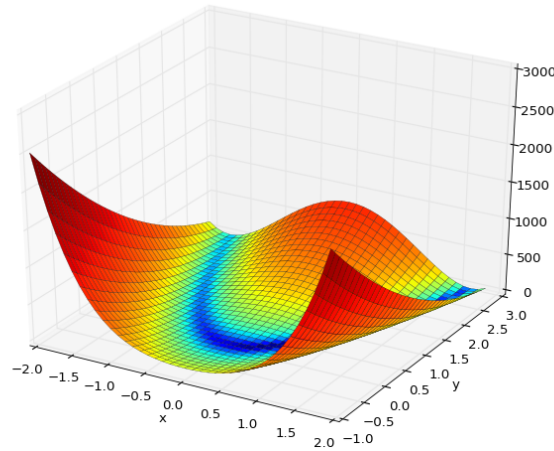


6.2.3. Función Rosenbrock

Esta función de costo, en su versión bidimensional, posee un mínimo en $[1,1]$. Es simple al igual que la función *Sphere*, pero es útil para evaluar el comportamiento de trayectorias de partículas en una nueva topología.

$$\text{Min}f(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \quad (12)$$

Figura 9: Representación gráfica de función Rosenbrock [15]

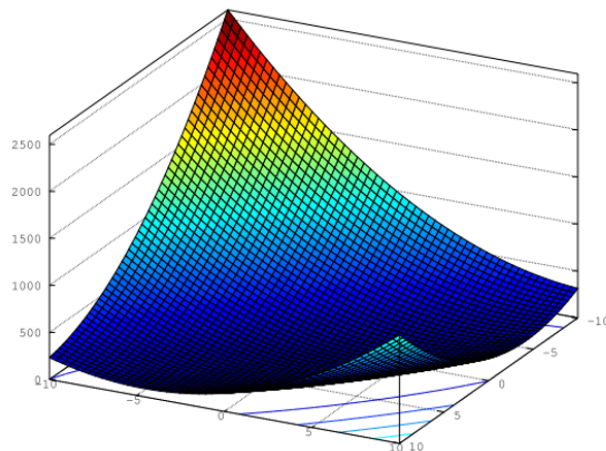


6.2.4. Función Booth

Es una función de costo simple que posee el mínimo absoluto descentrado del origen a diferencia de la función *Sphere*.

$$\text{Min}f(x) = (x + 2y - 7)^2 + (2x + y - 5)^2 \quad (13)$$

Figura 10: Representación gráfica de función Booth [15]

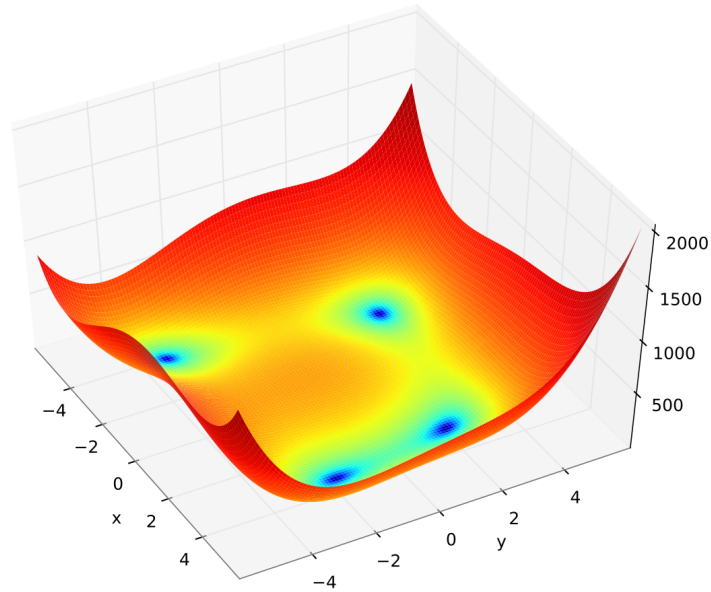


6.2.5. Función Himmelblau

Esta función de costo posee múltiples mínimos que poseen el mismo valor. Por lo tanto, todos son mínimos absolutos. Es útil para determinar la influencia de la posición inicial de las partículas sobre la decisión del punto de convergencia final.

$$\text{Min}f(x) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \quad (14)$$

Figura 11: Representación gráfica de función Himmelblau [15]



Otras funciones que se han utilizado en investigaciones como [16] para evaluar el rendimiento del algoritmo de PSO son:

- Función de Ackley
- Función de Rastrigin
- Función de Griewank
- Función de Schwefel
- Función de Schaffer F7
- Función de Beale
- Función de Goldstein-Price
- Función Gaussiana de Lunacek

6.3. Robots diferenciales

Existen diferentes tipos de robots móviles que son capaces de trasladarse en un medio planar por medio de actuadores rotacionales. Dentro de esta categoría se encuentran los robots diferenciales. Los robots diferenciales se destacan por tener dos ruedas montadas sobre un mismo eje de rotación. Cada rueda es propulsada independientemente por su propio actuador. Esto permite controlar tanto la tracción como la orientación del robot con las mismas ruedas. Los robots diferenciales, al poder ser controlados con solamente dos motores y dos ruedas, son ideales para aplicaciones en donde se requiere que estos posean dimensiones físicas reducidas. Con este tipo de diseño de robot móvil, se pueden realizar movimientos lineales y movimientos rotacionales sin traslación. El control de este tipo de robots se reduce al envío adecuado de velocidades angulares a cada uno de los actuadores para poder describir las curvas de movimiento necesarias [17].

6.3.1. Modelado de robots diferenciales

En el capítulo *Wheeled Mobile Robots: Nonholonomic Modeling* del libro “Modern Robotics” de Kevin Lynch [18], se presentan los modelos de unicyclo y de robots diferenciales que se explican más a detalle a continuación. Para lograr describir las curvas requeridas con los robots diferenciales, se debe llevar a cabo la transformación de las velocidades lineales y angulares del robot a las velocidades angulares de cada una de las ruedas. Primeramente, se debe definir la transformación que se realizará para cada rueda individualmente y luego se toma en cuenta ambos actuadores para obtener la cinemática del robot.

Modelo unicyclo

Para comenzar el modelado completo de la cinemática del robot diferencial, primero se debe partir del modelo unicyclo para relacionar la velocidad angular de una rueda con eje de rotación paralelo al piso y la velocidad lineal de la rueda respecto al piso. Utilizando la relación normal entre velocidades lineales y angulares se obtiene que:

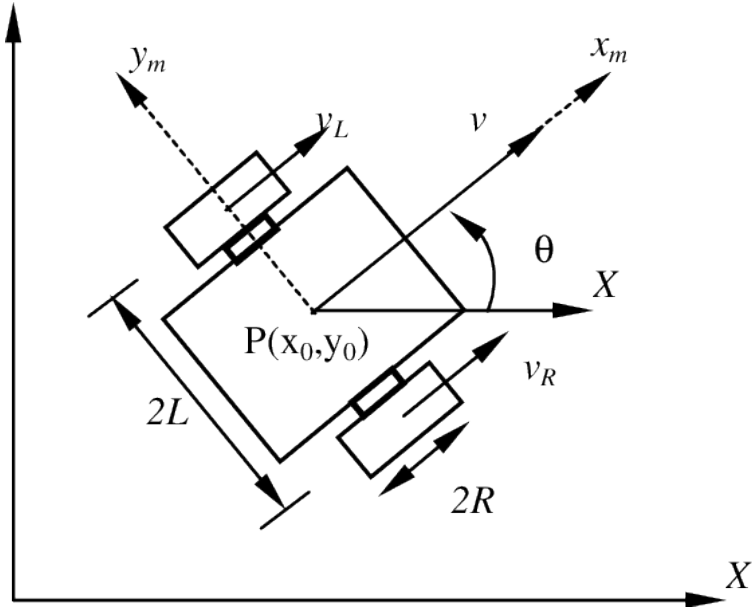
$$v_R = \Phi_R r; \quad v_L = \Phi_L r, \quad (15)$$

siendo v_R y v_L las velocidades lineales de las ruedas, Φ_R y Φ_L las velocidades angulares de las ruedas y r el radio de las ruedas [18].

Modelo diferencial

Con las ecuaciones anteriores (15) planteadas, ya se puede proseguir con el cálculo de la cinemática completa. En este caso, se desea relacionar las velocidades lineales de cada llanta del robot a la velocidad lineal del centro de masa y su velocidad angular. Se plantea un modelo como el que se observa en la siguiente figura:

Figura 12: Modelo de robot diferencial [19]



Velocidad lineal:

Se observa que la velocidad lineal del centro de masa del robot diferencial es el promedio entre las velocidades lineales de cada llanta. Por lo tanto, se puede plantear la ecuación como:

$$v = \frac{v_R + v_L}{2}, \quad (16)$$

y al combinar esta ecuación con el modelo unicycle planteado en (15), se deriva la siguiente expresión de v en términos de Φ_R y Φ_L :

$$2v = \Phi_R r + \Phi_L r \quad (17)$$

Velocidad angular:

Para la velocidad angular, se utiliza el planteamiento de centro instantáneo para relacionar las velocidades lineales de cada llanta con la rotación del robot completo. Primero, se coloca el centro instantáneo en la llanta izquierda del robot y se observa que:

$$\omega = \frac{v_R}{2l} \quad (18)$$

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

$$\omega = \frac{-v_L}{2l} \quad (19)$$

Por el método de superposición, se suma (18) y (19), y se obtiene que la relación entre la velocidad angular del robot y las velocidades lineales de las llantas es:

$$\omega = \frac{v_R - v_L}{2l}, \quad (20)$$

y reescribiendo esta ecuación en términos del modelo unicyclo planteado en (15), se obtiene:

$$2\omega l = \Phi_{Rr} - \Phi_{Lr} \quad (21)$$

Transformación de velocidades:

Para despejar las velocidades angulares de las llantas en términos de v y ω , primeramente, se suman las dos ecuaciones de velocidades lineal (17) y angular (21), y se obtiene:

$$\Phi_R = \frac{v + \omega l}{r} \quad (22)$$

Para lograr estructurar la ecuación de la velocidad angular de la llanta izquierda, se restan las mismas ecuaciones y se obtiene:

$$\Phi_L = \frac{v - \omega l}{r} \quad (23)$$

Ya con estas ecuaciones determinadas, se puede enviar el dato de velocidad angular directamente a los actuadores de las llantas. Para describir cierta curva de movimiento, se obtiene las velocidades lineal y angular necesarias del robot para obtener las velocidades requeridas en cada eje planar y sobre el eje de rotación:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad (24)$$

y por medio de las transformaciones establecidas con el modelo diferencial en (22) y (23), se determinan las velocidades de los motores (18).

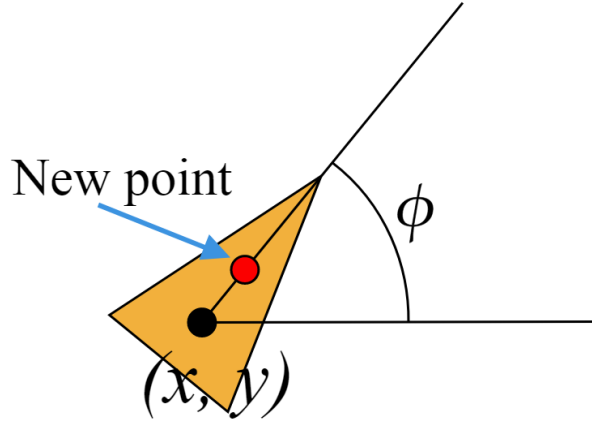
6.3.2. Difeomorfismo para transformación de cinemática de unicyclo

Al observar las ecuaciones de cinemática anteriores, queda claro que el sistema dinámico de un robot diferencial es altamente no lineal. Este sistema resulta ser no controlable con la cantidad de entradas de velocidad que posee este sistema y por lo tanto aplicarle control directamente no daría los resultados que se esperarían. Por lo tanto, es conveniente pensar en una forma de reducir la información necesaria del robot para su control adecuado. En muchas aplicaciones, solo se conocen los componentes de velocidad lineal de un punto en específico que no posee velocidad angular. Como se expone en (20), se puede realizar el siguiente ajuste para lograr controlar un robot diferencial con solo conocer la velocidad de un punto específico y orientación ϕ del robot.

Primeramente, se toma un punto a una distancia a del centro y al frente del robot diferencial a controlar. Si la geometría de la base del robot es cilíndrica, esta distancia a es igual a la distancia l entre el centro y las ruedas del robot. Por lo tanto, la posición de este punto está dada por:

$$\tilde{x} = x + l\cos\phi; \quad \tilde{y} = y + l\sin\phi \quad (25)$$

Figura 13: Punto de transformación de cinemática [21]



Se sabe que la velocidad del centro del robot está dada por las ecuaciones (24). Teniendo la posición del nuevo punto de control expresada como (25), se puede escribir la velocidad del nuevo punto como:

$$\dot{\tilde{x}} = \dot{x} - l\dot{\phi}\sin\phi = v\cos\phi - l\omega\sin\phi \quad (26)$$

$$\dot{\tilde{y}} = \dot{y} + l\dot{\phi}\cos\phi = v\sin\phi + l\omega\cos\phi \quad (27)$$

Asumiendo que podemos controlar directamente las velocidades del nuevo punto con las entradas u_1 y u_2 , y reescribiendo (26) y (27) de forma matricial con una matriz de rotación $R(\phi)$, una matriz de longitud y un vector de velocidades, se obtiene:

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & l \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad (28)$$

y despejando para las velocidades v y ω en términos de las entradas, se obtiene la ecuación:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{l} \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (29)$$

NOTA: en este caso se observa la necesidad de procurar que la distancia l del robot no sea muy pequeña y cause que la matriz de longitud inversa se vuelva singular.

Ecuación de cinemática transformada de unicycle: Con (29) establecida, finalmente obtenemos la expresión necesaria para poder determinar las velocidades lineal y angular de un robot diferencial a partir de los componentes de velocidad de un punto único:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\frac{1}{l}\sin\phi & \frac{1}{l}\cos\phi \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (30)$$

6.4. Controladores de posición y velocidad de robots diferenciales

6.4.1. Control proporcional de velocidades con saturación limitada

En el control de robots diferenciales, es común observar aplicaciones de control en donde se utiliza el error de posición, orientación y velocidad para determinar las velocidades de referencia que el robot debe recibir para describir una trayectoria definida. Como se muestra en el trabajo de investigación [20], las velocidades de entrada u_1 y u_2 utilizadas en (30), se calculan de la siguiente manera:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_d + I_x \cdot \tanh\left(\frac{k_x}{I_x} \cdot \tilde{x}\right) \\ \dot{y}_d + I_y \cdot \tanh\left(\frac{k_y}{I_y} \cdot \tilde{y}\right) \end{bmatrix}, \quad (31)$$

en donde $\dot{h}_d = [\dot{x}_d \ \dot{y}_d]$ son las velocidades del punto a seguir, $I_x, I_y \in \mathbb{R}_{\geq 0}$ son constantes de saturación, k_x y k_y son constantes del controlador, y $\tilde{h} = [(x_d - x) \ (y_d - y)]$ es el vector de errores de posición del robot respecto al punto objetivo. En el caso de un robot diferencial simple siguiendo una trayectoria suave, el vector \dot{h}_d es nulo y las constantes k e I se pueden colocar de la misma magnitud en ambos ejes (x, y).

Las constantes de saturación I junto con la función $\tanh(x)$, son utilizadas para evitar que las velocidades resultantes de referencia para el robot sean excesivamente grandes en caso el error inicial de posición sea demasiado grande. La magnitud de la cota superior de las velocidades es la magnitud de la constante de saturación.

6.4.2. Control PID de velocidad angular ω

En el control de robots móviles, es común encontrar la implementación de controladores PID para ajustes de posición y velocidad como se muestra en [21]. Estos controladores permiten la variación de tres parámetros k_p , k_i y k_d para modificar el comportamiento del sistema. En aplicaciones con robots diferenciales, se suele utilizar un controlador proporcional simple para el control de la velocidad lineal v y un controlador PID para la velocidad angular ω . En el caso de robots diferenciales que utilizan la transformación del modelo unicycle presentada en (30), y que requieren la velocidad de un solo punto, se puede prescindir de un controlador PID y solo utilizar controladores proporcionales como se muestra en (31).

[1] Ecuación general de controlador PID:

$$\omega = K_p e(t) + K_i \int_0^t e(t) dt + K_v \frac{de(t)}{dt} \quad (32)$$

6.4.3. Control proporcional de velocidad lineal v

Las constantes k_x y k_y de los controladores proporcionales en (31) pueden ser arbitrarias y ajustadas empíricamente dependiendo su rendimiento en diferentes aplicaciones. Sin

¹*Nota: Ejemplo de implementación de controlador PID en lenguaje C se puede encontrar en anexos

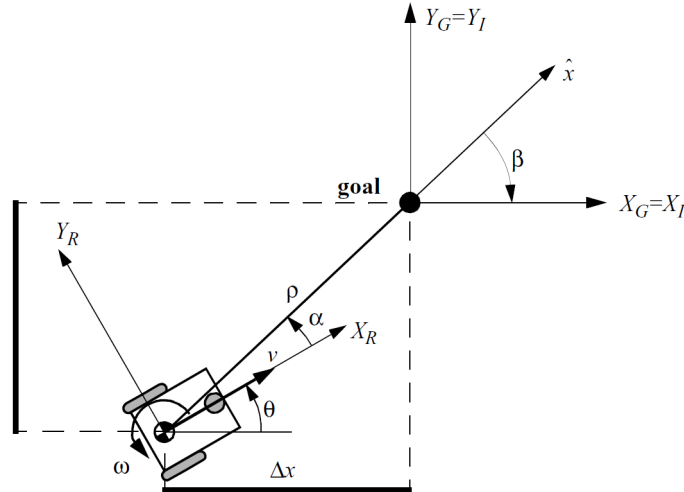
embargo, en aplicaciones en donde un robot debe seguir una trayectoria definida para llegar a una meta, lo anterior puede ser no adecuado. En el caso de robots que deben alcanzar una meta, se prefiere utilizar parámetros variables en el tiempo que permitan modificar el comportamiento del sistema mientras más tiempo transcurre. Para este caso, se pueden utilizar parámetros k_x y k_y en función del error de posición del robot respecto a su meta. Se busca que la velocidad de convergencia hacia la meta decrezca mientras el robot se aproxima más a la meta para no sobrepasarla accidentalmente y causar un comportamiento oscilatorio al alcanzarla. En [21], se utiliza:

$$K_p = \frac{v_0 \cdot (1 - e^{-\alpha \|e_p\|^2})}{\|e_p\|}; \quad \alpha, v_0 \in \mathbb{R}_{>0} \quad (33)$$

6.4.4. Controlador de pose

Los controladores presentados anteriormente facilitan el manejo de robots diferenciales para llegar a la meta establecida. Sin embargo, estos no toman en cuenta la pose u orientación final que el robot tendrá al llegar a la meta. La pose final del robot depende de la pose inicial, y tomar en cuenta estos datos al controlar el robot permite lograr trayectorias más suaves de convergencia a la meta. En la sección 3.6 *Motion Control (Kinematics Control)* del libro “Introduction to Autonomous Mobile Robots” de Roland Siegwart [17] se presenta el siguiente controlador de pose.

Figura 14: Cinemática de robot y referencias de interés [17]



Primeramente, el error de posición entre el robot y el punto de meta se expresa en el marco de referencia del robot de la manera:

$$\mathbf{e} = {}^R [x, y, z]^T \quad (34)$$

Se desea encontrar una matriz \mathbf{K} tal que el control dado por

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \mathbf{K} \cdot \mathbf{e} = \mathbf{K} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}^R \quad (35)$$

cause que $\lim_{t \rightarrow \infty} e(t) = 0$. La relación entre las velocidades cartesianas y las velocidades angular ω y lineal v del robot diferencial está dada por (24). Y ahora, esta relación se debe trasladar a un espacio de coordenadas polares sabiendo que α es el ángulo entre el eje X_R del marco referencial del robot y el vector entre el centro del robot y el punto de meta como se observa en la Figura 14, β es el ángulo de orientación entre el eje horizontal del marco inercial y el vector entre el robot y la meta, y ρ es la distancia euclideana del vector entre el centro del robot y el punto de meta.

Ecuaciones polares de coordenadas:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \quad (36)$$

$$\alpha = -\theta + \text{atan2}(\Delta y, \Delta x) \quad (37)$$

$$\beta = -\theta - \alpha \quad (38)$$

Ahora la relación de velocidades en coordenadas polares estaría dada por la siguiente ecuación matricial (siempre y cuando el robot se encuentre viendo frontalmente hacia la meta, es decir que $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$):

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos(\alpha) & 0 \\ \frac{\sin(\alpha)}{\rho} & -1 \\ \frac{-\sin(\alpha)}{\rho} & 0 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (39)$$

Si en caso el robot estuviera orientado de tal manera que la meta se encuentra detrás del mismo ($\alpha \in [\pi, -\frac{\pi}{2}] \cup [\frac{\pi}{2}, \pi]$), la relación matricial entre velocidades estaría dada por:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & 0 \\ -\frac{\sin(\alpha)}{\rho} & 1 \\ \frac{\sin(\alpha)}{\rho} & 0 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (40)$$

Como se puede observar, las ecuaciones anteriores se indefinen cuando el robot converge al origen del sistema $(\rho, \alpha, \beta) = [0, 0, 0]$. En este caso, la meta es el origen del sistema y es el punto que se desea alcanzar. Por lo tanto, se estructuran las siguientes leyes de control para evitar este problema y controlar las velocidades lineal y angular del robot:

Leyes de controlador de pose:

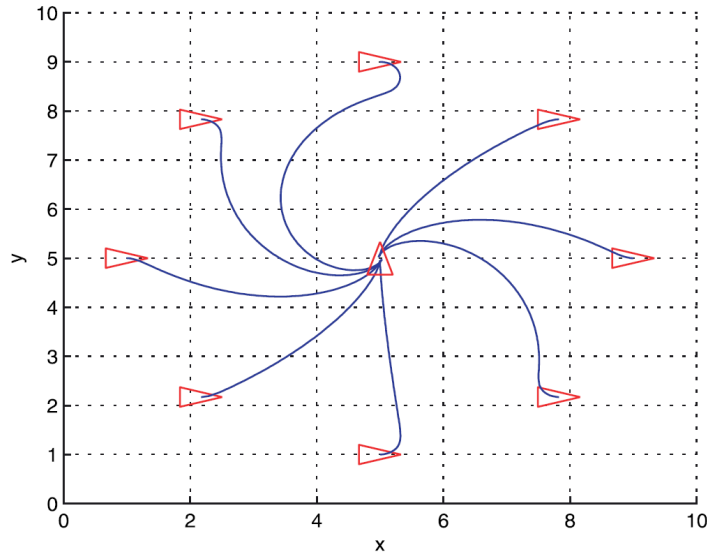
$$v = k_\rho \cdot \rho \quad (41)$$

$$w = k_\alpha \cdot \alpha + k_\beta \cdot \beta \quad (42)$$

Diciéndolo de otra manera, los factores de ρ y α en las leyes de control ayudan al robot a orientarse y seguir la línea entre el robot y la meta, mientras que el factor de β orienta dicha línea para lograr la pose final al llegar a la meta. En [17], se presenta la evaluación de estabilidad de este controlador dependiendo las constantes multiplicadoras de cada variable y se plantean las siguientes condiciones para garantizar la convergencia hacia una meta definida:

$$\begin{aligned} k_\rho &> 0 \\ k_\beta &< 0 \\ k_\alpha - k_\rho &> 0 \end{aligned}$$

Figura 15: Trayectorias resultantes de convergencia con controlador de pose [22]



6.4.5. Controlador de pose con criterio de estabilidad de Lyapunov

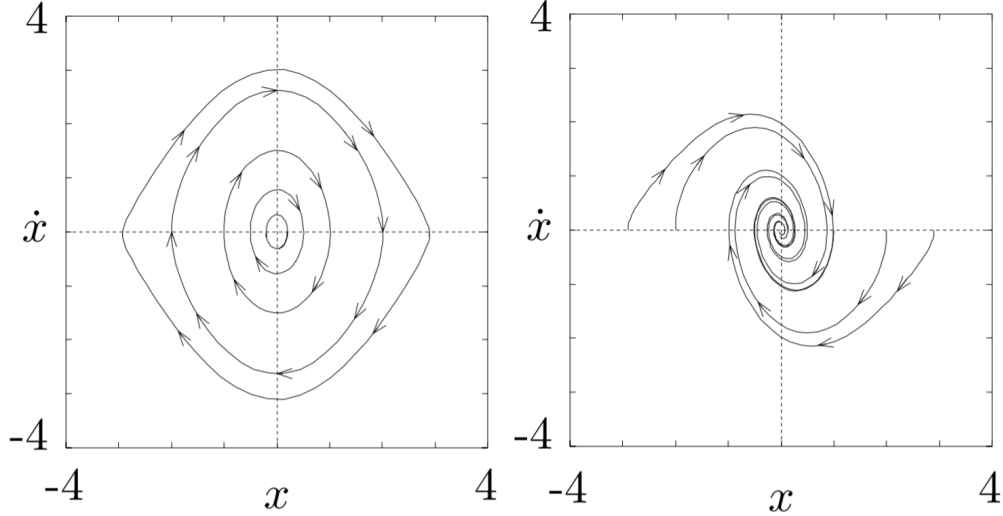
En la investigación [23], se analiza el uso de un controlador de pose similar al presentado en la subsección anterior, pero utilizando leyes de control para las velocidades lineal y angular un tanto diferentes. En este caso, se utiliza la misma conversión de la pose del robot diferencial a coordenadas polares [36]. La relación de velocidades está dada por [39] o [40] dependiendo del rango en donde se encuentre el ángulo α . El objetivo de este controlador es que el estado del sistema converja a $(\rho, \alpha, \beta) = [0, 0, \beta]$ en un tiempo finito.

Para asegurar la convergencia asintótica del robot hacia la meta, se utiliza el Criterio de Estabilidad de Lyapunov. Este método de estabilidad de sistemas mecánicos con soluciones cerca de puntos de equilibrio obtiene su nombre del matemático ruso Aleksandr Mikhailovich Lyapunov. [2] En resumen, el criterio de Lyapunov afirma que, si existe una solución en las cercanías de un punto de equilibrio Lyapunov-estable X_o de una ecuación diferencial homogénea $\dot{X} = f(X)$, esta se mantiene cerca de X_o para todo $t > t_o$. Asimismo, este criterio afirma que el punto de equilibrio X_o es asintóticamente estable si posee estabilidad

²Más elaboración sobre este tema se puede encontrar en el capítulo No.4 *Lyapunov Stability Theory* del libro “A mathematical introduction to robotic manipulation” de R.M. Murray [24].

de Lyapunov y además es un atractor de soluciones que convergen a X_o cuando $t \rightarrow \infty$.

Figura 16: Estabilidad de Lyapunov vs. estabilidad asintótica [24]



Para determinar las leyes de control necesarias para lograr que el robot converja al origen del sistema (o meta), se parte de los criterios de Lyapunov para que el sistema sea asintóticamente estable. Para estructurar una función Lyapunov definida positiva, se toma en cuenta la medida de “energía interna” [24] que posee el sistema. En este caso, la energía del sistema depende de los parámetros ρ y α que son los que deben converger a cero para llegar al punto de equilibrio. Por lo tanto:

$$V = V_1 + V_2 \quad (43)$$

$$V = \frac{1}{2}\rho^2 + \frac{1}{2}\alpha^2 \quad (44)$$

Luego, se deriva esta función respecto al tiempo para obtener la expresión (45) y reemplazando las derivadas de ρ y α por las expresiones de las velocidades lineal y angular del robot definidas en (39), se obtiene (46):

$$\dot{V} = \dot{\rho}\rho + \dot{\alpha}\alpha \quad (45)$$

$$\dot{V} = \rho \cdot (-v\cos(\alpha)) + \alpha \cdot \left(-\omega + \frac{v}{\rho}\sin(\alpha)\right) \quad (46)$$

Se puede observar que, para que los términos \dot{V}_1 y \dot{V}_2 sean no positivos se pueden definir las velocidades lineal y angular del robot como:

$$v = K_\rho \rho \cdot \cos(\alpha); \quad K_\rho > 0 \quad (47)$$

$$\omega = K_\rho \sin(\alpha) \cos(\alpha) + K_\alpha \alpha; \quad K_\alpha > 0 \quad (48)$$

Al reemplazar las expresiones anteriores (47) y (48) en la derivada de la función de Lyapunov expresada en (46) se obtiene:

$$\dot{V} = -K_\rho \rho^2 \cdot \cos^2(\alpha) - K_\alpha \alpha^2 \leq 0 \quad (49)$$

Por lo tanto, \dot{V} es una función negativa semi-definida. Y ahora, para finalmente evaluar la existencia de convergencia hacia el punto de equilibrio, se aplica el siguiente teorema:

Lema de Barbalat:

- Si $V(t)$ posee un límite finito cuando $t \rightarrow \infty$, y $\dot{V}(t)$ es uniformemente continua, entonces $V(t) \rightarrow 0$ cuando $t \rightarrow \infty$. [25]

Se puede observar que $V(t)$ expresada en (44) efectivamente posee un límite finito (cero) cuando $t \rightarrow \infty$, y $\dot{V}(t)$ expresada en (49) es uniformemente continua. Por lo tanto, $V(t) \rightarrow 0$ cuando $t \rightarrow \infty$. Esto causa que el error de posición se reduzca a cero llevando el estado del sistema a $(\rho, \alpha, \beta) = [0, 0, \beta]$. Con esto, se puede concluir que las expresiones dadas en (47) y (48) para las velocidades lineal y angular del robot sirven como leyes de control que logran llevar al robot diferencial a una meta por medio de una trayectoria suave y estable [24].

6.4.6. Controlador de lazo cerrado de direccionamiento de robot

En la investigación [26], se plantea el uso de un controlador de pose similar a los propuestos en las subsecciones anteriores y utilizando los mismos marcos de referencia observados en la Figura 14.

Para el planteo de este controlador, se utiliza la relación de velocidades planteadas en (39). Se plantean las siguientes expresiones para comenzar con el desarrollo del controlador:

$$\dot{\alpha} = -\omega - \dot{\beta} \quad (50)$$

$$\alpha^* = -\arctan(k_1\beta) \quad (51)$$

$$z = (-\arctan(-k_1\beta)) - \alpha \quad (52)$$

Se puede observar que la expresión (50) se puede deducir de las ecuaciones planteadas en (39) modificandolas con un poco de álgebra. La expresión (51) denota un controlador virtual que se le aplica al ángulo α del robot. La expresión (52) es la diferencia entre el controlador de α y el ángulo verdadero. Como siguiente paso, se calcula la tasa de cambio de la diferencia entre el controlador de α y el ángulo real. Esta se determina al derivar la expresión (52):

$$\dot{z} = \frac{d}{dt}(-\arctan(-k_1\beta)) - \dot{\alpha} \quad (53)$$

$$\dot{z} = \frac{k_1}{1 + (k_1\beta)^2} \dot{\beta} - (-\omega - \dot{\beta}) \quad (54)$$

Luego, como siguiente operación, se reemplaza el valor de $\dot{\beta}$ con su expresión correspondiente planteada en (39):

$$\dot{z} = \omega + \left(1 + \frac{k_1}{1 + (k_1\beta)^2}\right) \cdot \dot{\beta} \quad (55)$$

$$\dot{z} = \omega + \left(1 + \frac{k_1}{1 + (k_1\beta)^2}\right) \cdot \left(-\frac{v}{\rho} \sin(\alpha)\right) \quad (56)$$

De esta última operación, se despeja la velocidad angular del robot y se reemplaza \dot{z} por la expresión (57) para finalmente obtener la ecuación (58):

$$\dot{z} = -k_2 \frac{v}{\rho} z \quad (57)$$

$$\omega = \left(-k_2 \frac{v}{\rho} z\right) - \left(1 + \frac{k_1}{1 + (k_1 \beta)^2}\right) \cdot \left(-\frac{v}{\rho} \sin(\alpha)\right) \quad (58)$$

Con estas expresiones ya planteadas, se puede finalmente estructurar la ley de control a utilizar para la velocidad angular ω del robot. Solamente es necesario reemplazar z por la expresión dada en (52) y reordenar los términos:

$$\omega = \frac{2}{5} \cdot \left(-k_2 \frac{v}{\rho} (-\alpha - \arctan(-k_1 \beta))\right) - \left(1 + \frac{k_1}{1 + (k_1 \beta)^2}\right) \cdot \left(-\frac{v}{\rho} \sin(\alpha)\right) \quad (59)$$

$$\omega = \frac{2}{5} \cdot \frac{v}{\rho} \left[k_2 (\alpha + \arctan(-k_1 \beta)) + \left(1 + \frac{k_1}{1 + (k_1 \beta)^2}\right) \cdot (\sin(\alpha))\right] \quad (60)$$

Ya teniendo la ley de control para la velocidad angular ω del robot (60), se puede observar que la velocidad lineal v del robot es una variable libre. Por lo tanto, la única restricción que se le debe aplicar a esta variable para garantizar la convergencia del controlador es que $v \rightarrow 0$ cuando $t \rightarrow \infty$. Con un controlador proporcional simple dependiente del error decreciente de posición del robot respecto a la meta es suficiente (26).

Figura 17: Trayectorias de robot con controlador de direccionamiento ($k_1 = 1, k_2 = 3$) [26]

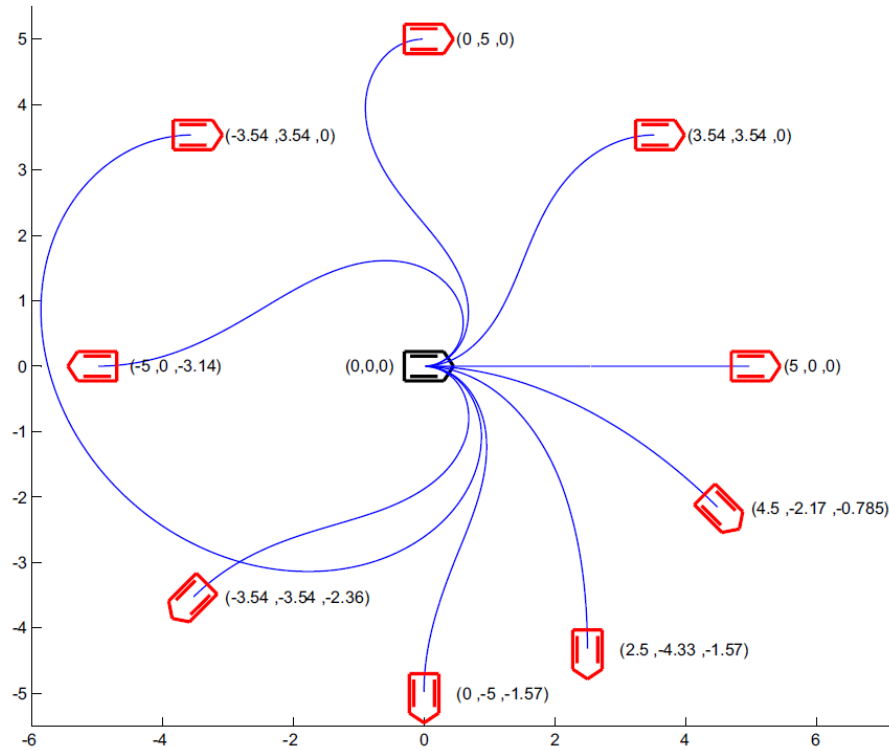
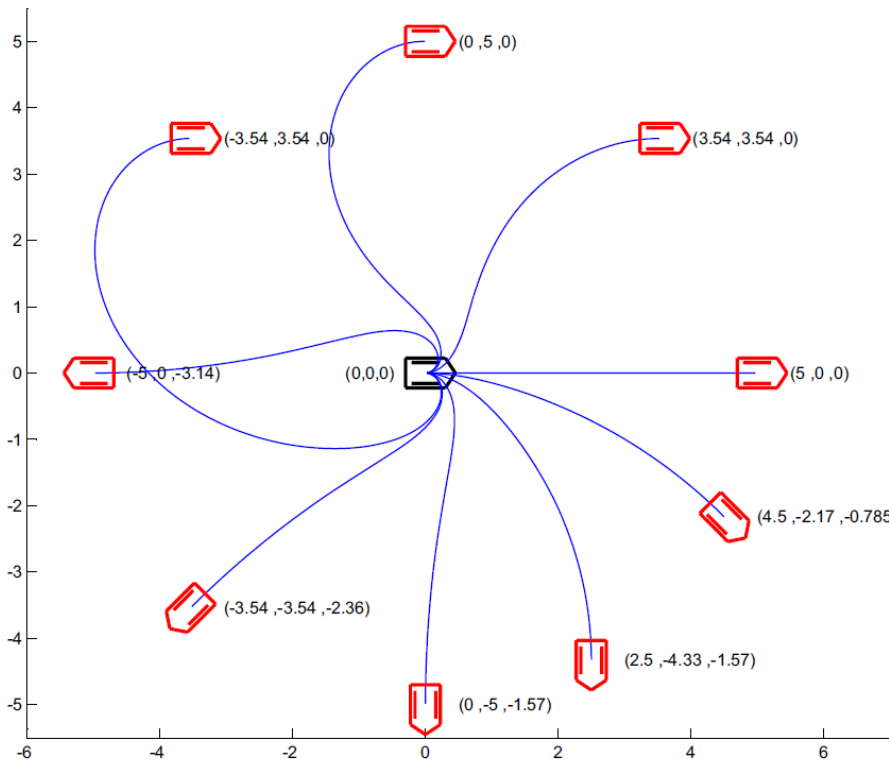


Figura 18: Trayectorias de robot con controlador de direccionamiento ($k_1 = 1, k_2 = 10$) [26]



6.4.7. Control por medio de regulador lineal cuadrático (LQR)

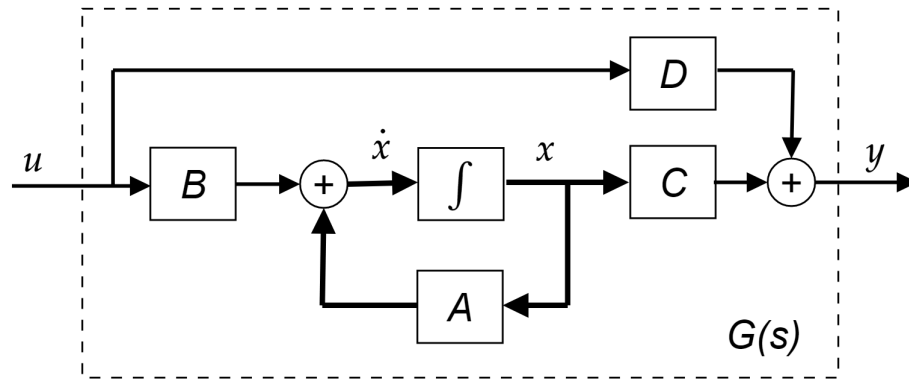
Representación de espacio de estados:

Para el control de robots diferenciales también se puede plantear un análisis del modelo mediante su representación en espacio de estados. En este caso, el sistema se puede plantear como linealmente invariante en el tiempo (LTI) de la siguiente manera:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u}) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \implies \begin{cases} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \quad (61)$$

En donde \mathbf{x} es el vector de las variables de estado del sistema (posición y velocidad), \mathbf{u} es el vector de entradas del sistema, $\dot{\mathbf{x}}$ es el cambio de estado del sistema, y \mathbf{y} es el vector de salidas del sistema [27].

Figura 19: Función de transferencia de sistema en representación LTI de espacio de estados



En el caso de los robots diferenciales, el cambio de estado del sistema no depende del estado actual, sino únicamente de las entradas del controlador no escalado. Por lo tanto, la matriz de sistema \mathbf{A} es cero y la matriz de control \mathbf{B} es la identidad. La salida del sistema es únicamente el estado del mismo, por lo que la matriz de salida \mathbf{C} es la identidad y la matriz de *feed-forward* \mathbf{D} es cero.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{u} \\ \mathbf{y} = \mathbf{x} \end{cases} \quad (62)$$

Regulador lineal cuadrático

La motivación detrás del uso de un controlador LQR es lograr llevar a cabo el control del sistema de manera óptima. Esto quiere decir que se desea llevar al sistema a un estado determinado de la manera más rápida posible utilizando lo más mínimo posible de control. Para llevar a cabo el diseño de este tipo de controlador, se debe ejecutar un proceso de optimización de algún criterio que defina el desempeño del controlador [27]. El controlador

LQR parte de un criterio de desempeño J que se define de la siguiente manera:

$$J = \int_0^{t_f} \|x(t)\|^2 + \|u(t)\|^2 dt \quad (63)$$

Esta expresión define el desempeño del controlador como la suma de las áreas bajo las curvas tanto de la magnitud del vector de estados $x(t)$ respecto al tiempo como de la magnitud del control $u(t)$ respecto al tiempo. Se desea minimizar el área bajo la curva respecto a $x(t)$ para reducir el estado transitorio y evitar trayectorias irregulares del cambio de estados. Y se desea minimizar el área bajo la curva de la magnitud del control $u(t)$ para evitar utilizar excesivo control sobre el sistema. El criterio de desempeño J se expresa vectorialmente de la siguiente manera:

$$J = \int_0^{t_f} (x^T \mathbf{Q}x + u^T \mathbf{R}u) \cdot dt \quad (64)$$

En donde \mathbf{Q} y \mathbf{R} son matrices no nulas utilizadas para ponderar el efecto de $x(t)$ y $u(t)$ sobre J . Usualmente estas matrices son variantes o modificaciones de la matriz identidad. Si se establece una ponderación $\mathbf{R} \gg \mathbf{Q}$, el control es “caro”, y por lo tanto el sistema penaliza el uso de un control $u(t)$ muy grande. Si $\mathbf{Q} \gg \mathbf{R}$, $x(t)$ ejerce más peso sobre el criterio de desempeño y el sistema no penaliza el uso de un control muy grande [27]. En el caso de los robots diferenciales, se desea penalizar el uso de controles muy grandes. Por lo tanto, se utiliza la condición $\mathbf{R} \gg \mathbf{Q}$.

Teniendo la expresión del criterio de desempeño (64), se observa que la ley de control que minimiza dicho criterio J estaría dada por:

$$\mathbf{u} = -\mathbf{K}\mathbf{x} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{x} \quad (65)$$

En donde \mathbf{R} es la matriz de ponderación del control dentro de J , \mathbf{B} es la matriz de entradas de la expresión (61) (matriz identidad en el caso planteado en (62)), y \mathbf{P} es la solución a la Ecuación Algebraica de Riccati (ARE):

$$-\mathbf{A}^T\mathbf{P} - \mathbf{P}\mathbf{A} - \mathbf{Q} + \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} = \mathbf{0} \quad (66)$$

En el caso de los robots diferenciales, se plantea un problema de rastreo de metas. Por lo tanto, se desea minimizar el error de posición respecto a puntos de operación $(\mathbf{x}_g, \mathbf{u}_g)$ en vez de al origen. Por esta razón, la ley de control por retroalimentación se reescribe como:

$$\mathbf{u} = -\mathbf{K} \cdot (\mathbf{x} - \mathbf{x}_g) + \mathbf{u}_g \quad (67)$$

6.4.8. Controlador lineal cuadrático integral (LQI)

Como se observó en la sección anterior, el regulador LQR permite la estabilización de un sistema dinámico alrededor de un punto de equilibrio o de operación. Sin embargo, en muchas aplicaciones existe incertidumbre entre el modelo del sistema y el sistema real, y el LQR es sensible a estas inexactitudes. Para lograr reducir este error de incertidumbre del LQR, se le puede acoplar acción integradora como lo hace el PID en control clásico [28].

Para estructurar este controlador, se plantea el sistema del error entre la referencia y las salidas del sistema como se observa en (68). Luego, se combina este sistema con el sistema LTI estándar, como se observa en (69):

$$\dot{\mathbf{x}}_I = \mathbf{e} = \mathbf{r} - \mathbf{y} = \mathbf{r} - \mathbf{C}\mathbf{x} \quad (68)$$

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{x}}_I \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{n \times n} & \mathbf{0}_{n \times p} \\ -\mathbf{C}_{p \times n} & \mathbf{0}_{p \times p} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_I \end{bmatrix} + \begin{bmatrix} \mathbf{B}_{n \times n} \\ \mathbf{0}_{p \times n} \end{bmatrix} \mathbf{u} + \begin{bmatrix} \mathbf{0}_{n \times p} \\ \mathbf{I}_{p \times p} \end{bmatrix} \mathbf{r} \quad (69)$$

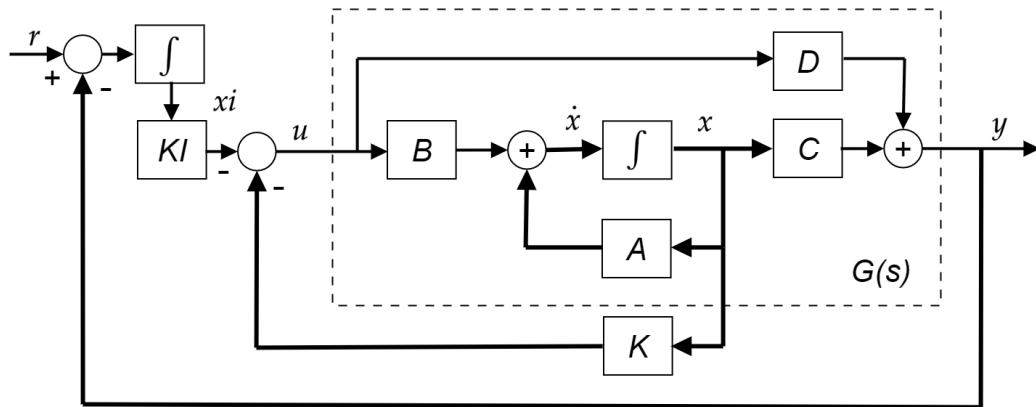
Al plantear el sistema de esta manera, se puede utilizar el mismo cálculo de LQR para obtener la matriz $\bar{\mathbf{K}}$ a partir de las nuevas matrices \mathbf{A} y \mathbf{B} del sistema planteado. Con esto, se puede plantear la retroalimentación negativa de estados más la acción integradora como:

$$\mathbf{u} = -\bar{\mathbf{K}}\bar{\mathbf{x}} = [\mathbf{K}_{m \times n} \quad \mathbf{K}_{Im \times p}] \cdot \begin{bmatrix} \mathbf{x}_{n \times 1} \\ \mathbf{x}_{Ip \times 1} \end{bmatrix} = -\mathbf{K}\mathbf{x} - \mathbf{K}_I\mathbf{x}_I \quad (70)$$

En el caso de robots diferenciales, se puede combinar la retroalimentación del error entre la posición actual del robot y la meta, y la integral del error acumulado entre la referencia (meta) y la posición actual.

$$\mathbf{u} = -\mathbf{K}\mathbf{e} - \mathbf{K}_I \int_0^t (\mathbf{r} - \mathbf{y}) d\tau \quad (71)$$

Figura 20: Estructura de controlador lineal cuadrático integral



6.5. Cálculo de suavidad de curvas por método de energía de deflexión de vigas

Muchas aplicaciones basadas en simulaciones computacionales producen listas de datos discretos. Estos datos se pueden procesar de diferentes maneras para analizar las características de una simulación determinada. Aparte de las mediciones estadísticas conocidas, también se puede determinar la suavidad de las curvas producidas por los datos discretos obtenidos. Esto se puede realizar por diferentes métodos, incluyendo técnicas que miden el potencial energético de las curvas para determinar su suavidad. Este tipo de criterio es utilizado por el método de energía de deflexión de vigas de Euler-Bernoulli.

6.5.1. Minimización de energía de curvas interpoladas por trazadores cúbicos

Una técnica muy útil para determinar la suavidad de datos discretos se basa en calcular la energía de deflexión de una curva interpolada de estos datos. Primeramente, se utilizan trazadores cúbicos para crear una curva continua que pase por cada dato discreto utilizandolos como marcadores. Estas curvas de trazadores cúbicos no solo son continuas en espacios bidimensionales, sino también poseen una propiedad de energía mínima. Luego, se asume que se posee una viga elástica de grosor infinitesimal que pasa a través de los marcadores siguiendo la curva interpolada. Con base en la teoría de vigas elásticas de Euler-Bernoulli, se sabe que esta viga se deflexiona siguiendo la curva que minimiza la energía de deflexión requerida entre dos marcadores [29]. Finalmente, esta energía de deflexión se puede calcular como:

$$W = \frac{1}{2} \int_{x_0}^{x_n} y''(x)^2 dx \quad (72)$$

Mientras la energía W sea más pequeña, la suavidad de la curva $y(x)$ es mayor. Un segmento recto presentaría $W = 0$. El comando $Y = \text{spline}(x,y,X)$ de MATLAB se puede utilizar para realizar la interpolación de trazadores cúbicos de una serie de datos, en donde (x, y) son los datos discretos y X es el vector de coordenadas abscisas correspondientes a todos los puntos a generar para la curva interpolada $y(x)$. Para derivar esta curva, se puede utilizar el comando $\text{diff}(Y)/h$ en donde h es la resolución de muestreo. Luego, para calcular la energía del cuadrado de la segunda derivada de la curva interpolada, se realiza integración numérica utilizando la siguiente fórmula:

$$\ell_k = x_k - x_{k-1} \quad (73)$$

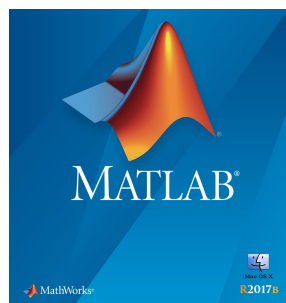
$$6W = \sum_{k=1}^n \ell_k (y_{k-1}''^2 + y_{k-1}'' y_k'' + y_k''^2) \quad (74)$$

6.6. Simuladores por medio de software

6.6.1. MATLAB R2017b

MATLAB es un software de cómputo numérico con su propio lenguaje de programación. El software funciona por medio de interpretes, por lo que este permite realizar operaciones vectoriales y de matrices facilitando la simulación de proyectos que involucran el movimiento de agentes dentro de un espacio determinado. Para algoritmos basados en movimiento de partículas es especialmente útil, ya que permite organizar las partículas vectorialmente y actualizar sus estados de manera rápida y eficiente.

Figura 21: Pantalla de inicio de MATLAB



6.6.2. WeBots R2019a

WeBots es un software *open source* que permite la simulación de robots móviles. Este programa fue desarrollado por el Dr. Oliver Michel del Instituto Federal Suizo de Tecnología EPFL para fines educativos. WeBots utiliza Open Dynamics Engine que permite simular colisiones y la dinámica de cuerpos rígidos. Esto permite simular el comportamiento físico de los robots para que la simulación sea lo más real posible. En este software, también se puede construir robots por medio de la definición geométrica y dinámica de sus componentes. Se pueden simular diferentes sensores y actuadores de catálogo. El programa permite utilizar controladores escritos en C, C++, Java, Python, MATLAB y ROS.

Figura 22: Pantalla de inicio de WeBots



Estableciendo el valor correcto de parámetros para MPSO

7.1. Simulación de partículas para ajuste de parámetros PSO

Primeramente, se buscó realizar la simulación del movimiento de partículas simples para observar el efecto de cada uno de los parámetros PSO. Para esto, se utilizó el software de MATLAB. Se buscó determinar cual era la combinación de parámetros indicada para lograr una extensa exploración del espacio de tarea y a la vez tener la mayor velocidad de convergencia posible hacia la meta establecida. Se utilizaron 200 partículas puntuales evaluadas en las funciones de prueba Sphere, Rosenbrock, Booth y Himmelblau. Se utilizó $\Delta t = 0.1s$ y las simulaciones se ejecutaron durante 12 segundos.

7.1.1. Características de simulación dependiendo el parámetro de inercia

En la sección 6.1.3 (Parámetros importantes del algoritmo), se expusieron los diferentes cálculos del parámetro de inercia ω que se pueden utilizar. Se realizaron simulaciones de parámetro de inercia constante, lineal decreciente, random, caótico y natural exponencial. Los parámetros de escalamiento c_1 y c_2 se fijaron en valores de 1.0 y 5.0, respectivamente. El parámetro de constricción φ se fijó en 1.0. En este caso, se utilizó la función de prueba Rosenbrock en donde las partículas convergen al mínimo $(x,y) = [1, 1]$. En las siguientes figuras, se observa la evolución en el tiempo de la localización y dispersión de las partículas calculadas con la media y desviación estándar de las posiciones (x,y) de todas las partículas del enjambre. Esto se realizó para evaluar el rango de exploración del enjambre, así como su velocidad de convergencia hacia la meta correcta dependiendo el tipo de inercia utilizado.

Figura 23: Dispersión de partículas utilizando parámetro de inercia caótica

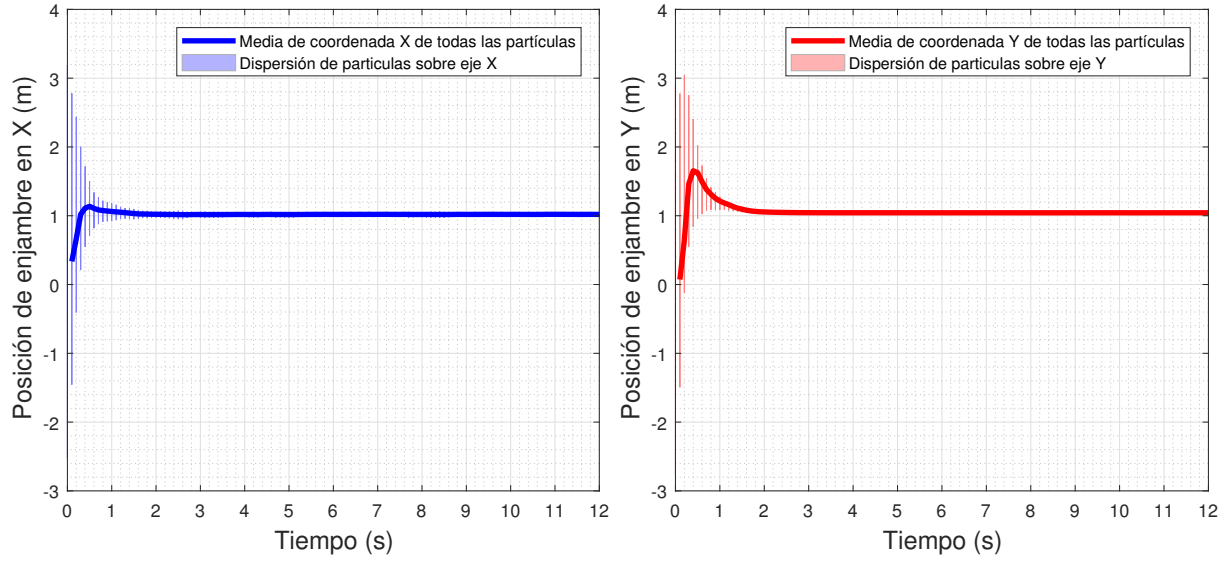


Figura 24: Dispersión de partículas utilizando parámetro de inercia random

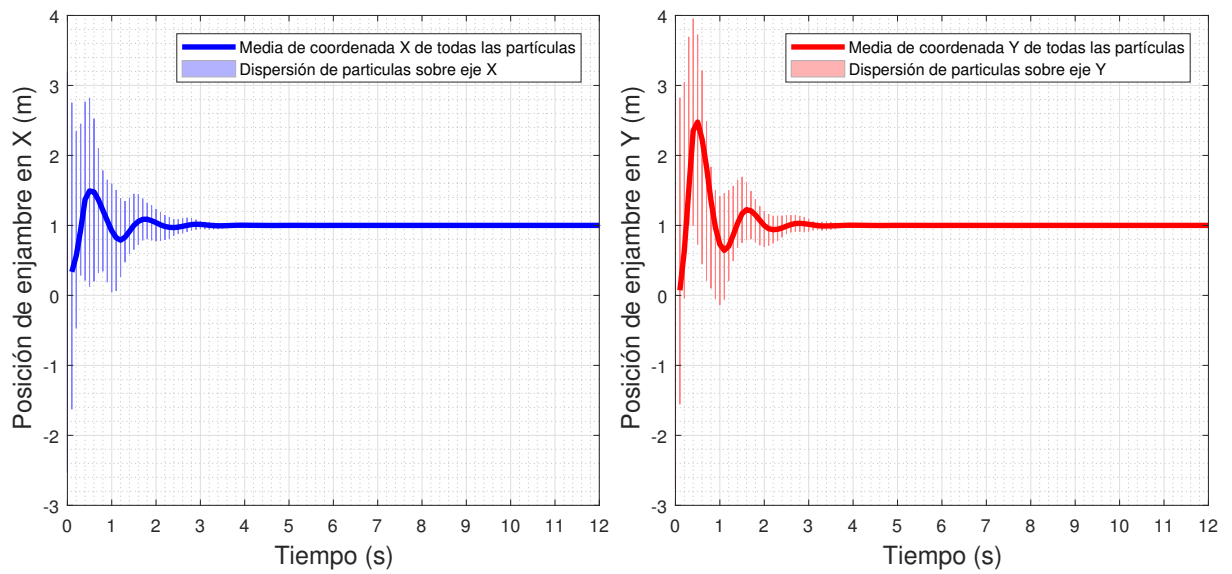


Figura 25: Dispersión de partículas utilizando parámetro de inercia constante

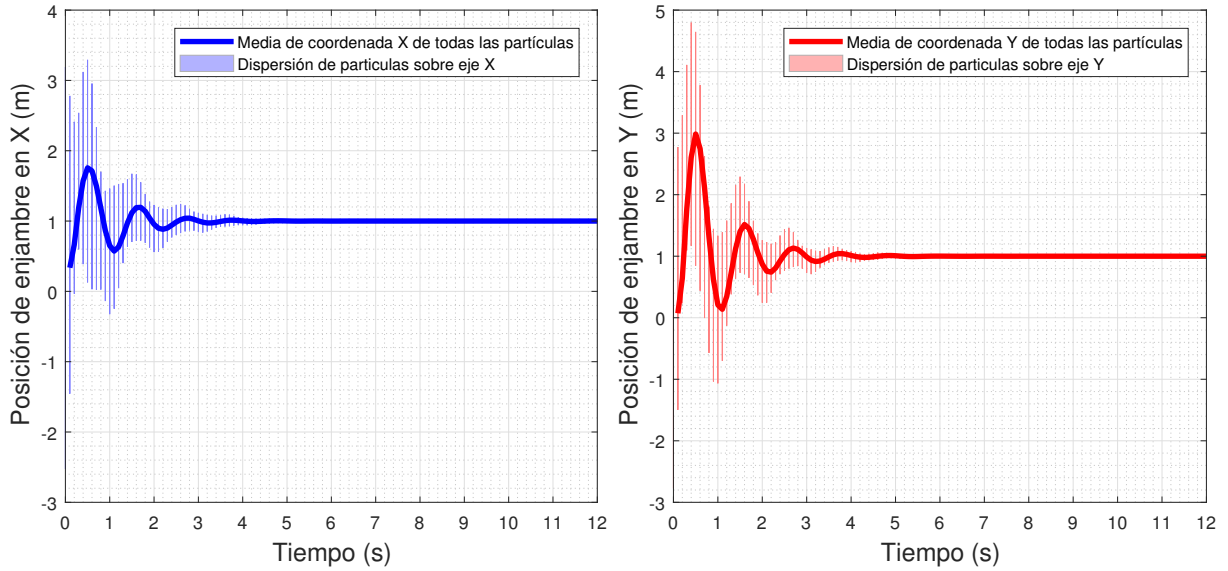


Figura 26: Dispersión de partículas utilizando parámetro de inercia lineal decreciente

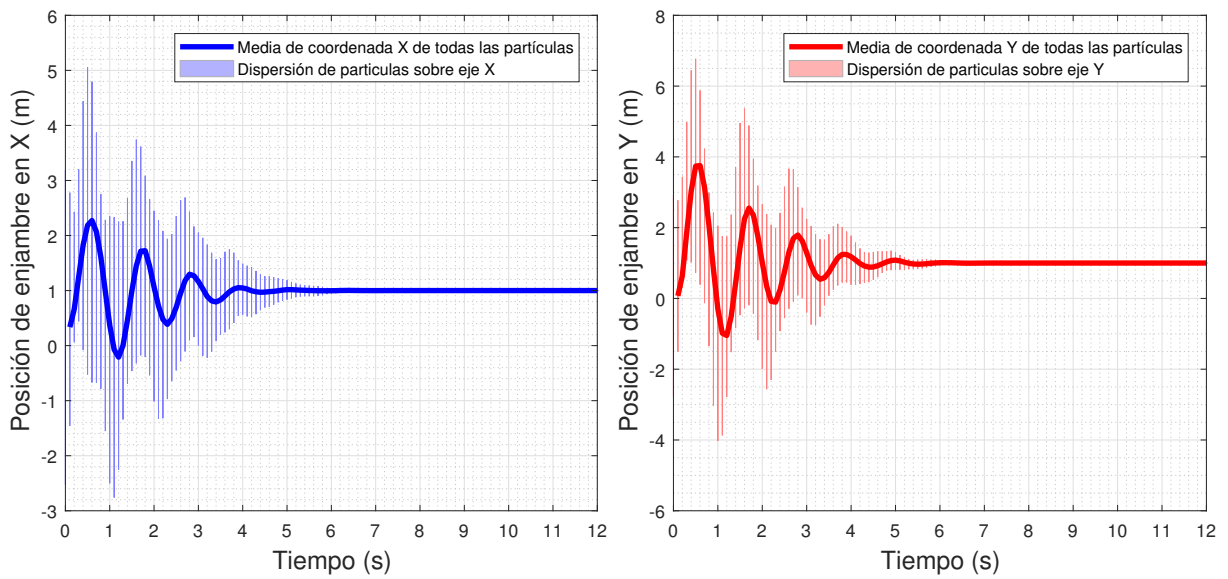
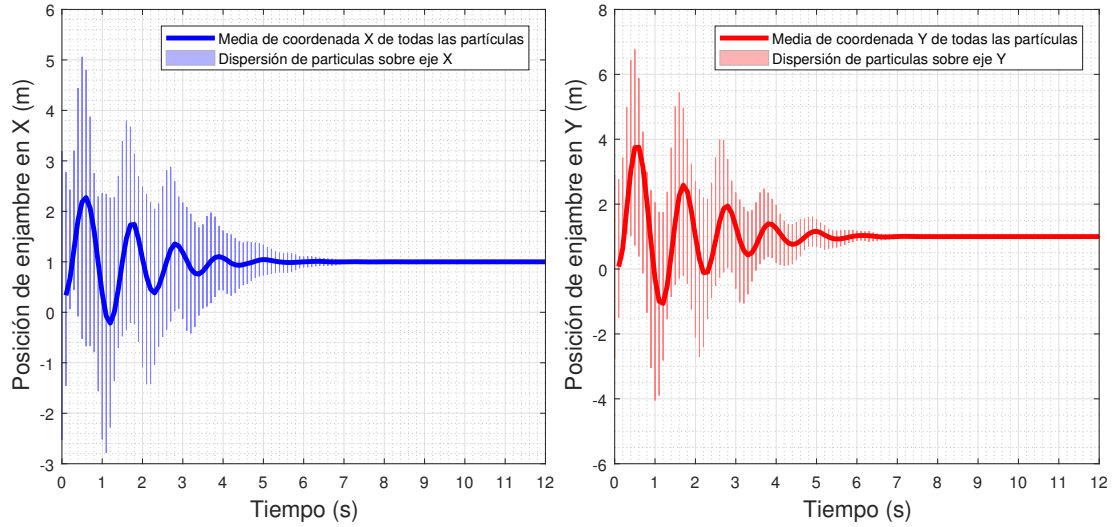


Figura 27: Dispersión de partículas utilizando parámetro de inercia natural exponencial



En las figuras anteriores, se puede observar que utilizar parámetros de inercia caótico y random (Figuras 23 y 24) causó que el enjambre tuviera alta velocidad de convergencia (<3 segundos). Sin embargo, la exploración del espacio fue mínima y se observó un brusco decrecimiento de la dispersión de enjambre por la falta de movimiento alrededor del espacio. El uso de un parámetro de inercia constante (Figura 25) resultó en más exploración del espacio pero esto no aumentó considerablemente. Al utilizar una inercia lineal decreciente y natural exponencial (Figuras 26 y 27), se observó una convergencia más lenta, pero la exploración del espacio fue considerablemente mayor, evidenciado por la mayor dispersión de las partículas. Esto permitió al enjambre evitar converger a mínimos locales de la función evaluada. El parámetro de inercia natural exponencial presentó la ventaja de obtener un tiempo similar de convergencia que el lineal decreciente, pero obtuvo un ligero aumento en la exploración del espacio. **Por lo tanto, se elige el parámetro de inercia natural exponencial para el MPSO¹**

Cuadro 3: Características de convergencia al variar parámetro de inercia ω

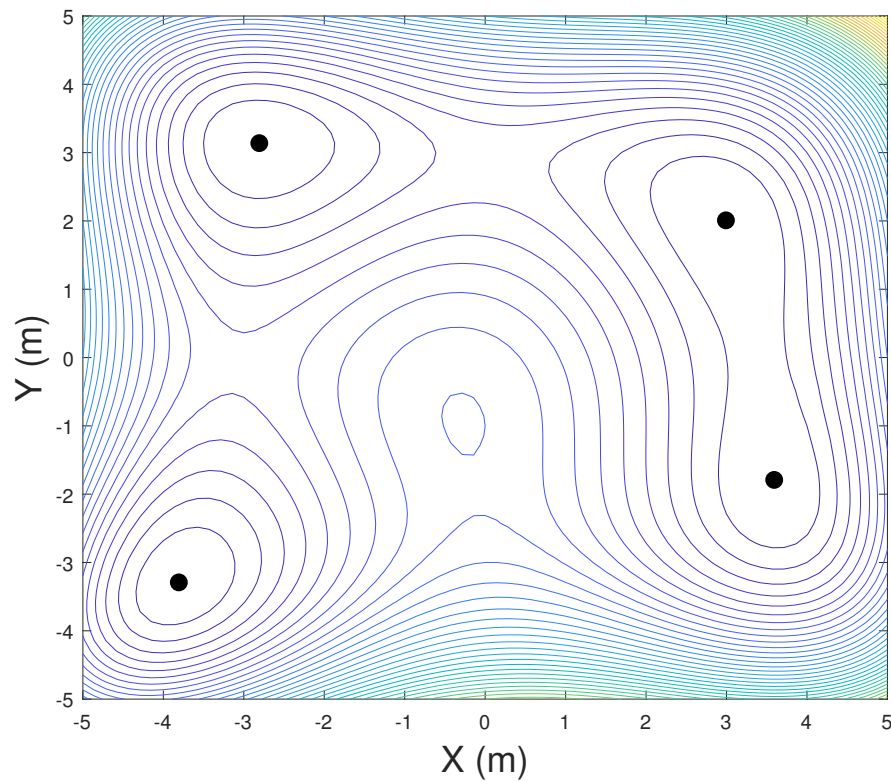
Parámetro de inercia	T. de convergencia (s)	Desv. estándar x^1	Desv. estándar y^1
Caótica	0.80	0.15	0.15
Random	2.20	0.75	0.75
Constante	2.80	0.80	1.40
Lineal decreciente	5.00	1.80	2.50
Natural exponencial	5.50	1.85	2.60

¹La desviación estándar de la tabla se mide en el primer segundo de la simulación para no tomar en cuenta la desviación propia de las posiciones iniciales y para tener un índice adecuado de la dispersión inicial

7.1.2. Características de simulación dependiendo los parámetros de escalamiento

En la sección 6.1.3, se expusieron los parámetros de escalamiento c_1 y c_2 para los factores cognitivo y social del algoritmo PSO. Una condición $c_1 > c_2$ da mayor peso a la explotación de mínimos locales de cada partícula (mayor exploración del espacio). Una condición $c_1 < c_2$ da mayor peso a la explotación del mínimo absoluto encontrado por el enjambre (mayor velocidad de convergencia). Se utilizó una inercia ω natural exponencial y el parámetro de constricción φ se fijó en 1.0. Se utilizó la función de prueba Himmelblau que posee cuatro posibles mínimos de convergencia para observar posibles casos de fragmentación de enjambre.

Figura 28: Contornos de nivel de función de costo Himmelblau con sus cuatro mínimos



Cuadro 4: Identificación de todos los mínimos de función de costo Himmelblau

No. de mínimo absoluto	Coordenada X (m)	Coordenada Y (m)
1	3.00	2.00
2	-2.80	3.13
3	-3.80	-3.30
4	3.60	-1.80

Figura 29: Dispersión de partículas utilizando $c_1 = 2$ y $c_2 = 2$

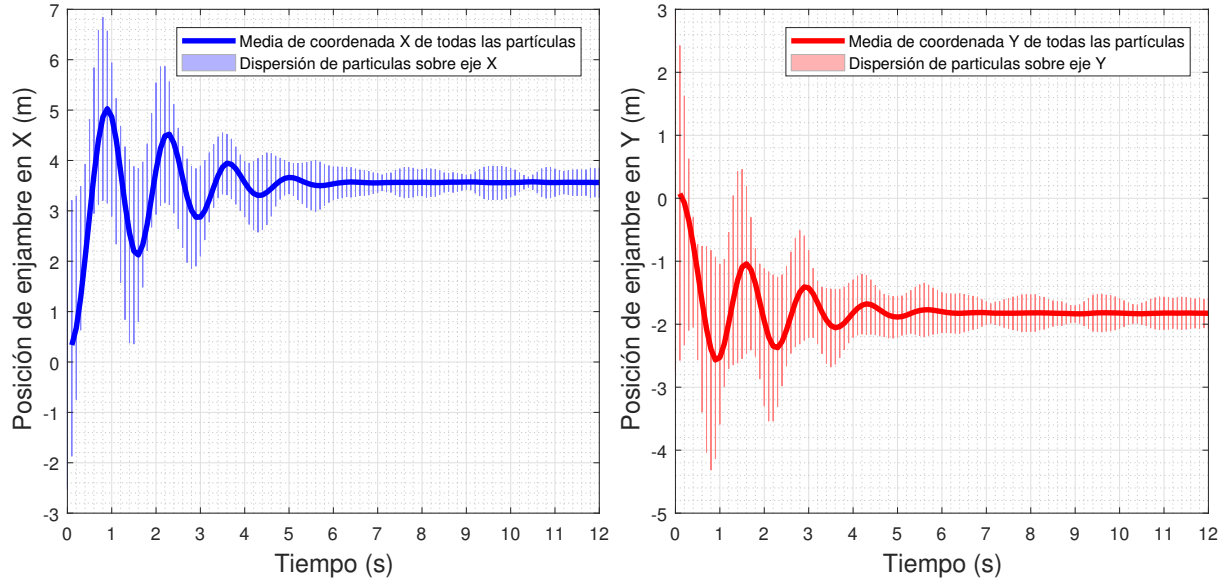


Figura 30: Dispersión de partículas utilizando $c_1 = 1$ y $c_2 = 5$

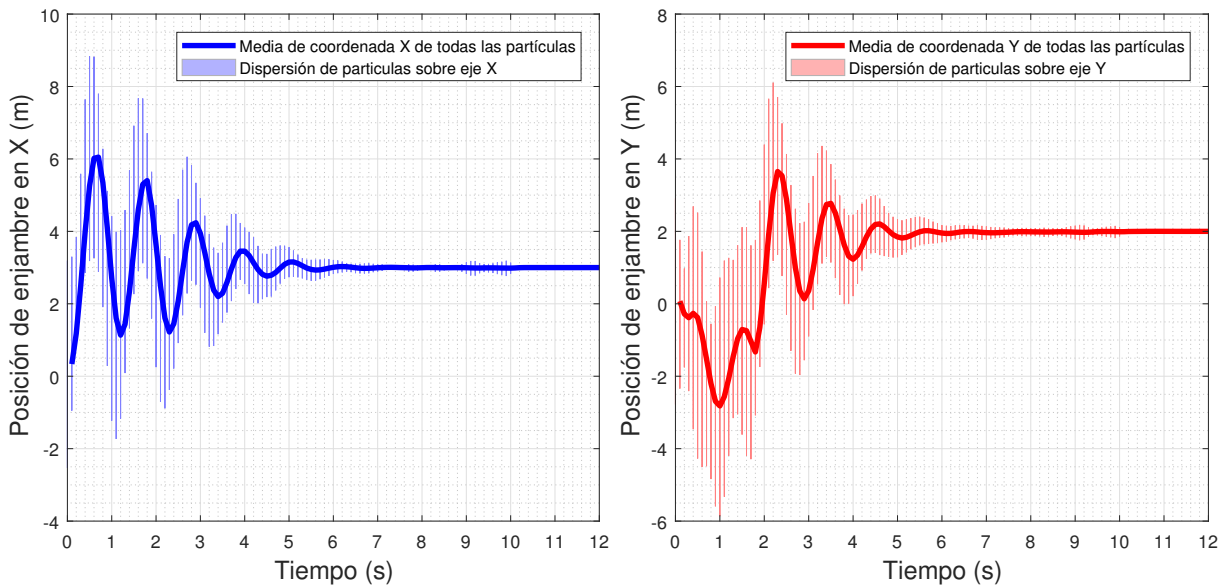


Figura 31: Dispersión de partículas utilizando $c_1 = 5$ y $c_2 = 1$

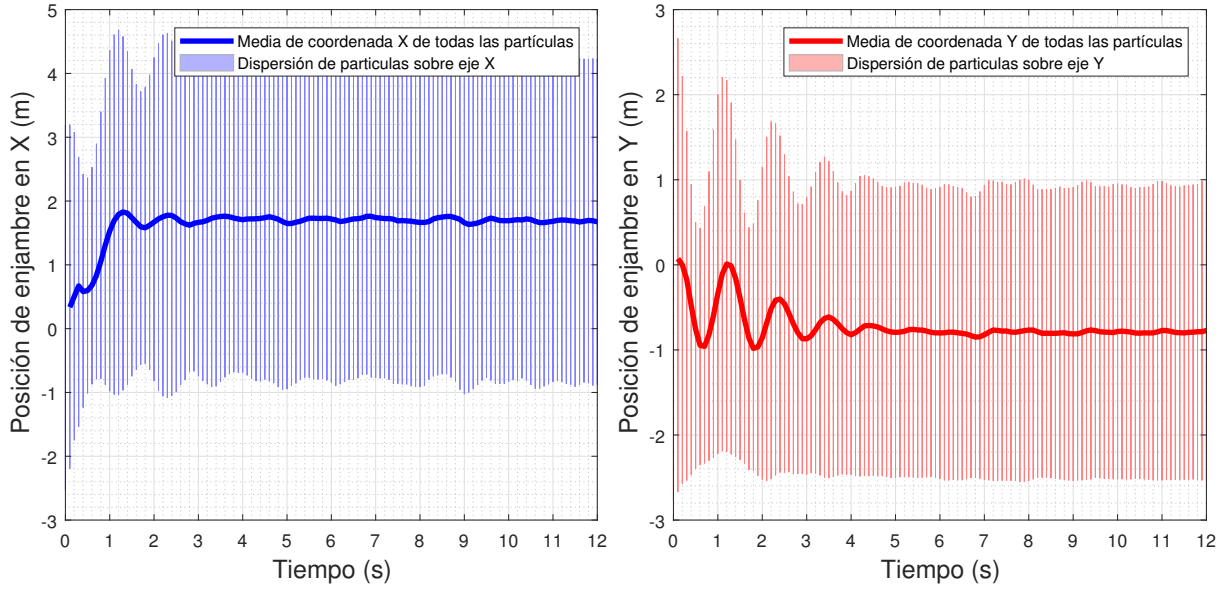
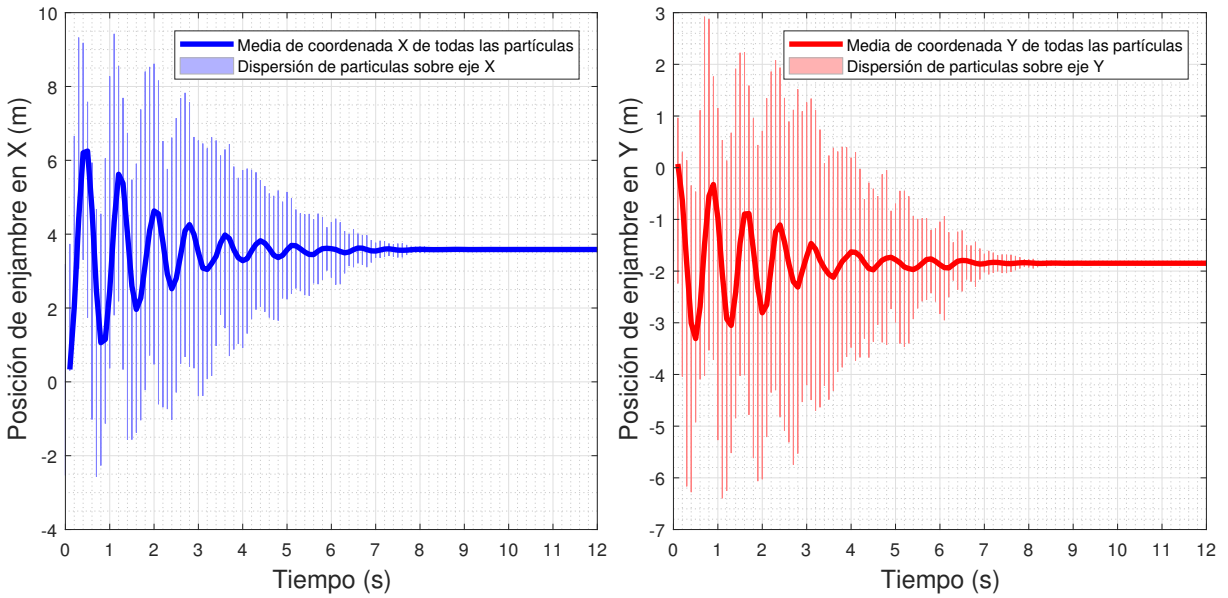


Figura 32: Dispersión de partículas utilizando $c_1 = 2$ y $c_2 = 10$



En las figuras anteriores, se puede observar el comportamiento que presentaron las partículas al moverse por el espacio dependiendo el factor (cognitivo o social) del PSO que poseía mayor ponderación. Al utilizar una ponderación igual para ambos factores (Figura 29), se obtuvo una velocidad de convergencia más lenta que al utilizar una mayor ponderación para el factor social (Figura 30). Esto se debió a que una mayor ponderación al factor social significó una mayor explotación de un punto específico. Por lo tanto, se aceleró la convergencia. En el caso de configurar una mayor ponderación al factor cognitivo (Figura 31), se obtuvo mayor exploración del espacio demostrado por la mayor dispersión observada. Sin embargo, se presentó convergencia errónea a mínimos locales y consecuente fragmentación del enjambre, como se muestra en dicha gráfica. En la Figura 32, se volvió a tomar el caso de mayor ponderación al factor social como en la Figura 30. Sin embargo, la magnitud de los parámetros se fijó al doble. Se observa que, en este caso, la dispersión del enjambre y la consecuente exploración del espacio fue mayor, y el tiempo de convergencia no aumentó considerablemente. **Por lo tanto, para el MPSO, se eligen parámetros de escalamiento c_1 y c_2 ajustados a 2 y 10, respectivamente**².

Cuadro 5: Características de convergencia al variar parámetros de escalamiento c_1 y c_2

Parámetro c_1	Parámetro c_2	Tiempo (s)	Desv. estándar x^2	Desv. estándar y^2
2	2	7.0	1.80	1.80
1	5	6.0	2.80	2.80
5	1	—	1.70	2.50
2	10	7.0	3.50	3.00

7.1.3. Características de simulación dependiendo el parámetro de constricción

En la sección 6.1.3 (Parámetros importantes del algoritmo), se expuso el parámetro de constricción φ que limita el tamaño del paso que puede dar cada partícula al actualizar su posición. Si el parámetro es mucho mayor a 1.0, puede presentarse divergencia en el algoritmo. Y si es mucho menor, la convergencia puede ser muy lenta. Se evaluaron valores de 0.5, 1.0, 1.1 y 1.3 para este parámetro. También se utilizó la ecuación (3) planteada para calcular este parámetro. Se fijó el cálculo del parámetro de inercia ω como natural exponencial. Los parámetros de escalamiento c_1 y c_2 se fijaron en valores de 1.0 y 5.0, respectivamente. En este caso, se utilizó la función de prueba Booth en donde las partículas convergen al mínimo $(x,y) = [1, 3]$.

²La desviación estándar se mide en el primer segundo de la simulación para no tomar en cuenta la desviación propia de las posiciones iniciales y para tener un índice adecuado de la dispersión inicial

Figura 33: Dispersión de partículas utilizando ecuación (3) de φ

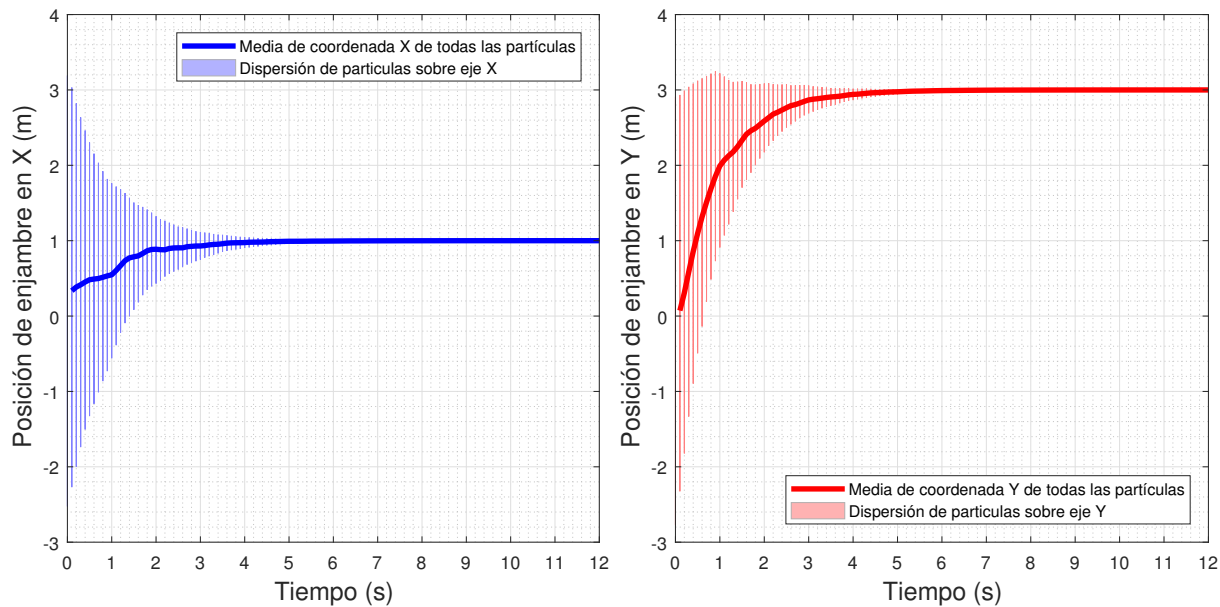


Figura 34: Dispersión de partículas utilizando $\varphi = 0.5$

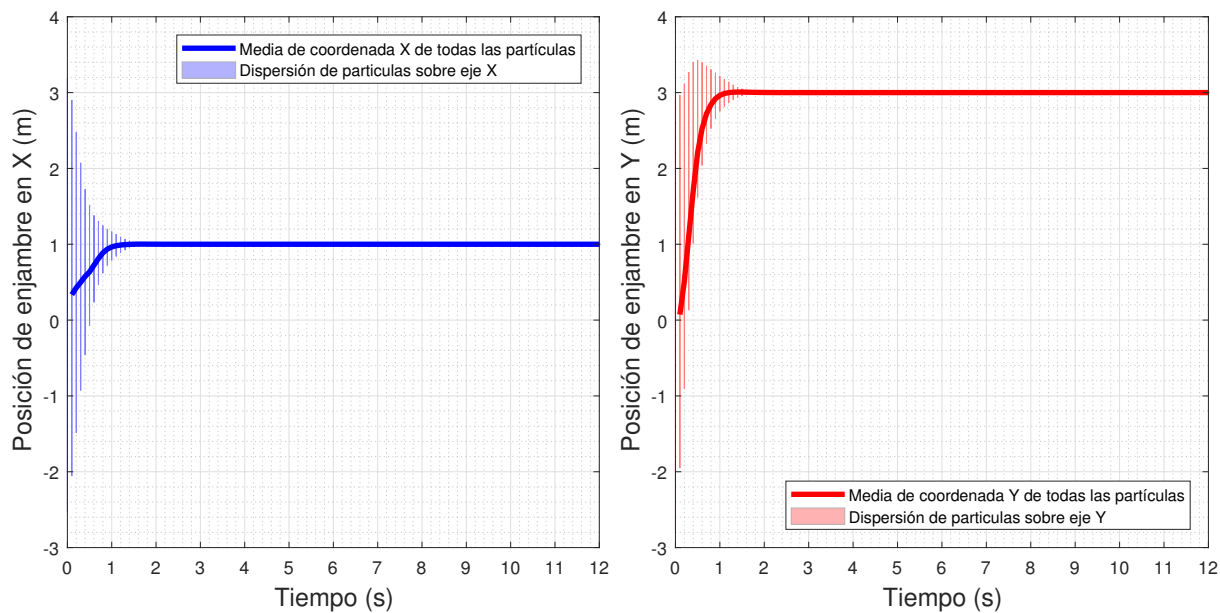


Figura 35: Dispersión de partículas utilizando $\varphi = 1.0$

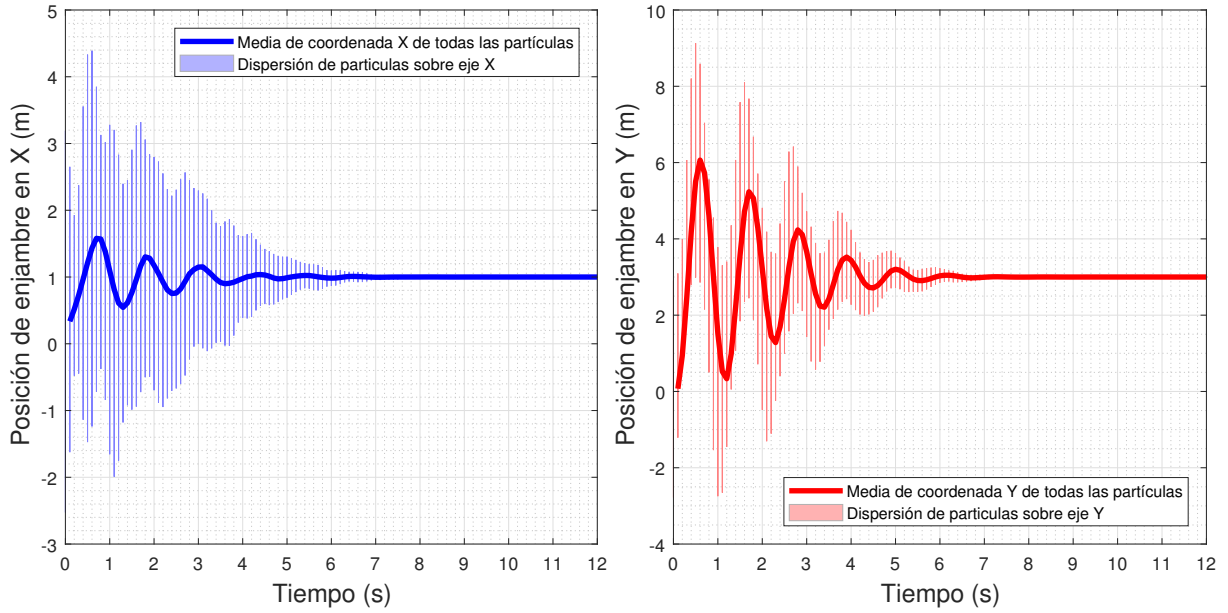


Figura 36: Dispersión de partículas utilizando $\varphi = 1.1$

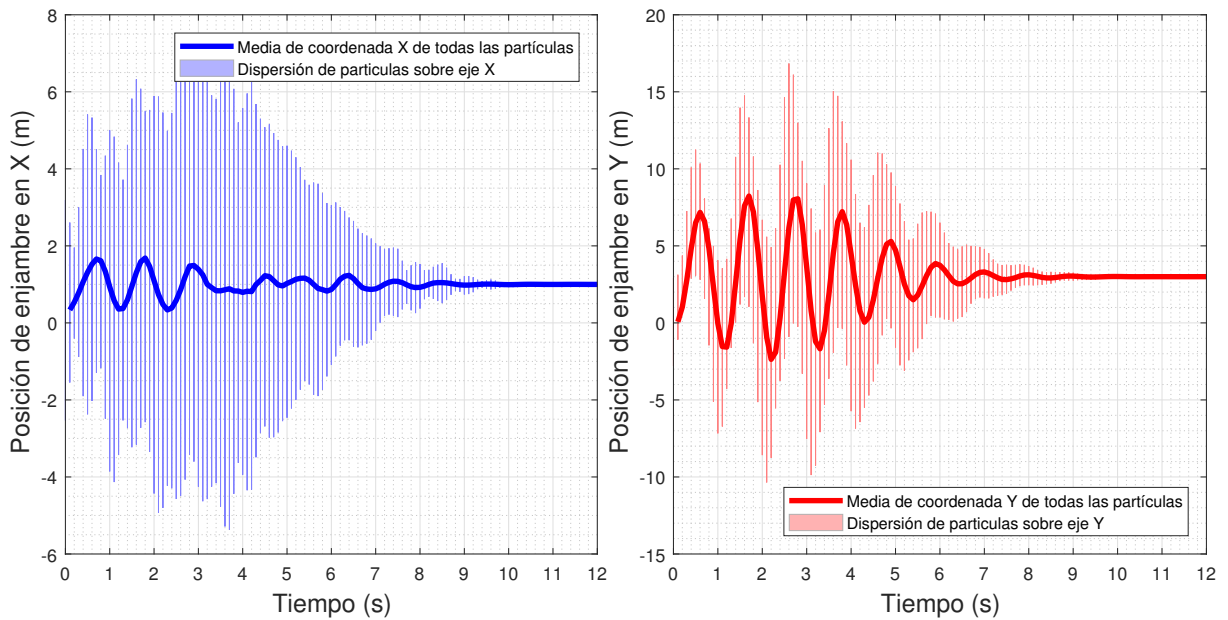
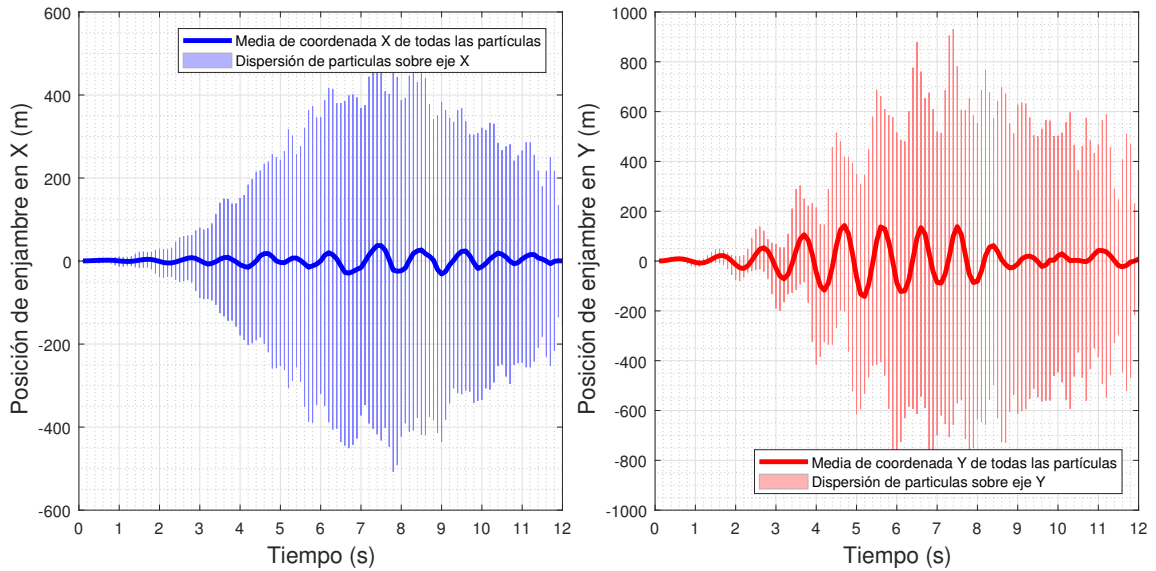


Figura 37: Dispersión de partículas utilizando $\varphi = 1.3$



En la Figura 33, se puede observar que utilizar la ecuación establecida para calcular el parámetro φ causó que la convergencia fuera ligeramente más lenta que utilizando $\varphi = 0.5$, como se muestra en la Figura 34. En ambos casos, existió poca exploración del espacio evidenciado por el brusco decrecimiento de la dispersión del enjambre. En la Figura 35, se puede observar que utilizar $\varphi = 1.0$ aumentó considerablemente la exploración de espacio y se tiene buena velocidad de convergencia. En la Figura 36, se puede observar que utilizar $\varphi = 1.1$ causó un crecimiento notable en la dispersión de enjambre, por lo que la convergencia fue más lenta. Por último, en la Figura 37, se observa que utilizar $\varphi = 1.3$ causó la divergencia del algoritmo por una extrema dispersión del enjambre. **Por lo tanto, se elige un parámetro de constricción $\varphi = 1.0$ para el MPSO³**

Cuadro 6: Características de convergencia al variar parámetro de constricción φ

Constricción (m)	T. de convergencia (s)	Desv. estándar x^3	Desv. estándar y^3
Eq. (3)	3.00	1.20	1.20
0.5	1.10	0.20	0.20
1.0	5.50	2.00	2.70
1.1	8.20	6.20	8.40
1.3	—	480	740

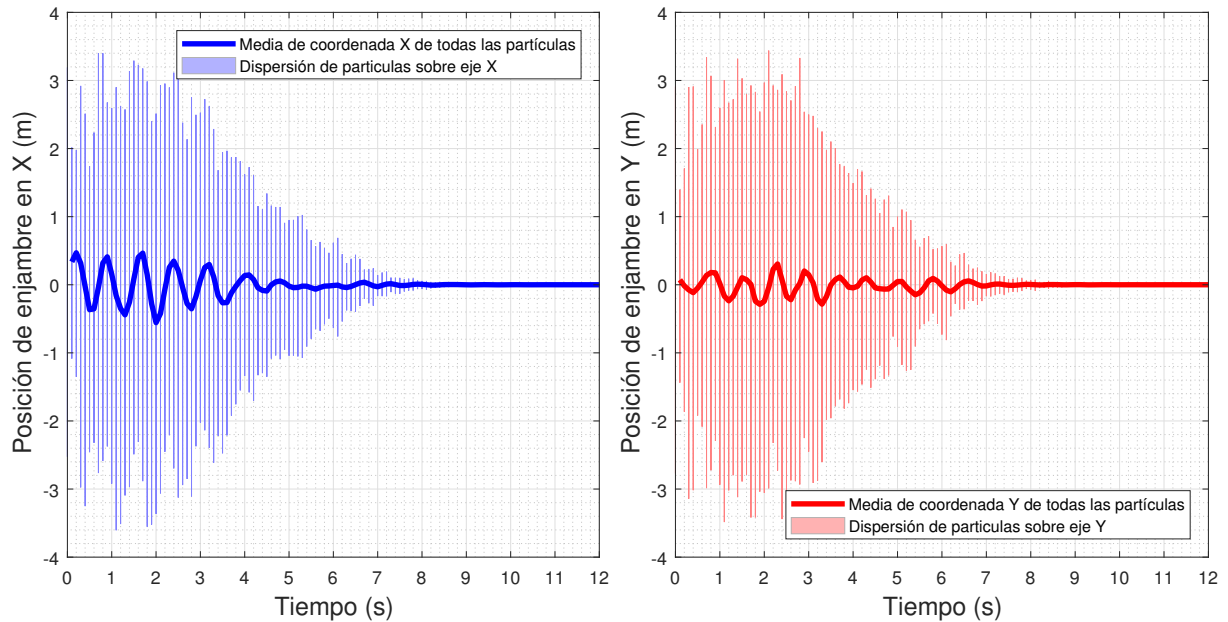
³La desviación estándar se mide en el primer segundo de la simulación para no tomar en cuenta la desviación propia de las posiciones iniciales y para tener un índice adecuado de la dispersión inicial (No aplica para Figura 36 y 37 en donde se toma dato en segundo 4 y 8, respectivamente)

7.2. Selección final del parámetros PSO

- Se eligió un parámetro de inercia (ω) natural exponencial calculado con la ecuación (9).
- Se eligieron parámetros de escalamiento $c_1 = 2$ y $c_2 = 10$
- Se eligió un parámetro de constricción $\varphi = 1.0$

En la siguiente figura, se observa que, con los parámetros elegidos, el enjambre de partículas logró tener un rango amplio de exploración del espacio y una velocidad adecuada de convergencia. El enjambre de 200 partículas logró converger al mínimo de la función de prueba Sphere localizado en $(x,y) = [0, 0]$ en 5 segundos con considerable dispersión a lo largo del espacio de búsqueda de $10 \times 10 \text{m}$.

Figura 38: Movimiento resultante de partícula con parámetros elegidos (Función de prueba Sphere)



Estructuración del algoritmo MPSO en simulación de Webots

8.1. Programación de algoritmo MPSO para simular convergencia en WeBots

Luego de haber determinado cuales parámetros MPSO eran necesarios para obtener el comportamiento deseado del enjambre, se continuó con la implementación del algoritmo a robots con dimensiones físicas. Para este propósito, se utilizó el software WeBots en donde se programó el controlador de los robots en lenguaje C. En este capítulo, se expone el diseño del algoritmo de acople del MPSO de partículas a los robots, así como la estructuración de la simulación en WeBots.

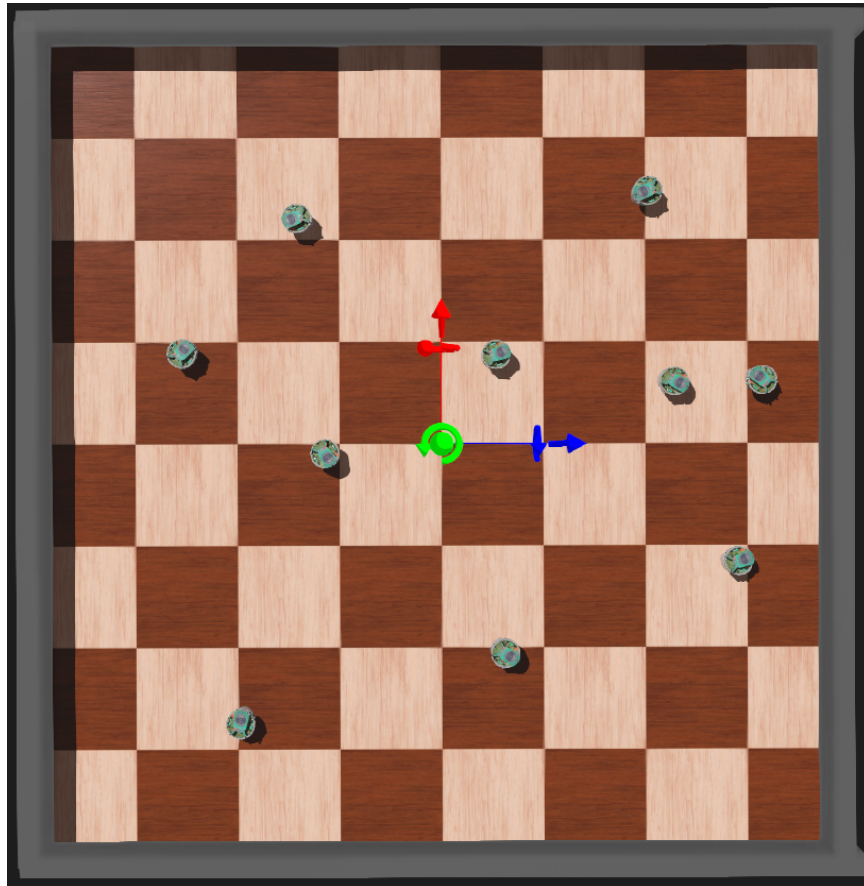
8.1.1. Elementos del mundo simulado en WeBots

Para comenzar la simulación, se construyó un ambiente de pruebas para el movimiento de los robots. Se colocó un piso de 2x2 metros para simular el tamaño real de una mesa de prueba que se podría construir en el mundo real. A este piso, se le colocaron paredes en los bordes para delimitar el movimiento de los robots. Se configuró una gravedad de 9.81 m/s^2 y se habilitaron las diferentes propiedades físicas del mundo para que los robots reaccionaran realísticamente ante colisiones y aceleraciones. Para los robots, se utilizó el modelo E-Puck ya incluido en el catálogo por defecto de WeBots. Este modelo posee las medidas reales de los robots E-Puck del Laboratorio ASL en Suiza. Se colocaron 10 E-Puck a lo largo del piso de la simulación. Se colocó una luz sobre la simulación y se habilitó la propiedad de rendering de sombras en la simulación para que se viera lo más real posible.

El origen del plano coordenado de la simulación se colocó en el centro del piso 2x2m de la simulación. El eje Y (de color verde en la simulación) apunta hacia el ojo del observador de

la simulación. El eje X (de color rojo en la simulación) se trató como el eje de las ordenadas en las mediciones de posición y este apunta al Norte del mundo simulado. El eje Z (de color azul en la simulación) se trató como el eje de las abscisas y apunta al Este del mundo simulado.

Figura 39: Mundo simulado en WeBots con E-Pucks implementados



8.1.2. Diseño del algoritmo de controlador MPSO

Para llevar a cabo la implementación del algoritmo MPSO a los robots de la simulación de WeBots, se estructuró el código de controlador de tal manera que lograra ejecutar acciones como determinar la posición y orientación actual del robot, determinar el costo de posición actual evaluada en la función de costo PSO (Fitness Function), actualizar la mejor posición local encontrada (Best Local) y la mejor posición global encontrada (Best Global), calcular las nuevas velocidades y posiciones PSO, transformar cinemáticamente las velocidades PSO a velocidades de los actuadores, y limitar dichas velocidades para evitar dañar los actuadores por sobrepasar márgenes de saturación o por muchos cambios bruscos en estas velocidades.

En las siguientes páginas, se puede observar el diagrama de flujo que contiene los pasos que ejecuta el algoritmo para realizar estas acciones:

Figura 40: Diagrama de flujo de configuración de controlador MPSO

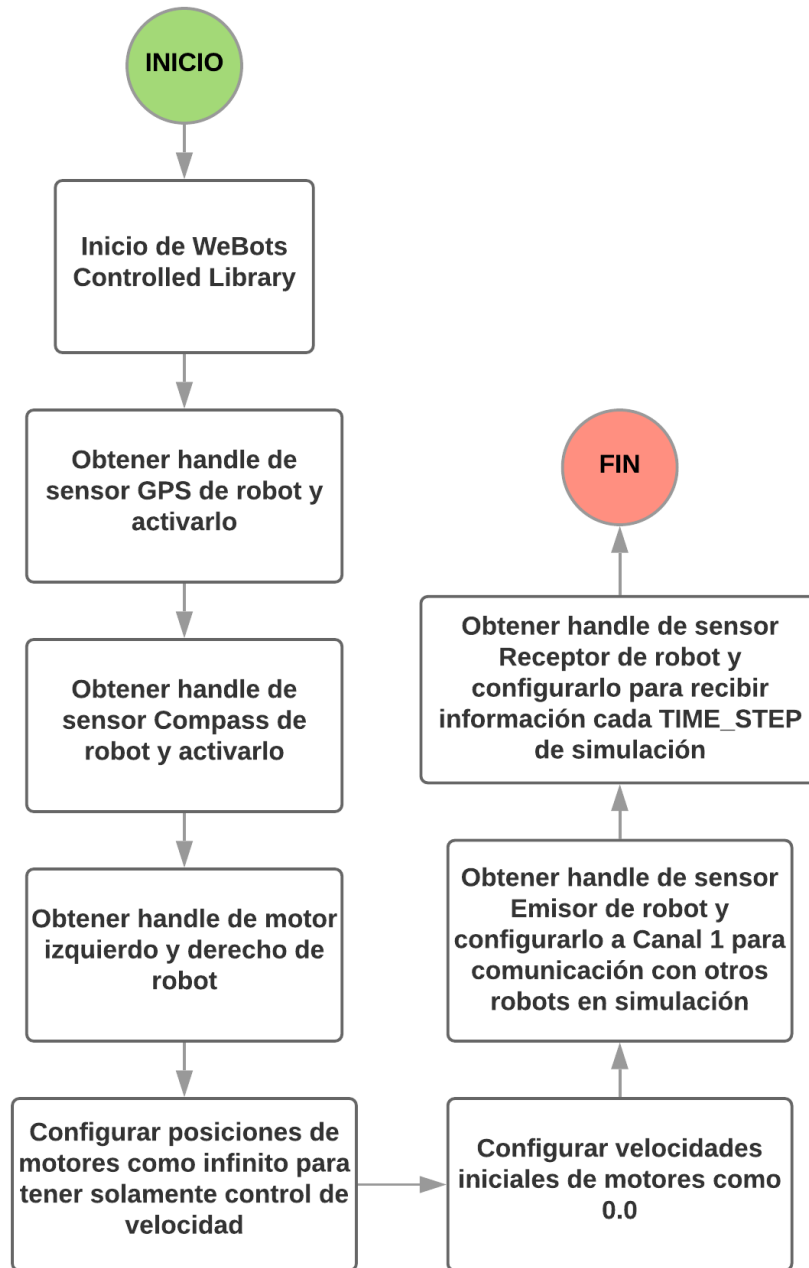


Figura 41: Diagrama de flujo de controlador MPSO (Pt.1)

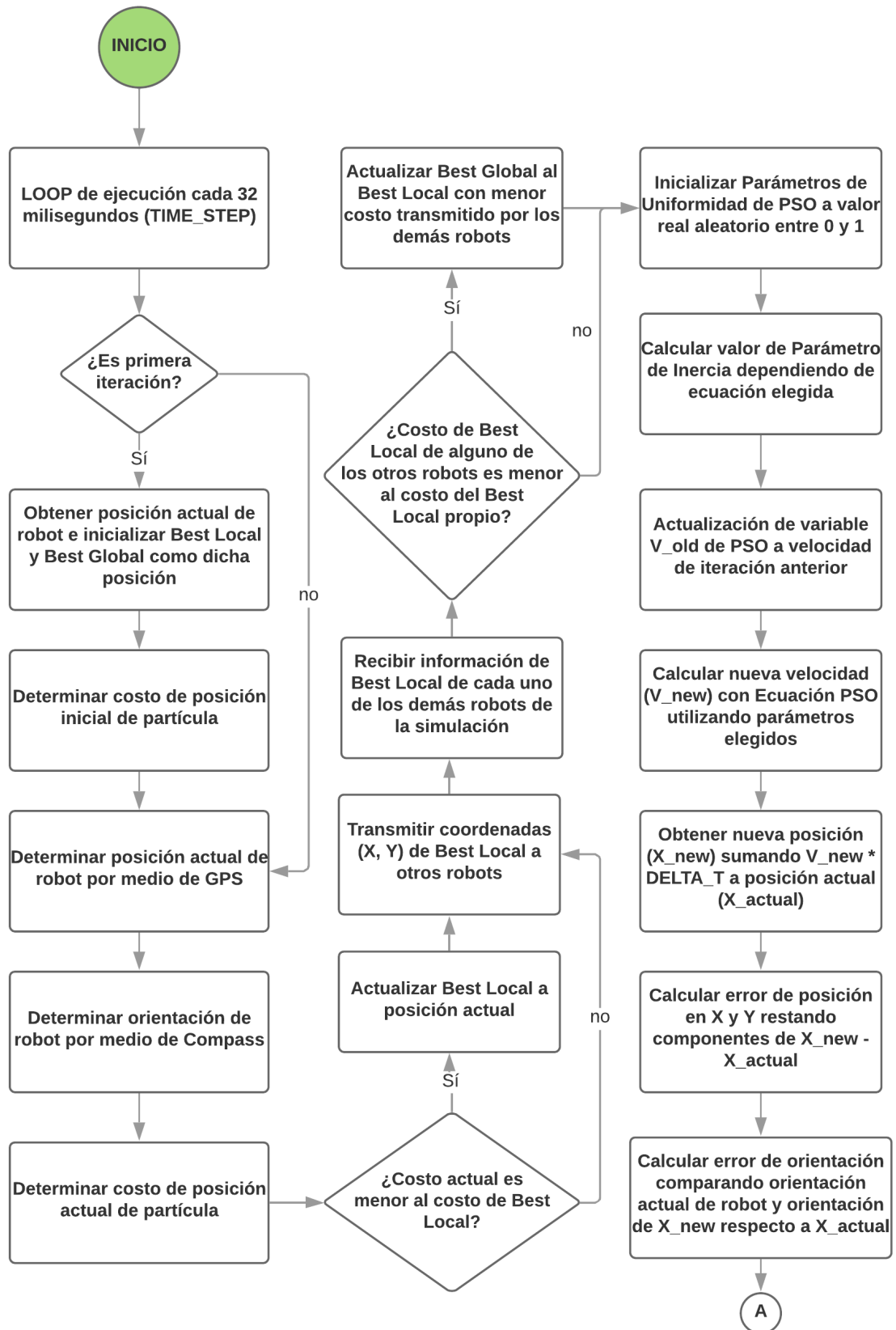
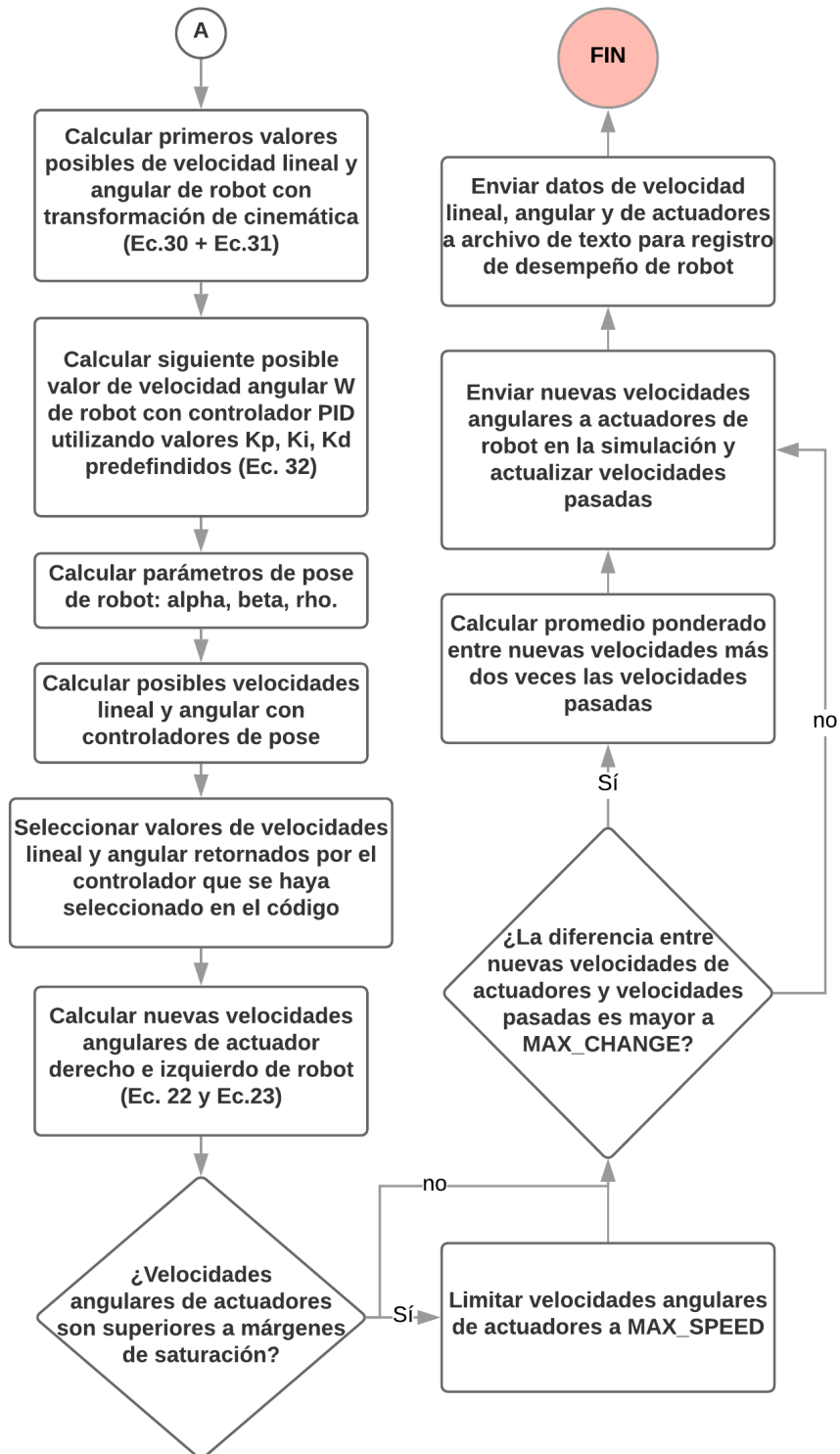


Figura 42: Diagrama de flujo de controlador MPSO (Pt.2)



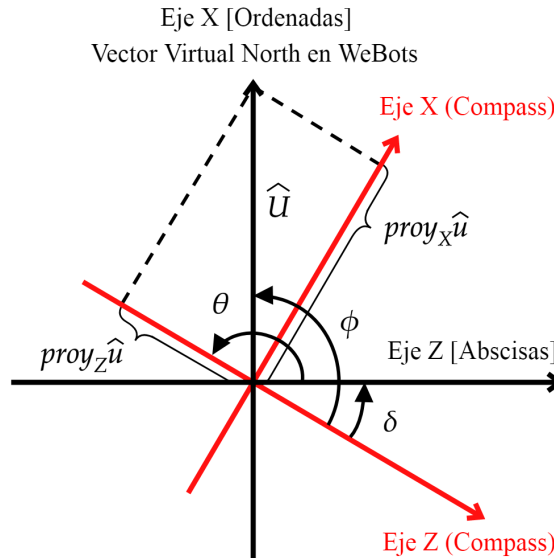
8.1.3. Variables de controlador en WeBots

- **TIME_STEP** (T_S): esta configura el período de ejecución del controlador de los robots en unidades de milisegundos. Para esta investigación, se utilizó un T_S de 32 milisegundos.
- **PSO_SCALER** (η): es un factor de escalamiento de la velocidad PSO para poder sumarla adecuadamente a la posición PSO de un robot. Este factor determina la longitud de los vectores directores del PSO para la planeación de movimiento del robot. Más detalle sobre este parámetro se puede encontrar en el siguiente capítulo.
- **MAX_SPEED** (Φ_{max}): esta variable configura el límite de velocidad angular máxima que se le puede enviar a los robots de simulación. Los motores de los E-Pucks pueden alcanzar un máximo de 6.28 rad/s. Por lo tanto, Φ_{max} se configura a esta cantidad.
- **MAX_CHANGE** ($\Delta\Phi_{max}$): esta variable configura el límite de cambio máximo permisible de velocidad angular en los motores de los robots de la simulación. Se limitó el cambio máximo permisible de velocidad angular a 1 rad/s para evitar hard stops o variaciones rápidas en velocidad que podrían dañar los motores.

8.1.4. Cálculo de orientación de robot

Para determinar la orientación de los robots, se utilizó el nodo (u objeto) **Compass** que llevaba incluido el modelo E-Puck de la librería de WeBots. Para extraer los valores de orientación de dicho nodo se utilizó la función `wb_compass_get_values()` que retornaba un vector de las proyecciones del Eje X de la simulación (*northDirection vector* en WeBots) sobre los ejes XZ del nodo *Compass*:

Figura 43: Ejes de simulación vs. ejes de nodo *Compass*



Vector de proyecciones:

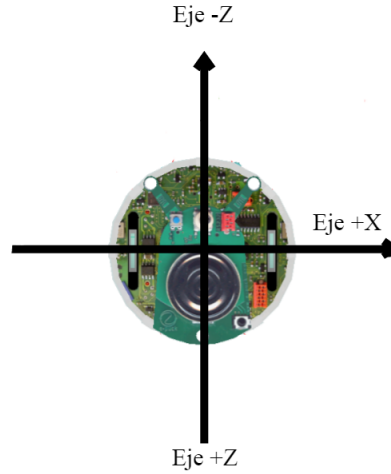
$$\hat{v} = [\text{proy}_X\hat{u} \quad \text{proy}_Y\hat{u} \quad \text{proy}_Z\hat{u}], \quad (75)$$

Ya con este vector de proyecciones, se pudo calcular el ángulo ϕ entre el eje +Z del nodo *Compass* del robot y el eje vertical (ordenadas) de la simulación (como se observa en la Figura 43) utilizando la ecuación:

$$\phi = \text{atan2}(\text{proy}_X\hat{u}, \text{proy}_Z\hat{u}) \cdot \frac{180}{\pi}, \quad (76)$$

Con esta última ecuación se logró determinar la orientación del vector norte de la simulación respecto al marco referencial del nodo *Compass*. Sin embargo, como se observa en la Figura 44, el eje frontal del robot está alineado al eje -Z del nodo *Compass*.

Figura 44: Ejes de E-Puck y nodo *Compass*



Por lo tanto, para obtener el ángulo verdadero θ del eje direccional -Z del robot respecto a la horizontal de la simulación (como se observa en la Figura 43), se realizaron los cálculos siguientes:

$$\delta = \phi - 90^\circ \quad (77)$$

$$\theta = 180^\circ - \delta \quad (78)$$

$$\theta = 270^\circ - \phi \quad (79)$$

Ya con la orientación θ determinada de los robots, se procedió a determinar la forma de calcular el error de orientación del robot respecto a la orientación que debe tener para apuntar hacia el punto objetivo o meta. Dicho calculo se presenta más adelante en esta sección.

8.1.5. Cálculo de posición de robot

Para determinar la posición de los robots, se utilizó el nodo (u objeto) **GPS** que llevaba incluido el modelo E-Puck de la librería de WeBots. Para extraer los valores de posición de dicho nodo, se utilizó la función `wb_gps_get_values()` que retornaba un vector de las posiciones del nodo GPS con respecto a los ejes de la simulación:

$$\hat{p} = [X \quad Y \quad Z], \quad (80)$$

Como se mencionó en la sección 8.1.1. (Elementos del mundo simulado en WeBots), el eje X de la simulación se tomó como el eje de las ordenadas y el eje Z como el eje de las abscisas. Por lo tanto, al extraer la posición horizontal del robot se tomó el tercer componente del vector de posición retornado por el GPS, el cual indicaba la posición respecto al eje Z de la simulación. Y para la posición vertical, se tomó el primer componente que indicaba la posición respecto al eje X de la simulación.

8.1.6. Cálculo de errores de posición y orientación de robots

Cálculo de error de posición de robot:

Este cálculo fue sencillo y se redujo a la resta entre la coordenada X del robot y la coordenada X del punto objetivo, al igual que con las coordenadas sobre Y¹. El vector de errores estaba dado por:

$$\tilde{h} = \begin{bmatrix} X_g - X_{actual} \\ Y_g - Y_{actual} \end{bmatrix} \quad (81)$$

Este vector es el que se ingresó a la ecuación (31) para calcular las velocidades de punto a ingresar a la ecuación (30) con el objetivo obtener las velocidades lineales y angulares de los robots.

Cálculo de error de orientación de robot:

Ya teniendo establecida la forma de calcular la orientación de los robots dentro de la simulación, se procedió a determinar la forma de calcular el error de orientación e_o respecto a la orientación del punto objetivo. Finalmente, se utilizaron las siguientes ecuaciones para dicho cálculo:

$$\theta_g = \text{atan2}(Y_g - Y_{actual}, X_g - X_{actual}) \quad (82)$$

$$e_o = \text{atan2}(\sin(\theta_g - (\theta_o \cdot \frac{\pi}{180})), \cos(\theta_g - (\theta_o \cdot \frac{\pi}{180}))); \quad (83)$$

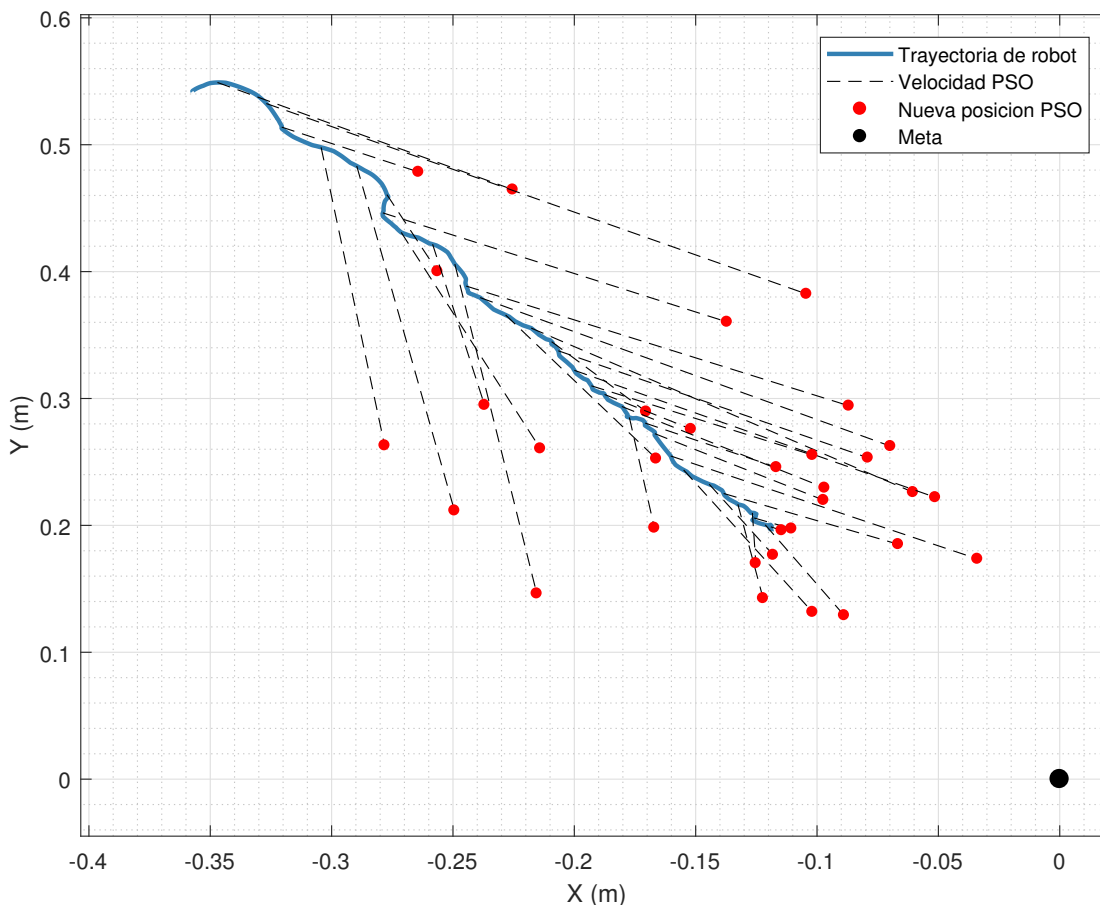
¹Recordar que coordenadas (X,Y) de los robots corresponden a coordenadas (Z,X) en mundo simulado.

Planeación de trayectorias de robots por medio de PSO

El algoritmo PSO se caracteriza por estar diseñado para el movimiento de partículas sin masa ni restricciones físicas de movimiento alrededor del espacio de búsqueda. Este algoritmo genera trayectorias vectoriales quebradizas que son adecuadas a nivel de partícula pero resultan inadaptables directamente a sistemas no holonómicos y con restricciones físicas como los robots diferenciales. Por esta razón, en esta investigación se propuso la utilización del PSO como un planeador de trayectorias que fuera actualizando el punto de referencia a seguir para que el robot fuera ejecutando maniobras suaves y controladas. No se pretendió seguir la línea recta generada por el vector de velocidad PSO al actualizar la nueva posición PSO del algoritmo, sino que el punto de nueva posición PSO se tomó como punto de referencia actualizable que los controladores persiguieran realizando segmentos curvos y continuos.

En la Figura 45, se puede observar la trayectoria descrita por el robot denotada por la línea azul. La meta de la función de costo utilizada en este caso se encuentra localizada en la coordenada (0,0) donde se observa el punto negro. En este caso, no se graficaron todos los vectores generados en cada iteración del PSO en dicha simulación para no saturar la gráfica. Se puede observar que mientras el robot se mueve a lo largo de su trayectoria, el planeador PSO calcula el vector de velocidad a seguir (líneas punteadas) y actualiza la nueva posición PSO. Los puntos rojos señalan las nuevas posiciones PSO calculadas en cada iteración del planeador. En esta investigación se referirá a estos puntos como **Marcadores PSO**.

Figura 45: Marcadores PSO para seguimiento de trayectoria



Los controladores de los robots diferenciales toman cada marcador PSO como punto de referencia al que deben llegar realizando un segmento curvo y continuo. Los controladores se encargan de reducir el error entre la posición actual del robot y la posición del marcador PSO actual. También se puede observar que la trayectoria no necesariamente pasa por cada uno de los marcadores PSO. Por la naturaleza quebradiza del PSO, los marcadores varían considerablemente en orientación y posición, y tratar de alcanzar cada punto causaría trayectorias muy irregulares.

Para lograr trayectorias regulares y continuas, el planeador PSO y el control del robot se ejecutaron simultáneamente. Primeramente, el robot comienza a describir la trayectoria necesaria para alcanzar el primer marcador PSO. Luego de una cantidad definida de iteraciones del controlador, el planeador PSO vuelve a actualizar la posición del marcador PSO sin necesidad de que el controlador haya logrado llevar al robot hacia el marcador anterior. El controlador vuelve a describir un segmento curvo, ahora en dirección hacia el nuevo marcador PSO. Y así se ejecutó el algoritmo sucesivamente hasta que el enjambre lograba llegar a la meta global de la función de costo analizada. Cada segmento curvo ejecutado para cada marcador PSO compuso una sección de la trayectoria continua descrita por cada robot convergiendo hacia la meta.

9.1. Parámetros del planeador de trayectorias PSO

Para el ajuste del planeador PSO se tuvieron los siguientes aspectos en cuenta:

- El período de muestreo de la simulación T_s tiene un efecto directo en la cantidad de tiempo que el controlador del robot tiene disponible para ejecutar una trayectoria hacia el marcador PSO actual.
- Si el período de muestreo es demasiado pequeño, el controlador ejecuta segmentos curvos muy cortos causando que la convergencia sea muy lenta.
- Si el período de muestreo es muy grande, los segmentos curvos ejecutados por el controlador en cada iteración son de mayor tamaño. Esto causa trayectorias irregulares, ya que el robot trata de acercarse más a cada marcador PSO individual.
- La longitud de los vectores de velocidad PSO dictan la distancia entre el robot y cada marcador PSO. Los controladores ejecutan el cálculo de velocidad dependiendo la magnitud del error de distancia entre estos. Por lo tanto, vectores de velocidad más largos causan aumentos más bruscos de velocidad en cada actualización del planeador PSO. Asimismo, vectores de velocidad muy cortos causarían una elevación en el tiempo de convergencia del enjambre hacia la meta.
- La proporción entre la velocidad de actualización del planeador PSO y la velocidad de actualización de señales de los controladores afecta directamente la regularidad de las trayectorias y de las velocidades de los actuadores de los robots. Si el planeador PSO actualiza la posición del marcador PSO cada 10 iteraciones del controlador, por ejemplo, los segmentos curvos descritos por el controlador rastreando cada marcador PSO son de mayor tamaño.

Para lograr regular todos los aspectos del planeador PSO anteriormente descritos, se variaron los valores de tres parámetros específicos:

1. Período de muestreo T_s : este parámetro descrito en el capítulo anterior puede tomar valores que sean múltiplos de 8 (16ms, 32ms, 64ms). Se utilizó para regular el tiempo que el controlador tiene disponible para ejecutar un paso del robot diferencial.
2. Escalamiento de velocidad PSO η : este parámetro también descrito en el capítulo anterior se encarga de escalar la longitud de los vectores de velocidad PSO a la hora de actualizar la posición PSO. Se buscó utilizar un η que lograra acelerar suficientemente el tiempo de convergencia del enjambre hacia la meta sin causar cambios bruscos de velocidad en cada actualización de los marcadores PSO.
3. PSO Step P_s : este parámetro determina el número de iteraciones del controlador que se deben ejecutar antes de cada actualización del marcador PSO por parte del planeador PSO. Cabe notar que el algoritmo PSO siempre se encuentra en ejecución calculando los vectores de velocidad PSO. Sin embargo, únicamente se actualiza la posición del marcador PSO cuando transcurre P_s .

9.2. Ecuación modificada de actualización de velocidad PSO

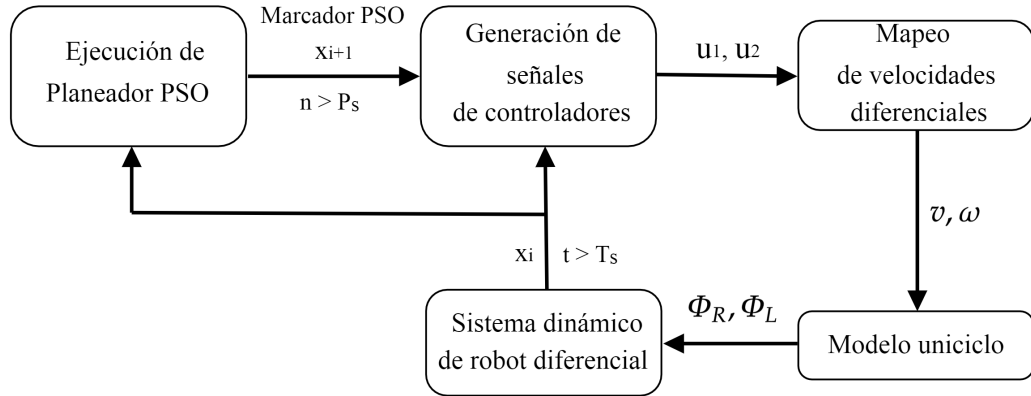
Para tener un control más directo de la longitud de los vectores PSO, se implementó el parámetro η . Esto se realizó con el objetivo de lograr que las posiciones de los **Marcadores PSO** siempre se mantuvieran dentro de las fronteras del espacio de búsqueda y así reducir movimientos bruscos de los agentes. Por lo tanto, la actualización de la posición PSO se realiza con una versión modificada de (2):

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \eta \mathbf{V}_{i+1} \quad (84)$$

9.3. Ciclo de ejecución de planeador PSO

En la siguiente figura se observa el ciclo de ejecución de las simulaciones en donde la iteración del controlador se realiza cada vez que transcurren T_s milisegundos y la actualización del marcador PSO se ejecuta cada P_s iteraciones.

Figura 46: Ciclo de ejecución de MPSO



En el siguiente capítulo, se presentan los diferentes controladores evaluados en esta investigación. Estos varían en velocidad y en agresividad de movimiento de los robots diferenciales. Por lo tanto, los parámetros del planeador PSO se debieron ajustar individualmente para cada controlador para lograr trayectorias y señales de control adecuadas. Se consideró como trayectoria adecuada aquella que garantizara una convergencia del enjambre en menos de 30 segundos, que fuera continua y suave. Se consideró como una señal de control adecuada aquella que fuera suficientemente continua y suave como para su aplicación en actuadores de robots reales sin dañar estos mismos por cambios de velocidad bruscos, vibraciones innecesarias o saturación de velocidad constante.

Implementación de controladores a robots de la simulación

El objetivo de esta investigación era principalmente lograr implementar el algoritmo PSO clásico a un enjambre de robots diferenciales para que estos pudieran converger en un tiempo finito hacia una meta definida dentro del espacio de tarea. Como se describió en el capítulo anterior, el PSO se utilizó como un planeador de trayectorias suaves. Este permitió trasladar la naturaleza quebradiza de las trayectorias de partículas a curvas suaves de convergencia hacia la meta.

Sin embargo, al llevar a cabo esta implementación, se observó que esta investigación no solamente debía enfocarse en las trayectorias generadas para la convergencia del enjambre hacia la meta, sino también en la suavidad de las señales de control a enviar a los actuadores de los robots. Para que este algoritmo fuera implementable en robots realistas con restricciones físicas y dinámicas (como los que se utilizaron en las simulaciones de WeBots), este debía garantizar que las velocidades de los robots fueran suficientemente suaves y regulares como para evitar dañar los motores de los robots.

Se buscó evitar la saturación excesiva de la velocidad de los motores, así como los cambios bruscos en la misma. En robots reales, controladores que causan muchos cambios bruscos pueden causar mucha vibración en la estructura del robot e incluso pueden quemar la circuitería interna de los actuadores. Por lo tanto, en esta investigación fue de gran importancia lograr determinar qué tipo de controlador garantizaba una convergencia exacta y rápida a la meta siguiendo la trayectoria del planeador PSO, y que a la vez disminuyera la saturación y cambios bruscos de velocidad en los actuadores de los robots. Se realizaron simulaciones en WeBots acoplando el planeador de trayectorias PSO a los diferentes controladores presentados en este capítulo.

10.1. Control proporcional de cinemática transformada (TUC)

10.1.1. Diseño de controlador de cinemática transformada

Como primer controlador se utilizó la transformación de cinemática del unicycle planteada en las ecuaciones (30) y (31). Este es un controlador proporcional que toma el error de posición (x, y) entre el robot y el marcador PSO de cada iteración del planeador de trayectorias para determinar las velocidades \dot{x} y \dot{y} del robot. Estas velocidades, a través de la transformada de cinemática de unicycle, se mapeaban a la velocidad lineal v y angular ω del robot.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\frac{1}{l}\sin\phi & \frac{1}{l}\cos\phi \end{bmatrix} \cdot \begin{bmatrix} \dot{x}_d + I_x \cdot \tanh\left(\frac{k_x}{I_x} \cdot \tilde{x}\right) \\ \dot{y}_d + I_y \cdot \tanh\left(\frac{k_y}{I_y} \cdot \tilde{y}\right) \end{bmatrix}, \quad (85)$$

En este caso, el ángulo ϕ de la ecuación (85) simboliza la orientación del robot respecto al eje horizontal del marco inercial. Por lo tanto, de aquí en adelante, se le referirá a este ángulo como θ . Este ángulo fue computado con la ecuación (77) utilizando los datos del nodo *Compass* de WeBots. Las variables \tilde{x} y \tilde{y} son los errores de posición. Por lo tanto, de ahora en adelante se referenciarán como e_x y e_y . La variable l es la longitud entre el centro del robot y una de las llantas de este. Las variables \dot{x}_d y \dot{y}_d son despreciables ya que las velocidades de un marcador PSO son cero.

Las variables I_x e I_y son las constantes de saturación para fijar el valor asintótico máximo limitado por el tangente hiperbólico. Su función es limitar el valor máximo que puede retornar este controlador proporcional en caso que el error de posición sea muy grande. Este valor fue ajustado para evitar la saturación de los motores angulares de los robots. Las constantes proporcionales k_x y k_y se calcularon con la ecuación (33).

Leyes de control de cinemática transformada:

$$v = I_x \cdot \tanh\left(\frac{k_x}{\Phi_{max}} e_x\right) \cdot \cos(\theta) + I_y \cdot \tanh\left(\frac{k_y}{\Phi_{max}} e_y\right) \cdot \sin(\theta) \quad (86)$$

$$\omega = \frac{-I_x}{l} \cdot \tanh\left(\frac{k_x}{\Phi_{max}} e_x\right) \cdot \sin(\theta) + \frac{I_y}{l} \cdot \tanh\left(\frac{k_y}{\Phi_{max}} e_y\right) \cdot \cos(\theta) \quad (87)$$

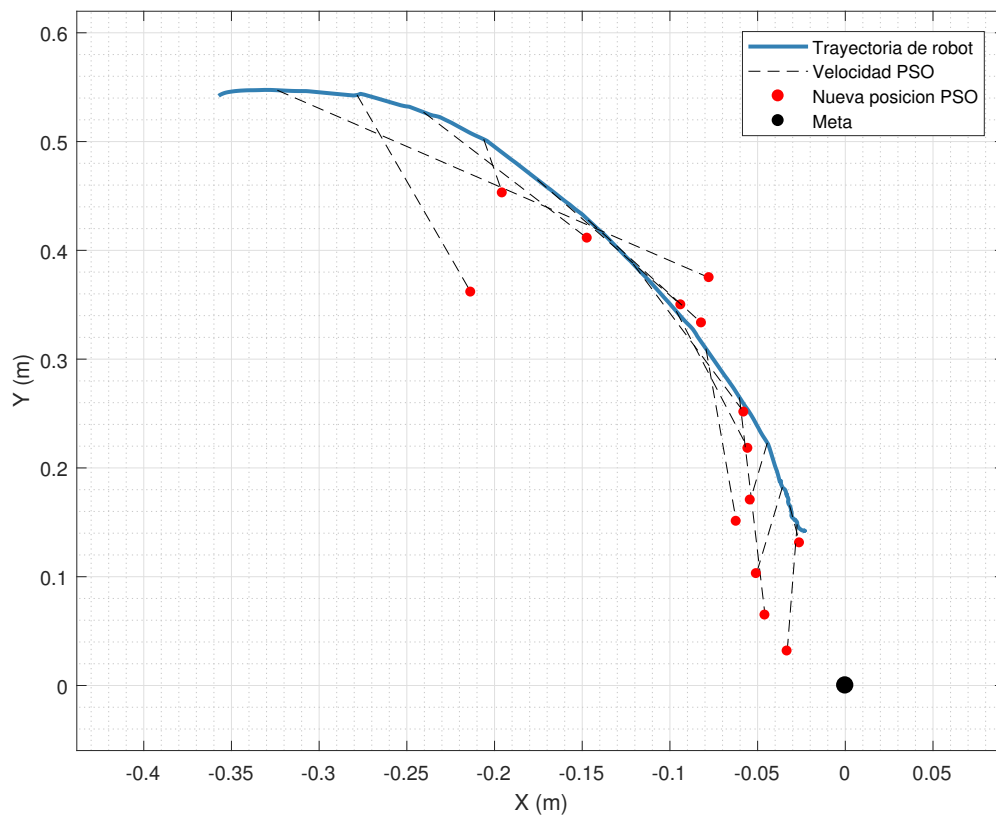
Las constantes de saturación I_x e I_y dentro de $\tanh(\cdot)$ se reemplazaron por la variable MAX_SPEED $\Phi_{max} = 6.28\text{m/s}$. Esta es la velocidad angular de saturación de los motores de los robots.

Implementación de controlador de cinemática transformada: Se realizaron simulaciones en WeBots para observar el comportamiento de los robots al converger a la meta. Se recopiló información como la trayectoria realizada por un robot del enjambre, así como sus velocidades lineal, angular, y rotacional de cada motor. En este caso, se utilizó la función de costo *Sphere* para observar la convergencia del enjambre al origen $(0,0)$ del espacio.

Cuadro 7: Parámetros utilizados de planeador PSO y de controlador TUC

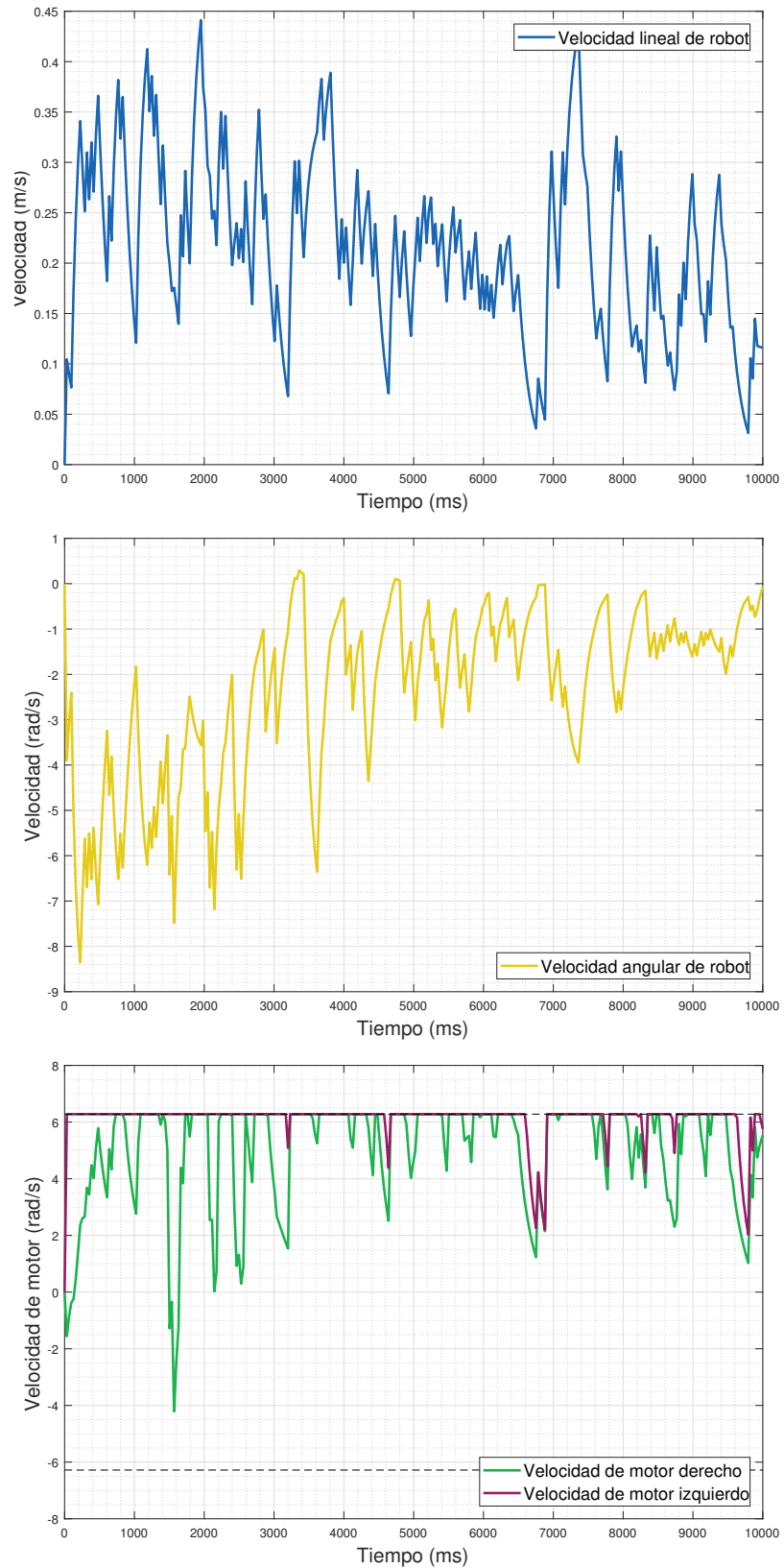
Parámetros de simulación	Valor de parámetro
T_s (ms)	16.00
η	0.02
P_s	1 iteración
$I_{x,y}$	2.00
l	35.00
v_o Ec. (33)	3.12
α Ec. (33)	2.00

Figura 47: Trayectoria generada por marcadores PSO y controlador TUC



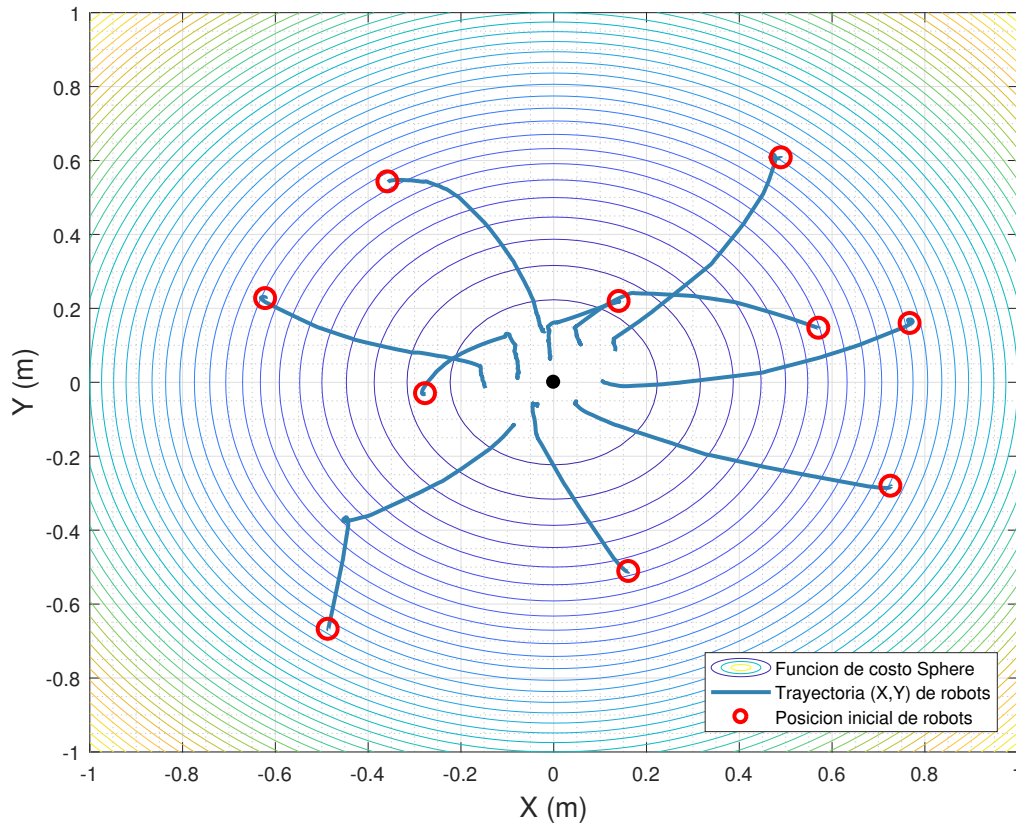
En la Figura 47, se puede observar la trayectoria en azul del agente "E-Puck(0)" de la simulación en WeBots. El punto negro con coordenadas (0,0) denota el mínimo absoluto de la función de costo *Sphere*. Se puede observar que los marcadores PSO generados en dicha simulación fueron capaces de generar una trayectoria suave hacia la meta.

Figura 48: Velocidad lineal y angular de robot con controlador de cinemática transformada



Como se puede observar en las figuras anteriores, el controlador de cinemática transformada para acoplar el PSO a los robots, sí logra llevar al robot hacia la meta. Se observa que las trayectorias son considerablemente suaves. Sin embargo, al observar las velocidades angulares de los motores, se puede notar que existen muchos cambios bruscos en fracciones de segundo, así como completa saturación de uno de los motores. Esto equivale a considerable vibración dentro de los robots y cambios de velocidad que podrían dañar los actuadores. Este controlador, con la configuración propuesta, no sería aplicable en la vida real, ya que podría no actuar de la manera esperada o incluso dañar al robot.

Figura 49: Trayectorias resultantes de enjambre con controlador de cinemática transformada



10.2. Controlador PID de velocidad angular (TUC-PID)

El siguiente controlador utilizado fue un PID clásico (32), únicamente para la velocidad angular ω del robot. La velocidad lineal v se operó con el controlador anterior en conjunto con el PID angular. Se realizaron varias iteraciones para configurar los parámetros de este y lograr suavizar las trayectorias así como suavizar las trayectorias de las velocidades. Se observó que, para que el sistema tuviera un comportamiento convergente hacia la meta, se debía cumplir la relación:

$$K_p > K_i > K_d \quad (88)$$

Luego de varias iteraciones del algoritmo utilizando el controlador PID para velocidad

angular ω , se determinó que la configuración adecuada para tener la trayectoria más suave posible y velocidad más regulada era la siguiente:

Cuadro 8: Parámetros utilizados de planeador PSO y de controlador TUC-PID (Velocidad lineal)

Parámetros de simulación	Valor de parámetro
T_s (ms)	32.00
η	1.00
P_s	1 iteración
$I_{x,y}$	2.00
l	35.00
v_o Ec.(33)	3.12
α Ec.(33)	2.00

Cuadro 9: Parámetros utilizados de controlador TUC-PID (Velocidad angular)

Parámetros de simulación	Valor de parámetro
K_p	0.500
K_i	0.100
K_d	0.001

Figura 50: Trayectoria generada por marcadores PSO y controlador TUC-PID

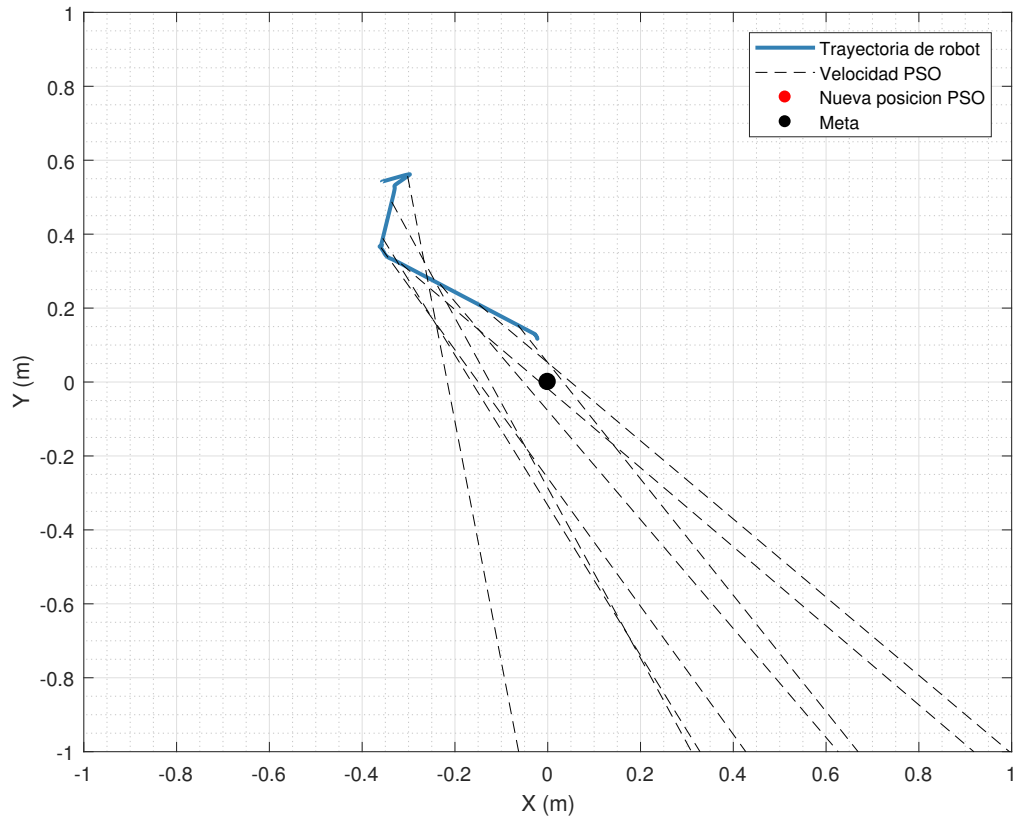
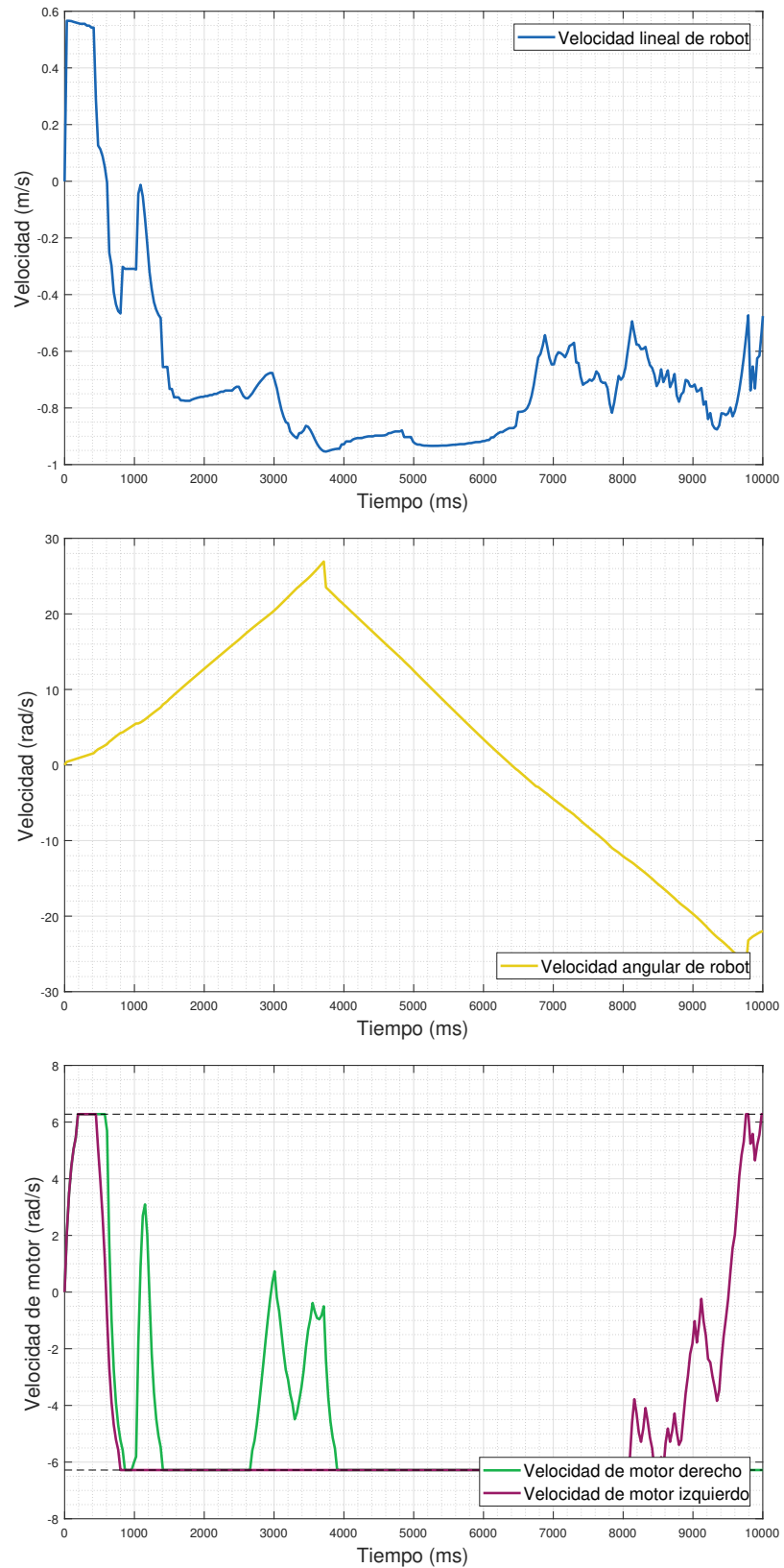
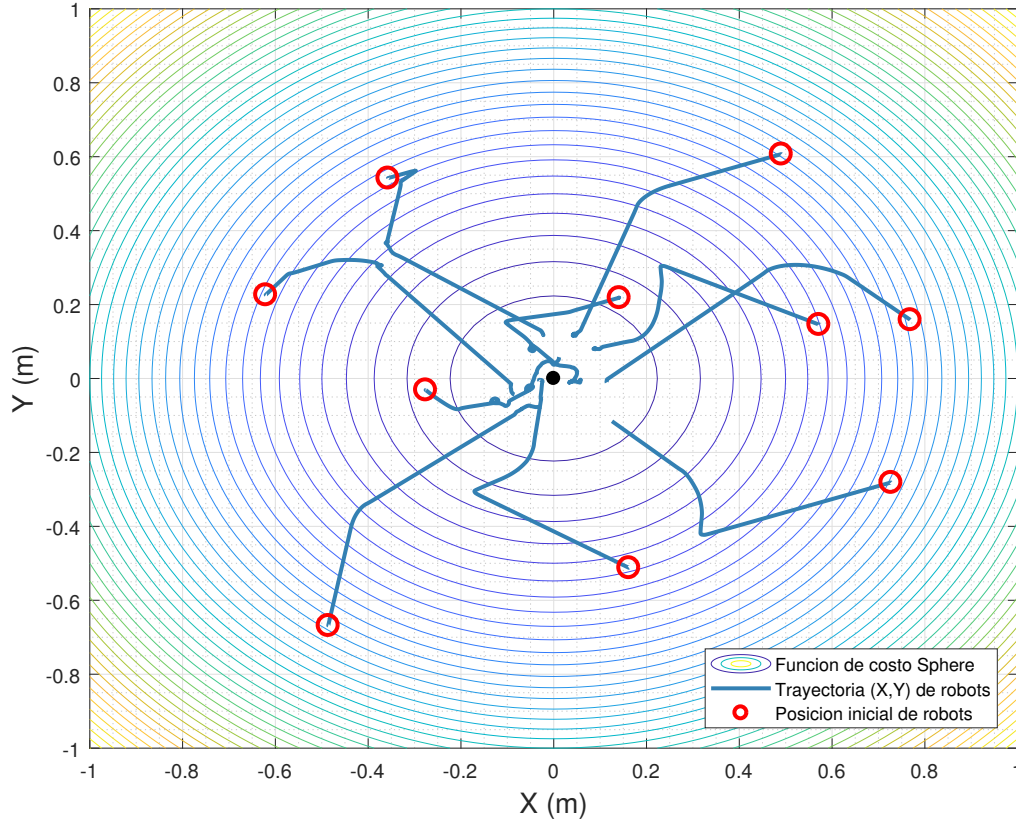


Figura 51: Velocidad lineal y angular de robot con controlador de cinemática y PID angular



En este caso, se observó que acoplar el controlador PID a ω logró suavizar tanto la velocidad lineal como angular del robot. Se puede comparar la irregularidad de la velocidad angular mostrada en la Figura 48b. y la suavidad de velocidad mostrada en la Figura 51b. En cuanto a las velocidades de los motores, se observó una reducción en la saturación de ambos motores. Aún se presentaron algunos picos bruscos de cambio de velocidad pero en menor cantidad que con el controlador TUC.

Figura 52: Trayectorias resultantes de enjambre con controlador de cinemática y PID angular



Se puede observar que las trayectorias fueron menos suaves en forma. Sin embargo, la velocidad de los motores se comportó de mejor manera. Cerca del centro se observó mayor irregularidad de las trayectorias a causa de colisiones entre robots. Para reducir los picos de velocidad fue posible acoplar un filtro de media móvil ponderada para recalcular $\Phi_{R,L}$ si el cambio entre iteraciones excedía 1 rad/s; esto sin afectar las trayectorias:

$$\Phi_{new} = \frac{\Phi_{new} + 2 \cdot \Phi_{old}}{3} \quad (89)$$

10.3. Controlador simple de pose de robot diferencial (SPC)

Para la implementación de este controlador, se utilizaron las leyes de control expresadas en (41) y (42), tanto para determinar la velocidad lineal v del robot como la velocidad

angular ω . Los parámetros de pose que utiliza este controlador (ρ, α, β) , se calcularon con las expresiones (36) (37) (38).

La configuración que se utilizó para los parámetros del controlador simple de pose respetando las restricciones para estabilidad fue la siguiente:

Cuadro 10: Parámetros utilizados de planeador PSO y de controlador SPC

Parámetros de simulación	Valor de parámetro
T_s (ms)	32.000
η	0.032
P_s	5 iteraciones
K_ρ	0.10
K_α	0.50
K_β	0.05

Figura 53: Trayectoria generada por marcadores PSO y controlador SPC

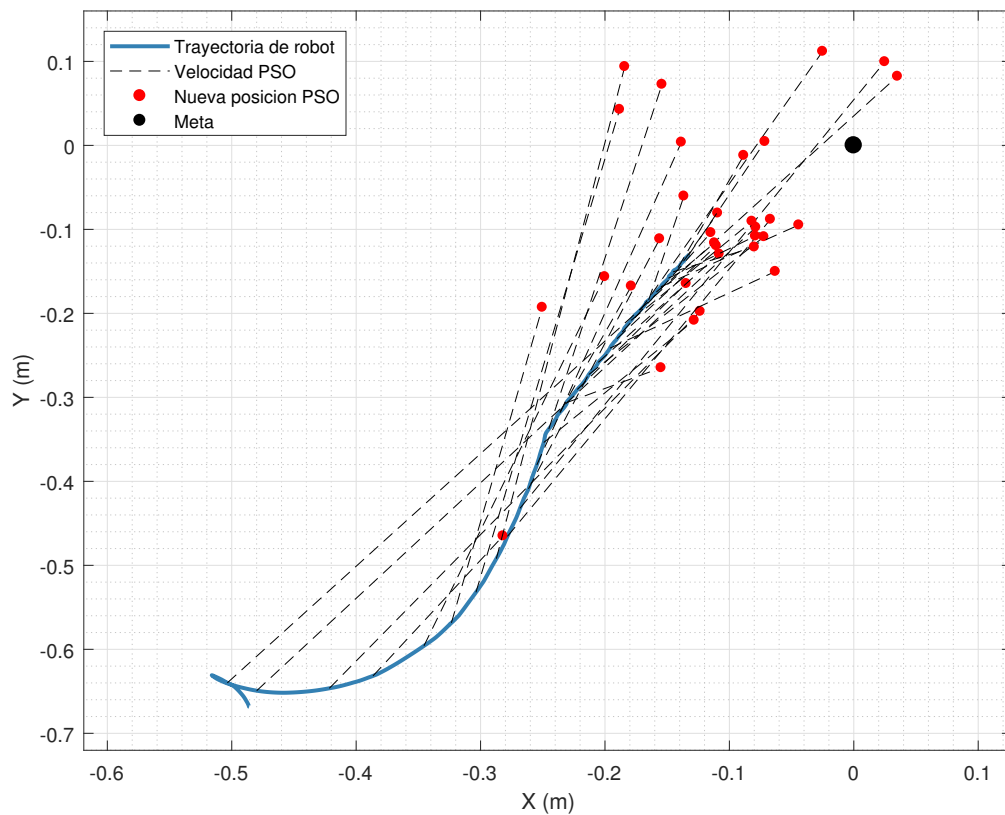
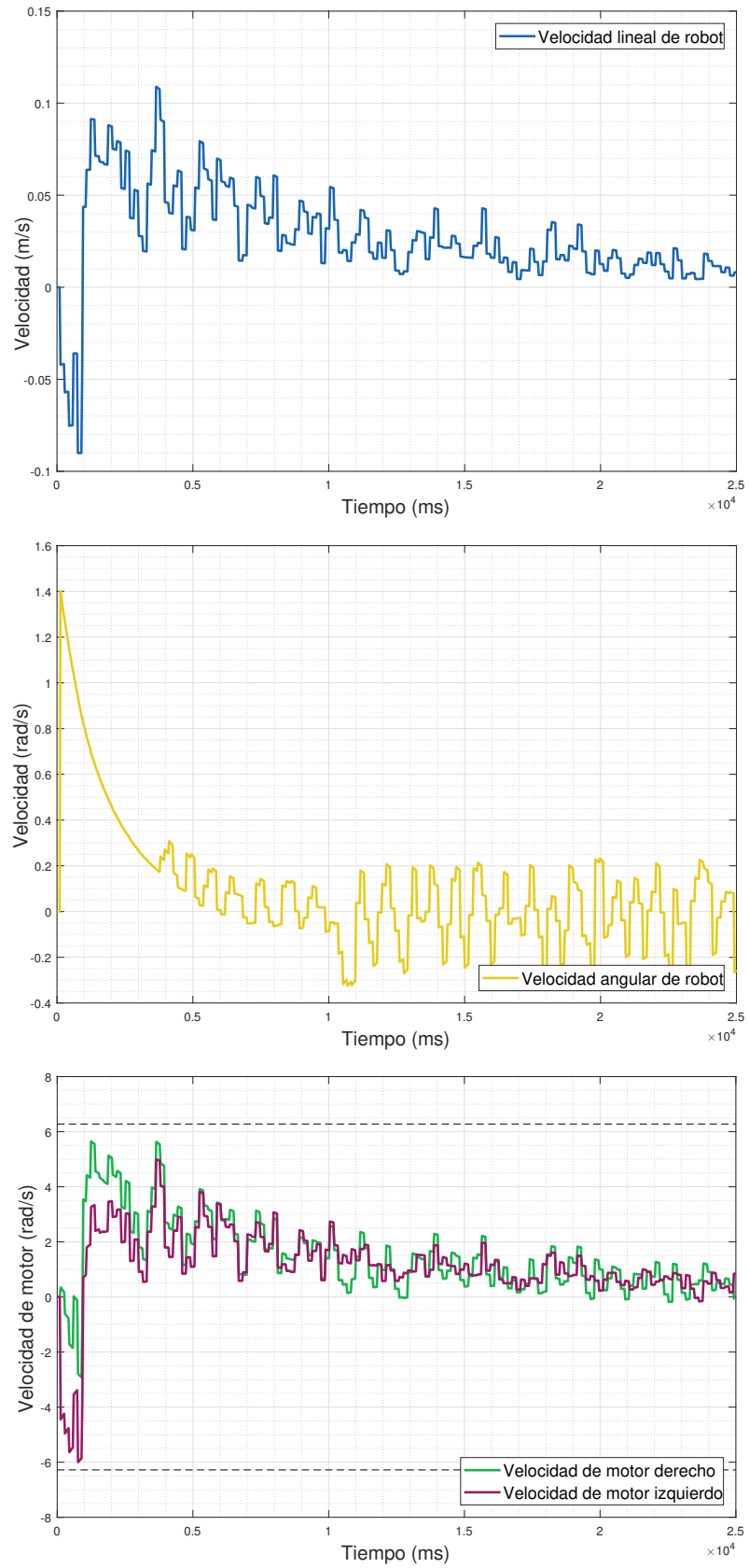
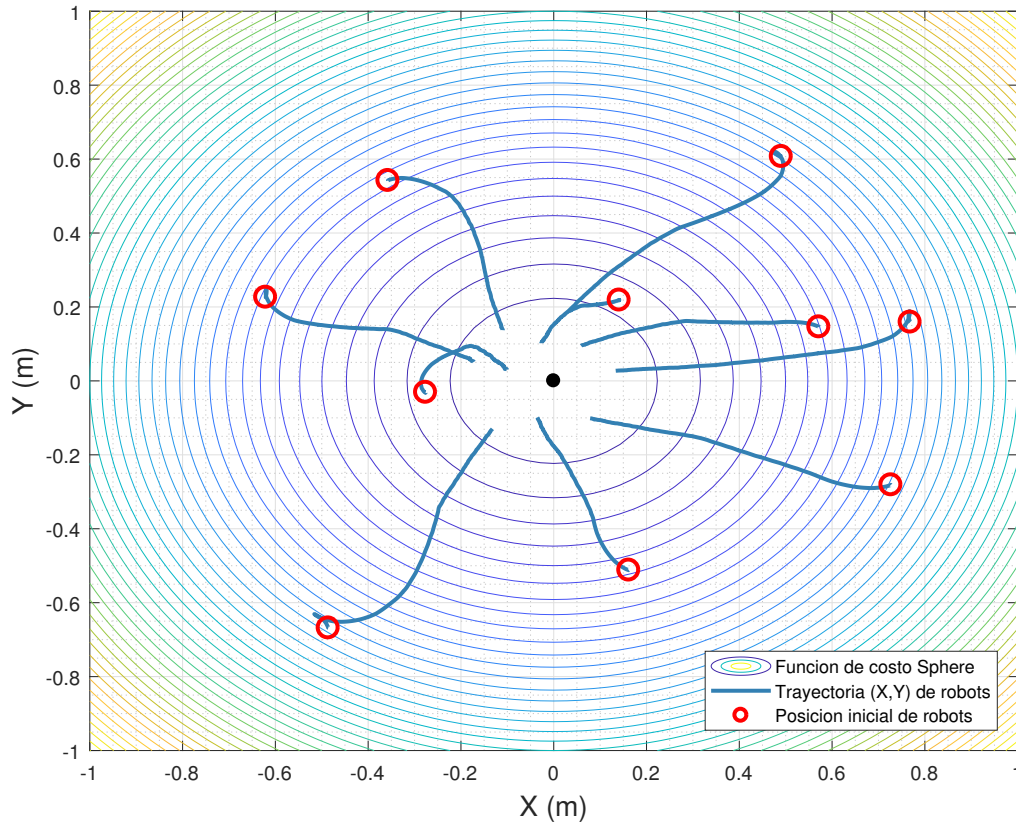


Figura 54: Velocidad lineal y angular de robot con controlador simple de pose



Como se observa en las figuras anteriores, este controlador presentó velocidades bastante irregulares. Se pudo notar que las velocidades lineal y angulares del robot, así como las velocidades rotacionales de los motores convergían a cero con el tiempo. Esto verificó el funcionamiento adecuado del controlador simple de pose al converger hacia la meta. Sin embargo, como el algoritmo PSO se basa en la actualización constante de puntos objetivo a seguir, se presentó un aumento de velocidad cada vez que se actualizaba el PSO. Por lo tanto, las velocidades presentaron picos bruscos que causaban vibración en los robots. Se ejecutó el planeador PSO cada 5 iteraciones del controlador para darle como mínimo 160 milisegundos al motor para reaccionar a los cambios de velocidad.

Figura 55: Trayectorias resultantes de enjambre con controlador simple de pose



10.4. Controlador Lyapunov de pose de robot diferencial (LSPC)

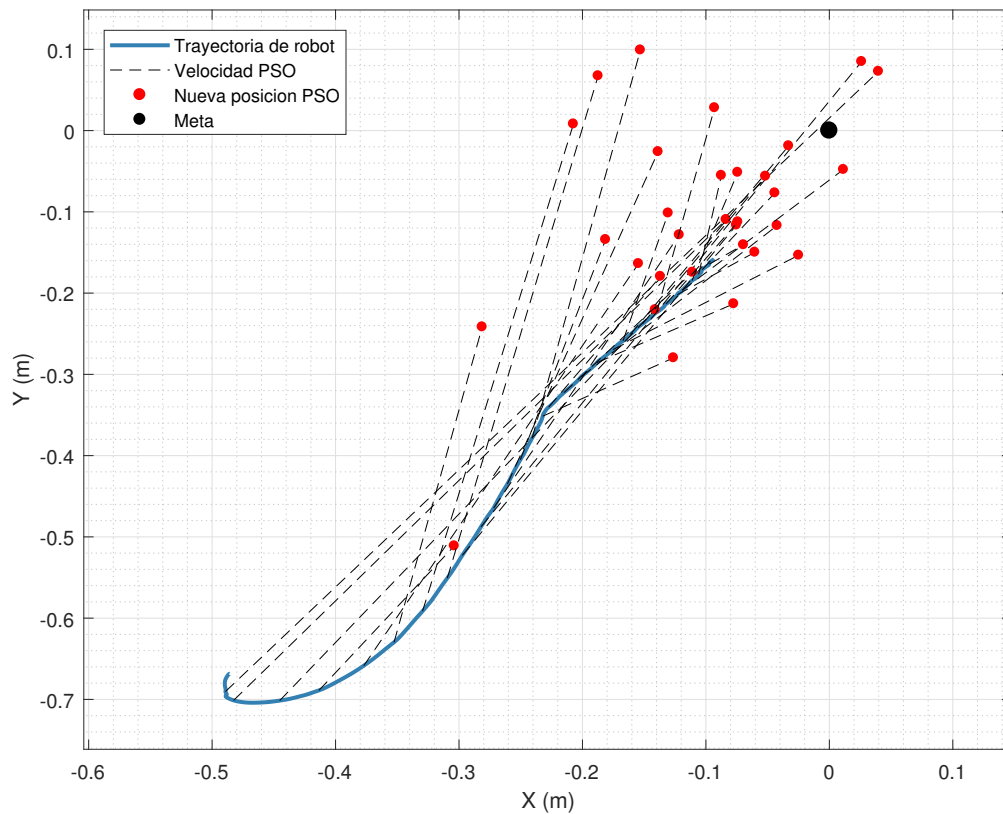
Para la implementación de este controlador, se utilizaron las leyes de control expresadas en (47) y (48), tanto para determinar la velocidad lineal v del robot como la velocidad angular ω . Los parámetros de pose que utiliza este controlador (ρ, α) , se calcularon con las expresiones (36) (37).

La configuración que se utilizó para los parámetros del controlador Lyapunov de pose respetando las restricciones para estabilidad fue la siguiente:

Cuadro 11: Parámetros utilizados de planeador PSO y de controlador LSPC

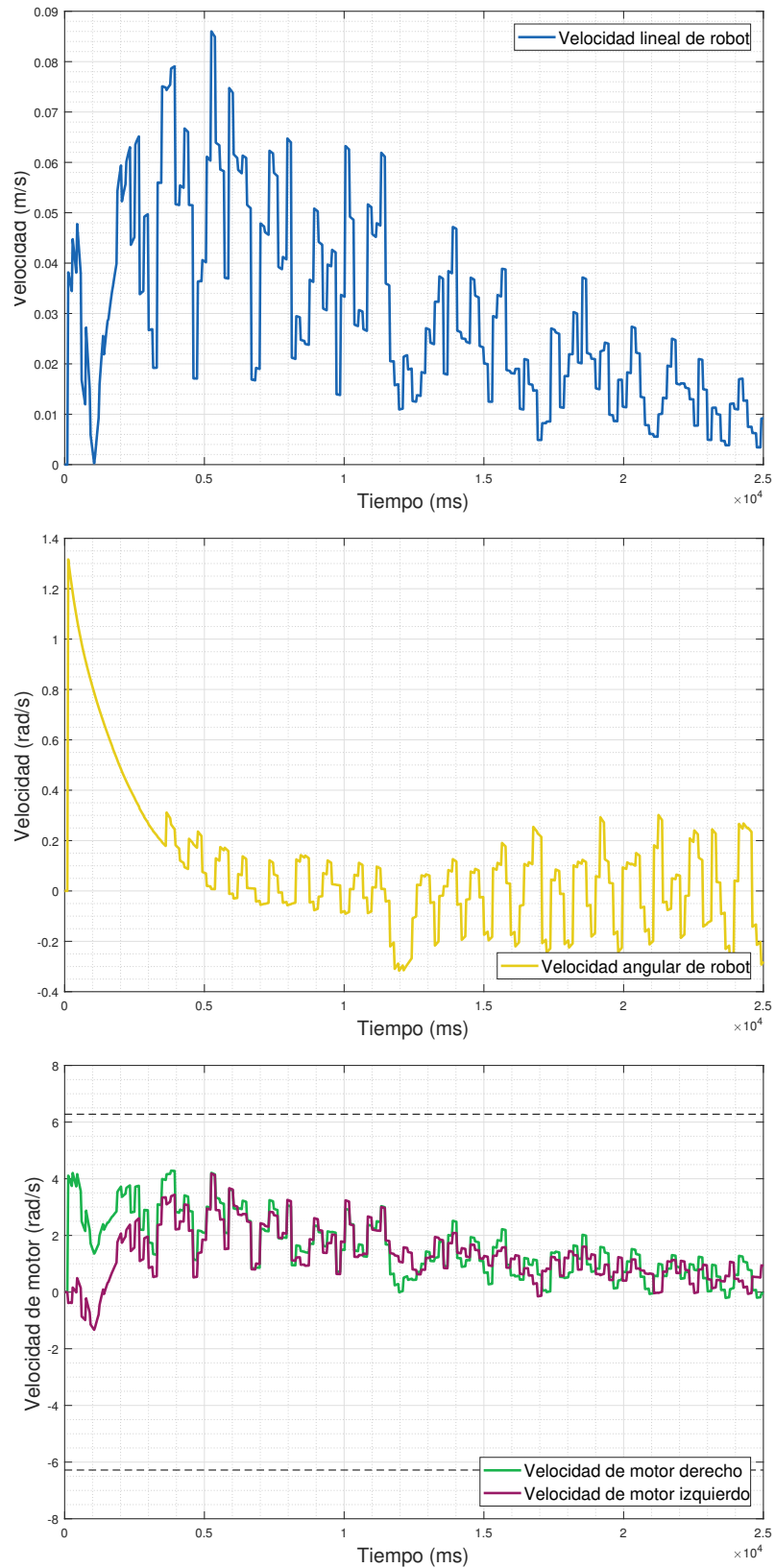
Parámetros de simulación	Valor de parámetro
T_s (ms)	32.000
η	0.032
P_s	5 iteraciones
K_ρ	0.10
K_α	0.50

Figura 56: Trayectoria generada por marcadores PSO y controlador LSPC



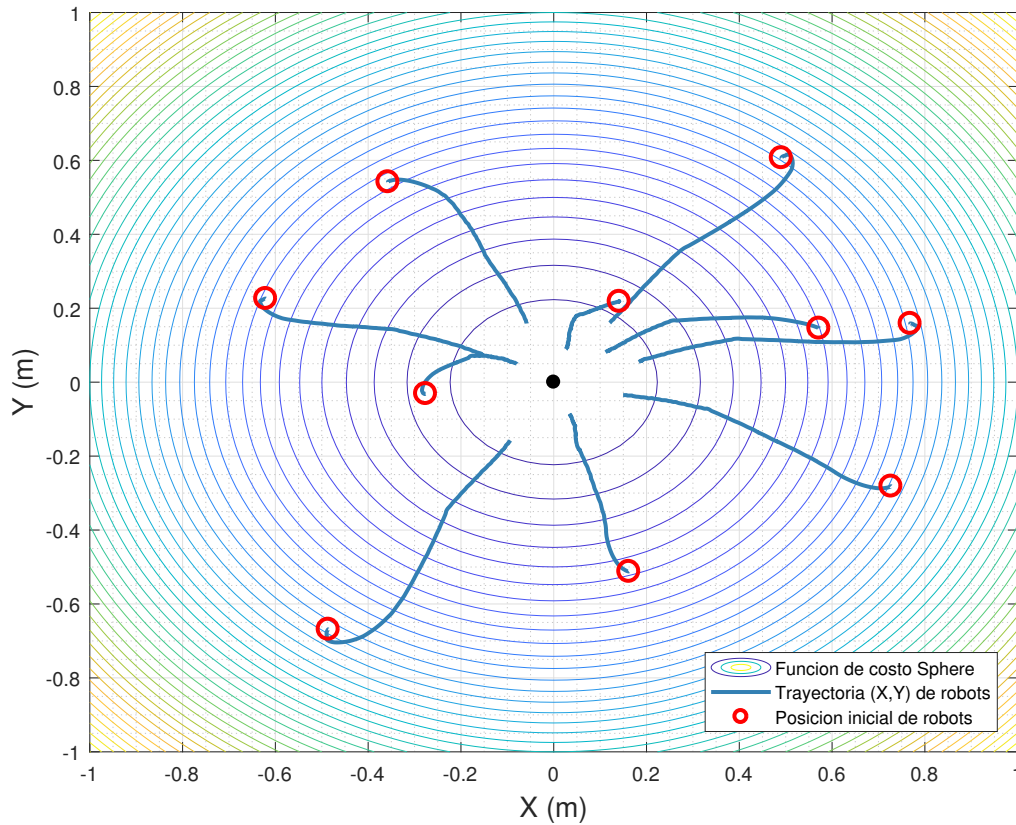
Al utilizar este controlador, se observaron las mismas trayectorias suaves que se obtuvieron con controlador simple de pose. Sin embargo, en la Figura 56, se puede observar que los vectores PSO estaban ligeramente más alejados de ser tangentes a la curva de la trayectoria. Esto fue causado por el hecho de que el controlador Lyapunov de pose mantiene el ángulo β entre la orientación del robot y la meta como una variable libre. Este ángulo no se toma en cuenta para las leyes de control que utiliza este controlador. El efecto que esto tuvo sobre las trayectorias es que se redujo la curvatura de las mismas, ya que el robot no trataba de orientar la línea de trayectoria hacia una pose específica sobre la meta.

Figura 57: Velocidad lineal y angular de robot con controlador de pose Lyapunov



Como se observa en las figuras anteriores, este controlador también presentó velocidades bastante irregulares. Se pudo notar que las velocidades lineal y angulares del robot, así como las velocidades rotacionales de los motores convergían a cero con el tiempo. Esto verificó el funcionamiento adecuado del controlador de pose Lyapunov al converger hacia la meta. Sin embargo, como el algoritmo PSO se basa en la actualización constante de puntos objetivo a seguir, se presentó un aumento de velocidad cada vez que se actualizaba el PSO. Por lo tanto, las velocidades presentaron picos bruscos que causaban vibración en los robots. Se ejecutó el planeador PSO cada 5 iteraciones del controlador para darle como mínimo 160 milisegundos al motor para reaccionar a los cambios de velocidad.

Figura 58: Trayectorias resultantes de enjambre con controlador Lyapunov de pose



En la Figura 58, se puede observar que ya no se presentó ningún pico al inicio de las trayectorias de los robots, al contrario de lo que se pudo observar en la Figura 55. Esto fue a causa de que el controlador de pose Lyapunov no tomaba en cuenta el cambio de signo en la velocidad lineal de los robots. Por lo tanto, estos solo se podían mover hacia adelante a diferencia del controlador simple de pose que sí permitió el desplazamiento hacia atrás.

10.5. Controlador de direccionamiento en lazo cerrado (CLSC)

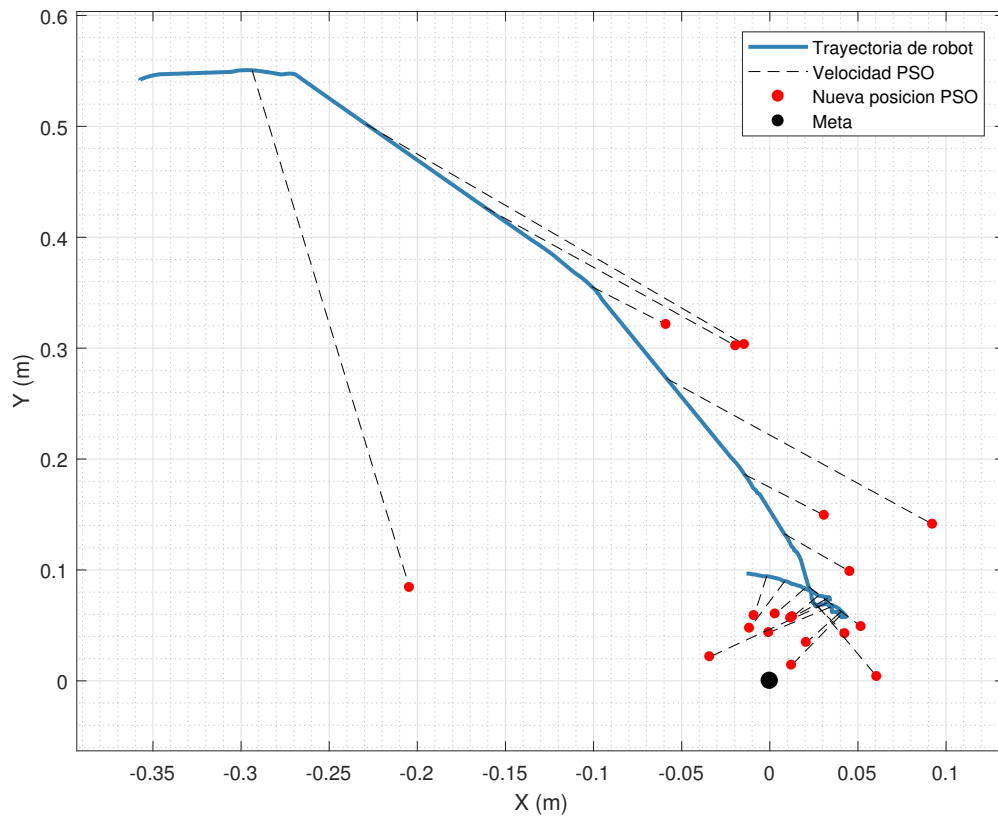
Este controlador se basó en implementar la ley de control (59) para la velocidad angular ω del robot en conjunto con el controlador de cinemática transformada para la velocidad

lineal v . Se utilizó la siguiente configuración de controlador para lograr la convergencia por medio de trayectorias de curvatura reducida:

Cuadro 12: Parámetros utilizados de planeador PSO y de controlador CLSC

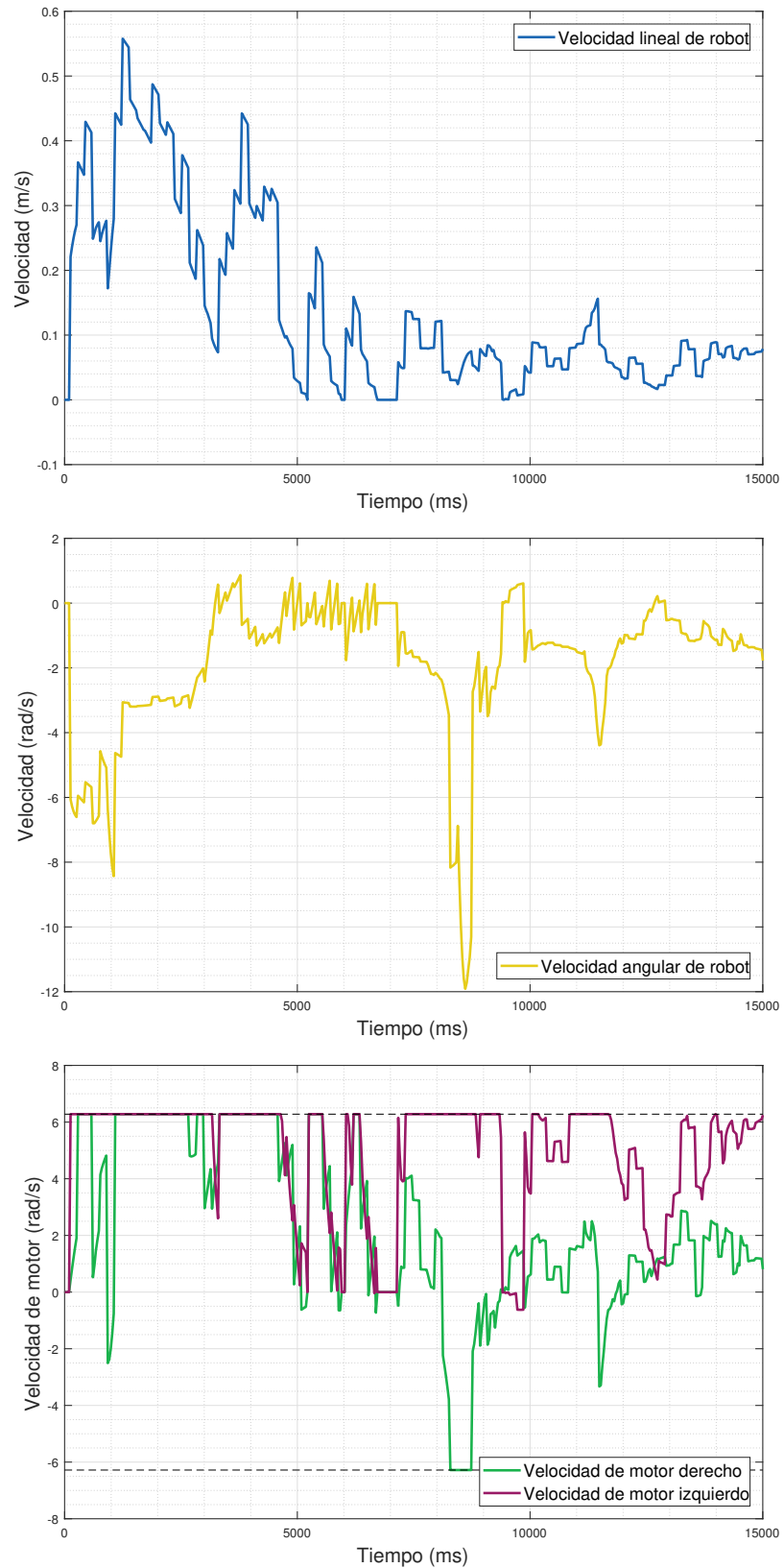
Parámetros de simulación	Valor de parámetro
T_s (ms)	32.000
η	0.032
P_s	5 iteraciones
K_1	1.00
K_2	10.00

Figura 59: Trayectoria generada por marcadores PSO y controlador CLSC



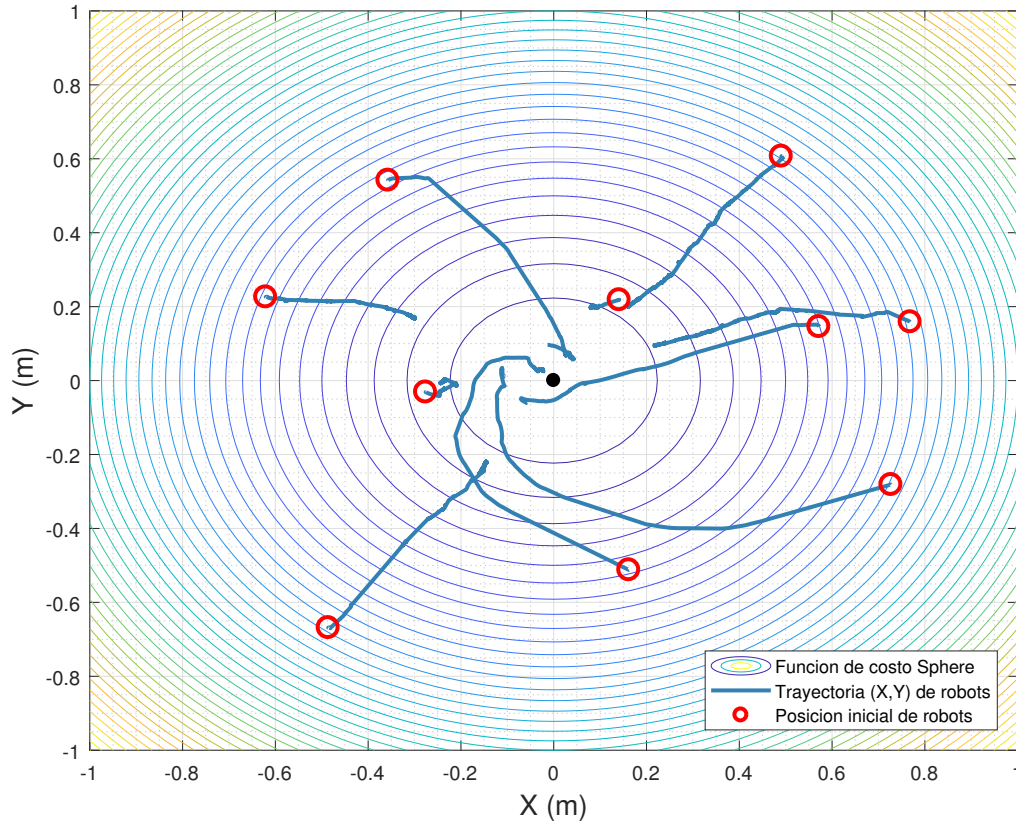
Al utilizar este controlador, se observaron trayectorias irregulares. En la figura anterior se observa que con este controlador, los robots describían trayectorias más amplias e incluso existían sectores en donde se mantenían únicamente en vibración.

Figura 60: Velocidad lineal y angular de robot con controlador de direccionamiento



En las figuras anteriores, se puede observar que el controlador de direccionamiento es agresivo y causa variaciones rápidas de velocidad. En algunos casos, los robots presentaron elevada vibración que podría llegar a dañar los motores de los robots. Por lo tanto, el uso de este controlador no es recomendado.

Figura 61: Trayectorias resultantes de enjambre con controlador de direccionamiento



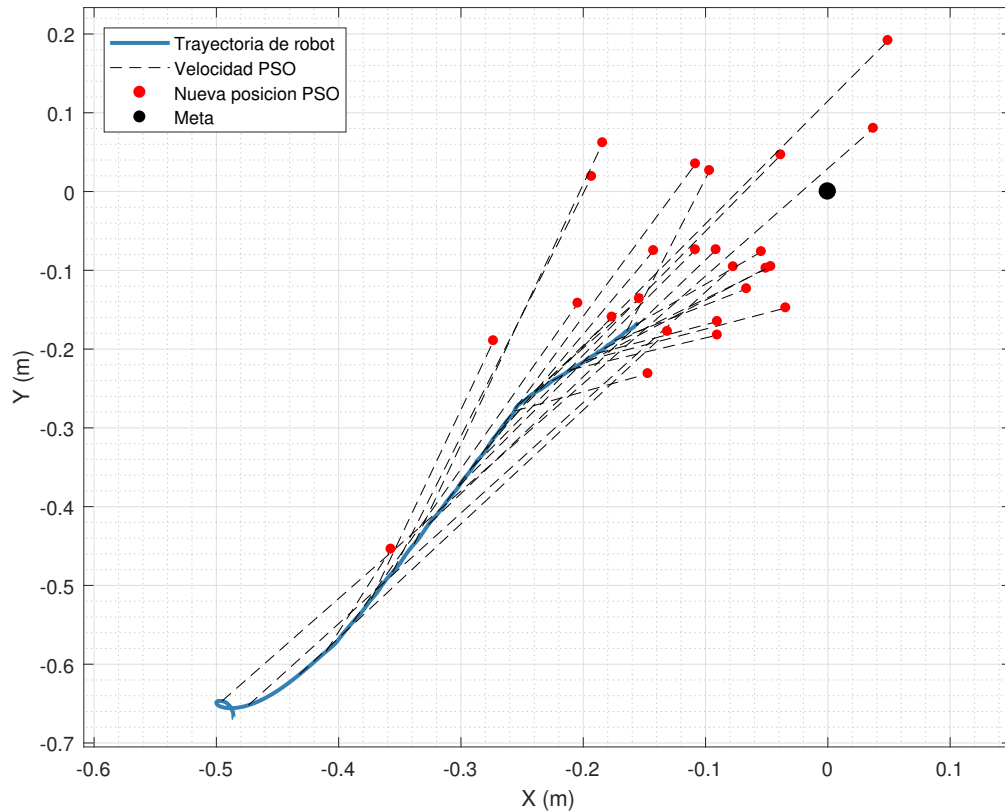
10.6. Controlador regulador lineal cuadrático y uniciclo transformado (TUC-LQR)

Este controlador se basó en implementar el controlador de cinemática transformada presentado en la Sección 9.1 de esta investigación. La diferencia en este caso fue que las entradas u_1 y u_2 de la ecuación (30) ya no se plantearon de la misma manera. En lugar de utilizar la ecuación (31) para el cálculo de las entradas, se utilizó el análisis de variables de espacio estado presentado en la ecuación (67). K era la matriz de coeficientes dada por el cálculo óptimo de controlador LQR realizado en MATLAB ($lqr(A,B,Q,R)$). El planeador PSO se configuró igual que para el LSPC (11). Se utilizó la siguiente configuración de controlador para lograr la convergencia por medio de trayectorias de curvatura reducida:

Cuadro 13: Características de simulación con controlador TUC-LQR

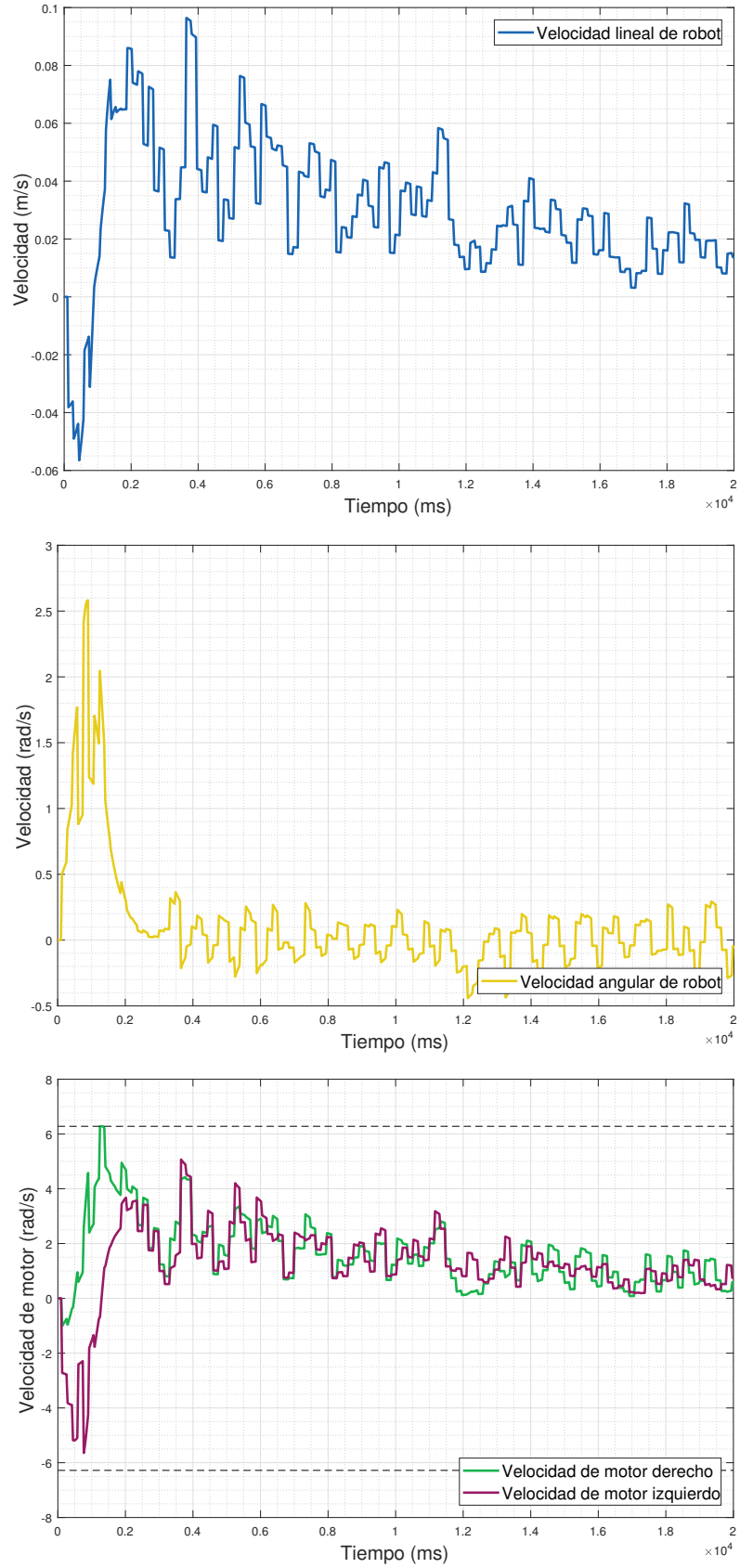
Parámetros de simulación	Tamaño de matriz	Valor de matriz
Matriz A	2x2	$0_{2 \times 2}$
Matriz B	2x2	$I_{2 \times 2}$
Matriz Q	2x2	$0.01 \cdot I_{2 \times 2}$
Matriz R	2x2	$I_{2 \times 2}$
Matriz resultante K	2x2	$0.1 \cdot I_{2 \times 2}$

Figura 62: Trayectoria generada por marcadores PSO y controlador TUC-LQR



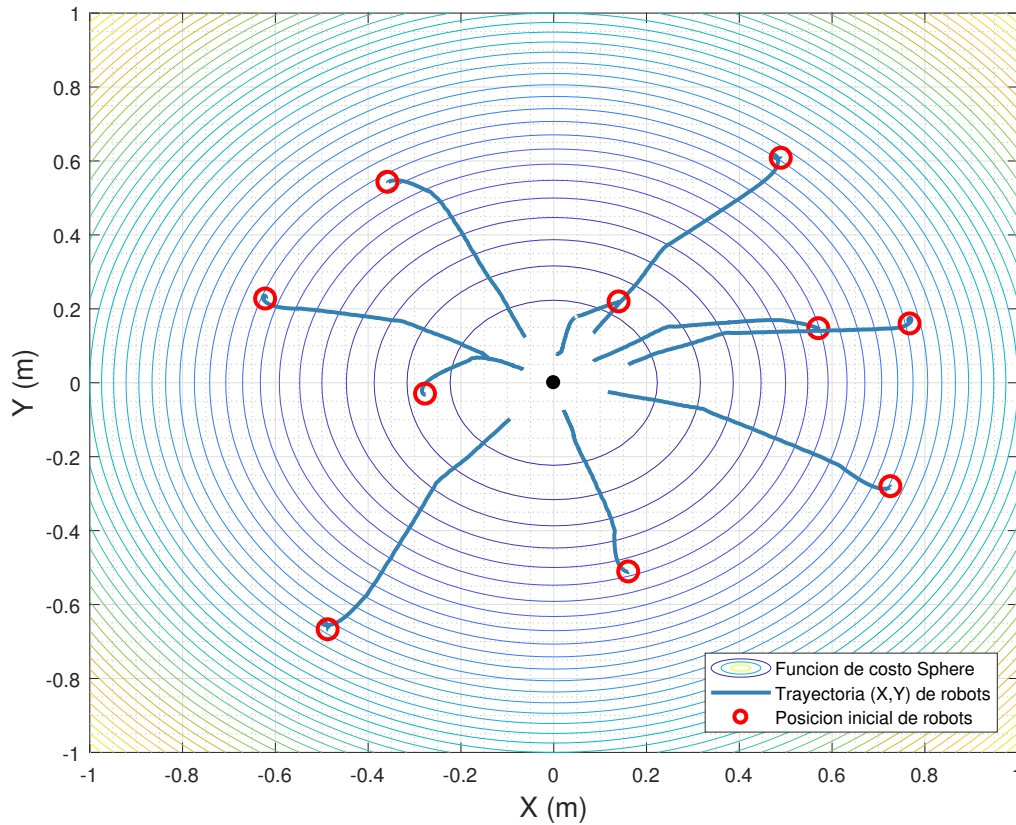
Al utilizar este controlador, se observaron trayectorias casi completamente rectas hacia la meta. Esto le proporcionó a los robots una mayor exactitud de convergencia hacia la meta al tener menores desviaciones por curvaturas muy grandes. Sin embargo, esto puede presentar complicaciones en aplicaciones en donde se posea un espacio de búsqueda con obstáculos que requiera un control que permita curvas más dinámicas de navegación como el control PID de la Sección 9.2. Se observa una considerable diferencia con los resultados del controlador de cinemática transformada presentado en la Sección 9.1 de esta investigación. En dicho controlador, se observó que la trayectoria de los robots sí presentaba cierta curvatura para llegar a la meta como se observa en la Figura 47.

Figura 63: Velocidad lineal y angular de robot con controlador TUC-LQR



En las figuras anteriores, se puede observar que el controlador TUC-LQR presentó un comportamiento similar al de los controladores SPC y LSPC. Este controlador se basa en la retroalimentación del error entre el marcador PSO de cada iteración y la posición del robot. En cada actualización de los marcadores PSO se presenta una aceleración y por lo tanto un pico de velocidad. Esto es lo que causa la irregularidad en las velocidades. Al igual que al SPC y LSPC, el planeador PSO se ejecutó cada 5 iteraciones del controlador para darle tiempo a los motores de reaccionar a las variaciones de velocidades. Las velocidades convergen a cero con el tiempo, dando así estabilidad al robot cuando este llega a la meta.

Figura 64: Trayectorias resultantes de enjambre con controlador TUC-LQR



En este caso, se observan trayectorias diferentes a las mostradas en la Figura 49. Se observa una curvatura mínima al inicio de las trayectorias de los robots en el momento en que estos se enderezan para apuntar hacia la meta. El resto de la trayectoria es un segmento casi recto. A pesar de que las trayectorias resultantes de este controlador son más directas por su casi inexistente curvatura, el tiempo de convergencia fue el mismo que el del SPC y LSPC (20 segundos). Esto se debió a la corta longitud de la distancia de cada marcador PSO. El beneficio de este corto escalamiento fue la reducción de picos bruscos en las velocidades, lo cuál fue más valioso sin importar el aumento en el tiempo de convergencia.

10.7. Controlador lineal cuadrático integral y uniciclo transformado (TUC-LQI)

Este controlador se basó en implementar el controlador de cinemática transformada presentado en la Sección 9.1 de esta investigación. La diferencia en este caso fue que las entradas u_1 y u_2 de la ecuación (30) ya no se plantearon de la misma manera. En lugar de utilizar la ecuación (31) para el cálculo de las entradas, se utilizó el análisis de variables de espacio estado presentado en las ecuaciones (70) y (71). El sistema LTI se planteó como (69) utilizando las matrices $\mathbf{A} = \mathbf{0}$, $\mathbf{B} = \mathbf{I}$, $\mathbf{C} = \mathbf{I}$, y la matriz $\bar{\mathbf{K}} = [\mathbf{K} \quad \mathbf{K}_I]$ se calculó con el mismo comando LQR de MATLAB utilizado en la sección anterior. Las entradas se calcularon como:

$$\mathbf{u} = -\mathbf{K} \cdot (1 - b_p) \cdot (\mathbf{x}_i - \mathbf{x}_{i+1}) - \mathbf{K}_I \cdot \mathbf{X}_i \quad (90)$$

$$\mathbf{X}_{i+1} = (1 - b_I) \cdot (\mathbf{X}_i + (\mathbf{p}_g - \mathbf{x}) \cdot \Delta t) \quad (91)$$

En este caso, se introdujo el parámetro b_p denominado amortiguamiento de control proporcional y el parámetro b_I que es el amortiguamiento del control integral. El primero, mientras tenga un valor más cercano a 1, reduce más los picos de velocidad presentes en cada actualización del marcador PSO. El segundo permite al robot diferencial llegar a la meta sin presentar las oscilaciones típicas de un control integral, ya que actúa como un freno del error integral acumulado. Se recomienda utilizar valores menores a 0.05 para este último parámetro. En (90), \mathbf{x}_i es la posición (x, y) actual del robot, \mathbf{x}_{i+1} es la posición del marcador PSO actual y \mathbf{X}_i es el error integral acumulado. En (91) se presenta la regla de actualización del error integral por medio de integración numérica. Este error integral se compone de la diferencia entre la mejor posición global encontrada hasta el momento por el enjambre \mathbf{p}_g y la posición actual del robot. De esta manera, se asegura que los robots no solo sigan la trayectoria definida por el planeador PSO, sino que lo hagan con velocidades suaves y continuas. Una aceleración continua hacia la mejor posición global reduce las aceleraciones causadas por la actualización de los marcadores PSO. Al aproximarse a la meta, el error integral deja de aumentar gradualmente, y gracias al amortiguamiento b_I este error no se mantiene constante en algún valor, sino que este decrece a cero conforme transcurre el tiempo. Esto asegura estabilidad de los robots en la meta.

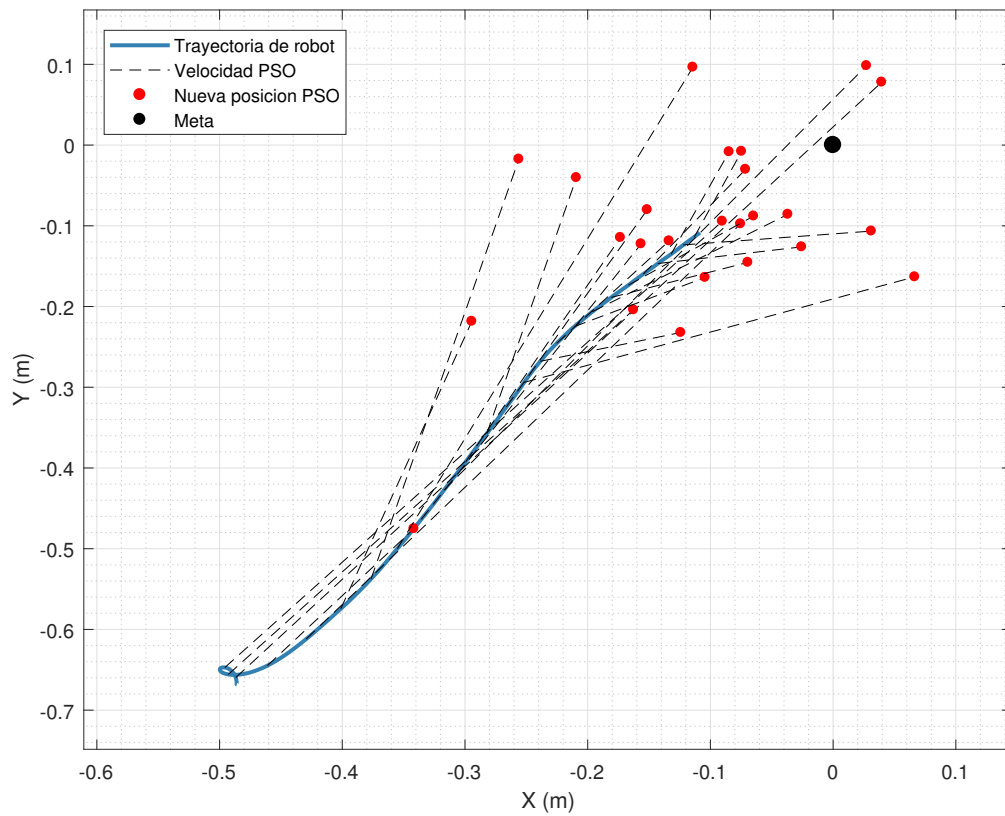
Cuadro 14: Matrices de ponderación utilizadas para cálculos de controlador TUC-LQI

Parámetros de simulación	Tamaño de matriz	Valor de matriz
Matriz Q	4x4	$I_{4 \times 4}$
Matriz R	2x2	$2000 \cdot I_{2 \times 2}$
Matriz resultante K	2x2	$0.2127 \cdot I_{2 \times 2}$
Matriz resultante K_I	2x2	$-0.0224 \cdot I_{2 \times 2}$

Cuadro 15: Parámetros utilizados de planeador PSO y de controlador TUC-LQI

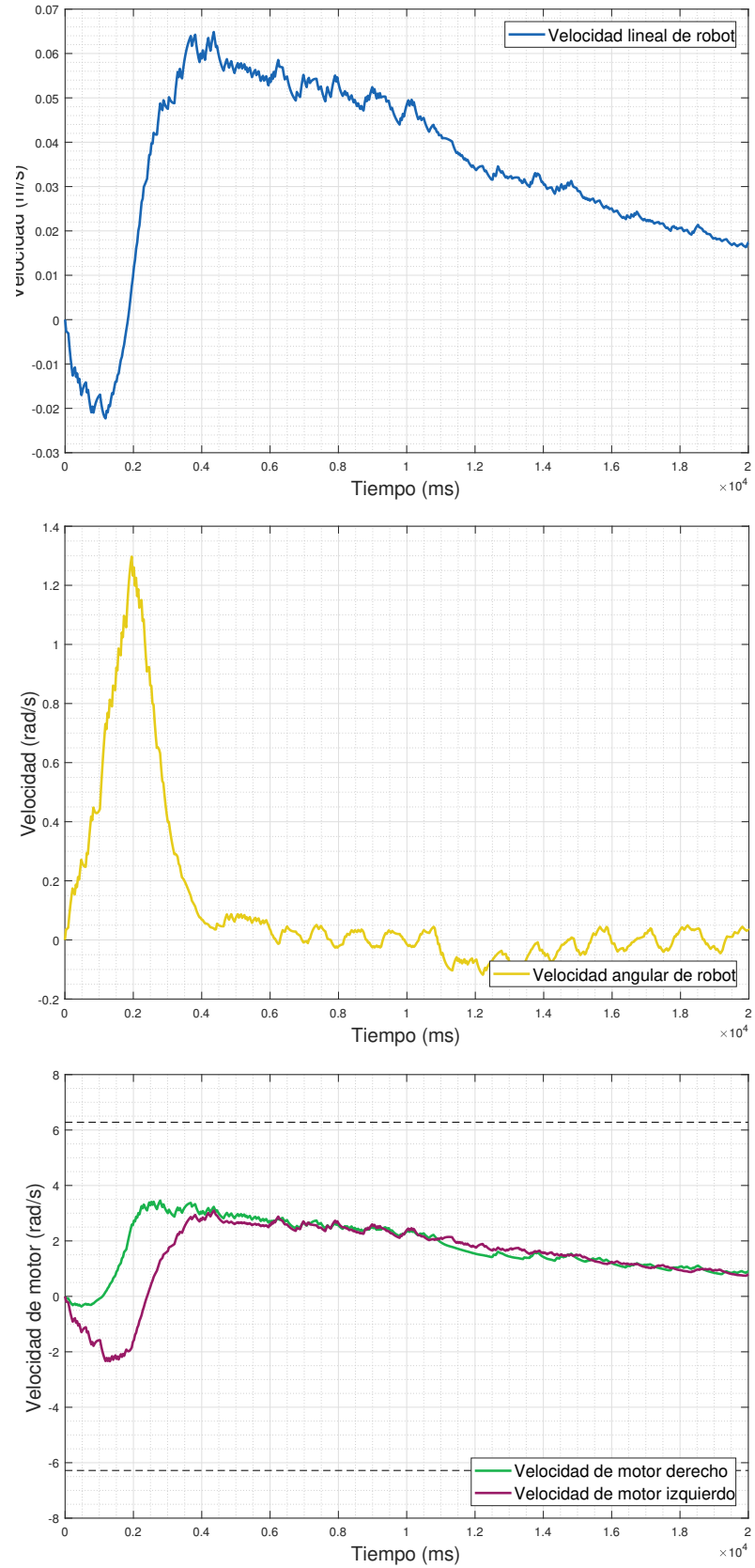
Parámetros de simulación	Valor de parámetro
T_s (ms)	32.000
η	0.032
P_s	1 iteración
b_p	0.95
b_I	0.01

Figura 65: Trayectoria generada por marcadores PSO y controlador TUC-LQI



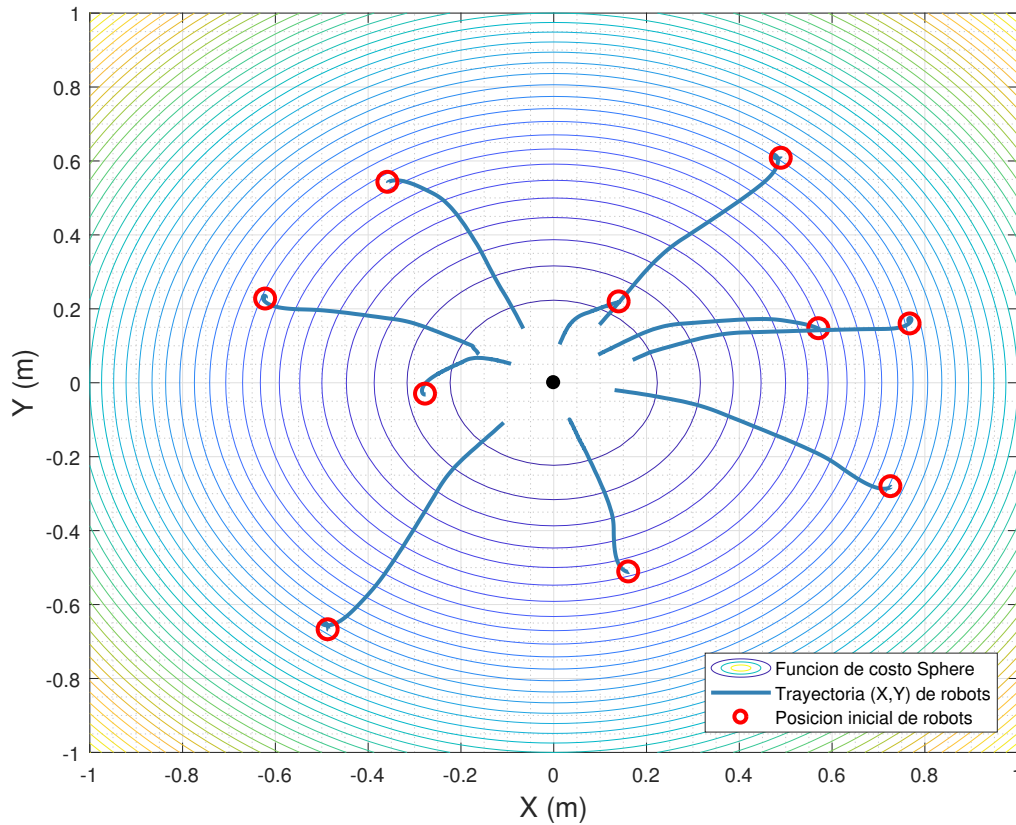
Al utilizar este controlador, se observaron trayectorias casi completamente rectas hacia la meta. Esto le proporcionó a los robots una mayor exactitud de convergencia hacia la meta al tener menores desviaciones por curvaturas muy grandes. Sin embargo, esto puede presentar complicaciones en aplicaciones en donde se posea un espacio de búsqueda con obstáculos que requiera un control que permita curvas más dinámicas de navegación como el control PID de la Sección 9.2. Estas trayectorias son casi idénticas, aunque ligeramente más suaves, a las trayectorias generadas por el TUC-LQR.

Figura 66: Velocidad lineal y angular de robot con controlador TUC-LQI



En las figuras anteriores, se puede observar que el controlador TUC-LQI resultó en las velocidades de motores más suaves comparadas con los resultados del resto de controladores. Esto incluso con planeador PSO ejecutándose en cada iteración del controlador. Se observa que los picos por las aceleraciones de las actualizaciones del marcador PSO casi desaparecieron gracias a la constante aceleración hacia la mejor posición global y al amortiguamiento del control proporcional. Asimismo, se puede observar que las velocidades de los motores tienden a cero gracias al amortiguamiento del control integral para darle estabilidad a los robots cuando estos lleguen a la meta.

Figura 67: Trayectorias resultantes de enjambre con controlador TUC-LQI



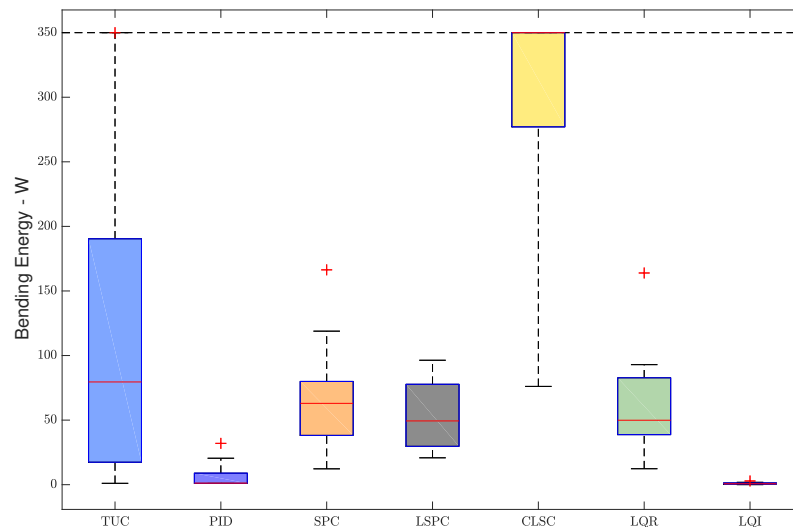
En este caso, se observan trayectorias diferentes a las mostradas en la Figura 49. Se observa una curvatura mínima al inicio de las trayectorias de los robots en el momento en que estos se enderezan para apuntar hacia la meta. El resto de la trayectoria es un segmento casi recto. El tiempo de convergencia fue el mismo que el del TUC-LQR (20 segundos). Esto se debió a la corta longitud de la distancia de cada marcador PSO. El beneficio de este corto escalamiento fue la reducción de picos bruscos en las velocidades, lo cuál fue más valioso sin importar el aumento en el tiempo de convergencia.

10.8. Comparación de desempeño de controladores

Luego de llevar a cabo todas las simulaciones necesarias para evaluar el rendimiento del MPSO en WeBots, se realizó la comparación para determinar cuales controladores poseían el mejor desempeño en cuanto a la generación de trayectorias suaves y convergentes hacia la meta, y generación de velocidades suaves que obtuvieran bajo porcentaje de saturación de los motores robóticos. En cuanto a la comparación de trayectorias, se observa en las Figuras 64 y 67 que los controladores TUC-LQR y TUC-LQI lograron generar las trayectorias más rectas hacia la meta, en comparación con los controladores SPC y LSPC que generaron trayectorias suaves pero con amplia curvatura (Figura 55 y 58). Los controladores TUC y TUC-PID generaron trayectorias más irregulares pero de igual manera convergentes hacia la meta, como se observa en las Figuras 49 y 52. El controlador TUC presentó la convergencia más rápida a los 10 segundos, seguido por el TUC-PID y el CLSC que convergieron a los 15 segundos. Por otro lado, los controladores SPC, LSPC, TUC-LQR y TUC-LQI obtuvieron la misma velocidad de convergencia (20 segundos).

Para evaluar el desempeño de los controladores en cuanto a suavidad de las curvas de control generadas, se utilizó el método de energía de deflexión de trazadores cúbicos basado en la teoría de vigas elásticas de Euler-Bernoulli [29]. Se realizó una simulación por cada controlador colocando los robots en diferentes posiciones y orientaciones respecto a la meta para observar la naturaleza del control al tener diferentes condiciones iniciales. Durante cada simulación, se recopilaron datos de velocidad de los actuadores derecho e izquierdo de cada uno de los 10 robots E-Puck utilizados en el mundo de Webots. Esto daba un total de 20 curvas por controlador a las cuales se le calculaba la energía de deformación W utilizando la ecuación (74). Los resultados se graficaron en formato de caja y bigote para observar la dispersión de los datos para cada controlador.

Figura 68: Suavidad de curvas de cada controlador calculada con energía de deformación

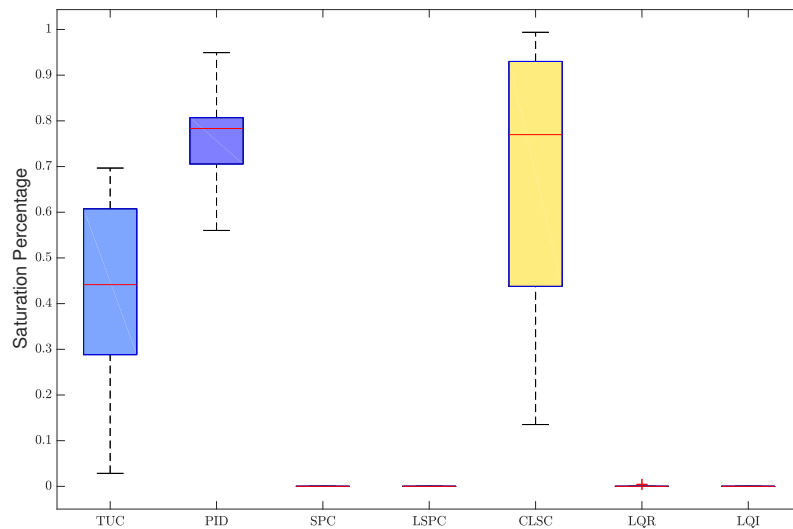


Como se observa en la Figura 68, el controlador que estadísticamente generó curvas con la mayor suavidad (menor energía de deflexión W) fue el TUC-LQI, concordando con lo observado en la Figura 66c. El segundo mejor controlador en cuanto a generación de

curvas suaves de velocidad fue el TUC-PID con un filtro de picos de velocidad acoplado. El controlador que estadísticamente presentó menor suavidad en las curvas de velocidad fue el CLSC, y esto concuerda con lo observado en la Figura 60c.

Otro aspecto a analizar en las curvas de velocidad generadas por cada controlador era el porcentaje de tiempo en que los actuadores de los robots se mantenían saturados. Para calcular esto, se determinó qué proporción de los datos totales de cada curva de velocidad específica presentaba saturación de los actuadores de los E-Puck (± 6.28 m/s).

Figura 69: Porcentaje de saturación de actuadores durante simulaciones



Como se observa en la Figura 69, los controladores que presentaron una mayor saturación de los actuadores fueron el TUC, el CLSC, y el TUC-PID. Estos dos últimos causaban que los motores de los robots se encontraran en promedio saturados aproximadamente el 80% del tiempo de la simulación. Al contrario, los controladores SPC, LSPC, TUC-LQR y TUC-LQI presentaron mínima saturación de los actuadores.

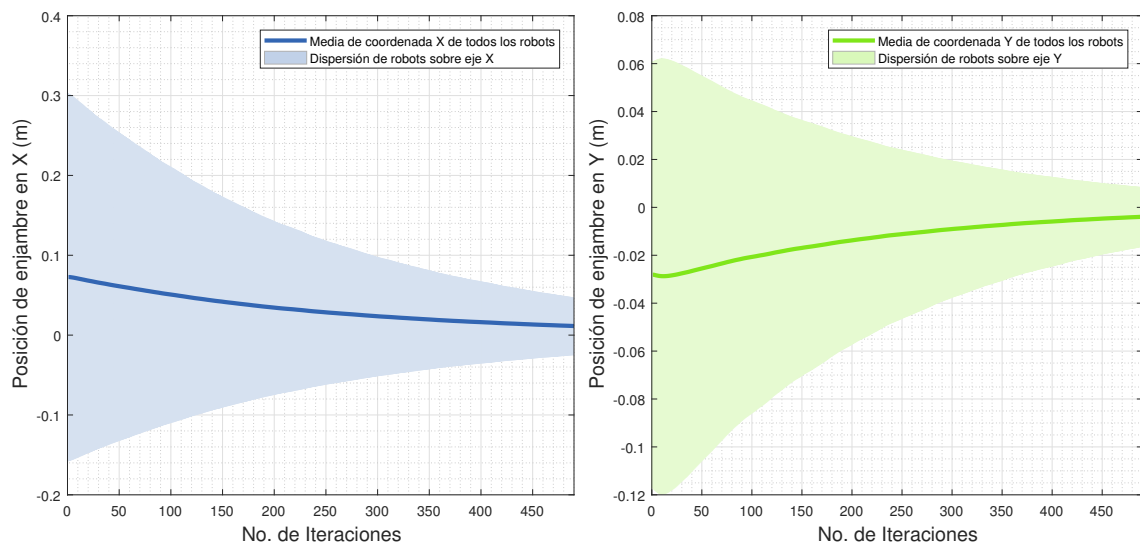
Cuadro 16: Resultados de controladores para algoritmo MPSO

Criterio evaluado	Mejor controlador	Peor controlador
Suavidad de trayectorias curvas	Control LSPC	Control CLSC
Trayectorias de menor curvatura	Control TUC-LQI	Control CLSC
Rapidez de convergencia	Control TUC	Control CLSC
Reducción de picos de velocidad	Control TUC-LQI	Control LSPC
Reducción de saturación	Control TUC-LQI	Control TUC-PID
Dinamicidad de movimiento	Control TUC-PID	Control TUC-LQR

Como se observa en el Cuadro 14, el controlador TUC-LQI es el que presenta mejores ventajas en cuanto a reducción de saturación y hard stops en los motores de los robots. Asimismo, logró generar las trayectorias más directas hacia la meta. A comparación con otros controladores, este es más rígido en cuanto a sus trayectorias, por lo que su implementación es adecuada en aplicaciones de búsqueda de metas en ambientes con ningún o pocos obs-

táculos. El segundo controlador con mejor desempeño fue el TUC-PID con el filtro acoplado. Este generó trayectorias irregulares, pero estas no influyeron en su tiempo de convergencia. Este controlador también fue el segundo control que mejor regulaba las velocidades en cuanto a suavidad de las curvas. Sin embargo, cabe notar que este fue el controlador que más saturación presentó en los motores de los robots. Por lo tanto, el uso de este controlador es recomendado para futuras aplicaciones de robótica de enjambre con el PSO, tomando en cuenta una limitación adecuada de los valores de saturación para no dañar los motores de los robots. Los controladores SPC y LSPC no presentaron buena regulación de velocidades, por lo que no se recomienda su uso para futuras aplicaciones. El CLSC no logró generar trayectorias adecuadas hacia la meta. Por lo tanto, como controlador final del algoritmo MPSO, se seleccionó el controlador de TUC-LQI, ya que genera las mejores trayectorias, presenta la mejor reducción de picos irregulares de velocidad y evita la saturación de los actuadores robóticos.

Figura 70: Dispersión (X,Y) de robots E-Puck en espacio de búsqueda



- El parámetro PSO de inercia ω elegido en MATLAB fue el natural exponencial, ya que este lograba que el enjambre obtuviera un comportamiento de convergencia rápida sin dejar a un lado una amplia exploración del espacio de tarea, como se muestra en la Figura 23.
- Los parámetros PSO de escalamiento elegidos en MATLAB fueron $c_1 = 2$ y $c_2 = 10$. Estos lograron dar una convergencia rápida al enjambre de robots y a la vez una amplia exploración del espacio de tarea sin fragmentar el enjambre por falsa convergencia en mínimos locales.
- El parámetro PSO de constricción φ elegido en MATLAB poseía un valor de 1.0, ya que este lograba que el enjambre obtuviera un comportamiento de convergencia hacia la meta al no causar que los pasos de actualización fueran muy grandes y el sistema divergiera.
- Los parámetros PSO de inercia y escalamiento elegidos en MATLAB fueron aplicables en el algoritmo MPSO que se implementó en la simulación de WeBots. El parámetro de constricción φ fue el único que se modificó para poder ser implementado en la simulación de los controladores en WeBots. Se le fijó un valor de 0.8 para reducir los pasos de la actualización para tomar en cuenta el tamaño físico de los robots.
- Los controladores del algoritmo MPSO acoplados al planeador PSO que permiten obtener las trayectorias curvas más suaves son el controlador de pose simple (SPC) y el controlador de pose con estabilidad Lyapunov (LSPC). Sin embargo, estos presentan curvas agresivas de velocidades, lo cual puede dañar los actuadores de los robots. Por otro lado, el controlador TUC-PID con filtro acoplado logró suavizar las curvas de velocidades generadas. Sin embargo, presentó alto porcentaje de saturación en los motores robóticos.
- El controlador TUC-LQI, fue el que mejor desempeño obtuvo en cuanto a la generación de trayectorias directas a la meta. También logró eliminar casi cualquier irregularidad o variación brusca en las velocidades de los motores y redujo la saturación de los

actuadores a cero. Por lo tanto, este controlador es el ideal para implementar el algoritmo PSO en robots diferenciales como los E-Puck. El controlador TUC-LQR obtuvo trayectorias similares, pero sus velocidades generadas no fueron las más adecuadas.

- En fases de implementación a robots reales, se recomienda utilizar una estructura cilíndrica para el cuerpo de los robots y las mismas medidas del robot E-Puck para que los controladores diseñados en esta investigación sean aplicables directamente.
- Se recomienda experimentar con simulaciones de WeBots utilizando enjambres de una mayor cantidad de agentes para observar el comportamiento del sistema con los controladores diseñados.
- En fases de implementación futuras, se recomienda probar el sistema de rastreo por visión de computadora diseñado en la Fase I del Proyecto Robotat en conjunto con los algoritmos de control desarrollados en esta investigación.
- Se recomienda solicitar con tiempo los componentes que se deseen utilizar en caso se desee realizar una implementación del algoritmo MPSO en robots reales.

-
- [1] A. Rodas, “Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre”, UVG, 2019, págs. 11-51.
 - [2] M. Castillo, “Diseñar e implementar una red de comunicación inalámbrica para la experimentación en robótica de enjambre”, UVG, 2019, págs. 22-42.
 - [3] Y. Shi y R. Eberhart, “A modified particle swarm optimizer”, en *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, IEEE, 1998, págs. 69-73.
 - [4] J. C. Bansal, P. Singh, M. Saraswat, A. Verma, S. S. Jadon y A. Abraham, “Inertia weight strategies in particle swarm optimization”, en *2011 Third world congress on nature and biologically inspired computing*, IEEE, 2011, págs. 633-640.
 - [5] S. Konduri, E. O. C. Torres y P. R. Pagilla, “Dynamics and control of a differential drive robot with wheel slip: application to coordination of multiple robots”, *Journal of Dynamic Systems, Measurement, and Control*, vol. 139, n.º 1, pág. 014505, 2017.
 - [6] R. Grandi, R. Falconi y C. Melchiorri, “A navigation strategy for multi-robot systems based on particle swarm optimization techniques”, *IFAC Proceedings Volumes*, vol. 45, n.º 22, págs. 331-336, 2012.
 - [7] M. Egerstedt, *Georgia Tech Robotarium*, <http://www.robotics.gatech.edu/robotarium>, Accessed: 2019-04-29.
 - [8] M. Bonani, *EPFL E-Puck robot*, <http://www.e-puck.org/>, Accessed: 2019-04-29.
 - [9] J. Kennedy y R. Eberhart, “Particle Swarm Optimization”, IEEE, 1995, págs. 1-7.
 - [10] J. Carr, “An introduction to genetic algorithms”, *Senior Project*, vol. 1, n.º 40, pág. 7, 2014.
 - [11] A. Kaveh, *Advances in metaheuristic algorithms for optimal design of structures*. Springer, 2014.
 - [12] T. Beielstein, K. E. Parsopoulos y M. N. Vrahatis, *Tuning PSO parameters through sensitivity analysis*. Universitätsbibliothek Dortmund, 2002.

- [13] D. Wang, D. Tan y L. Liu, “Particle swarm optimization algorithm: an overview”, *Soft Computing*, vol. 22, n.º 2, págs. 387-408, 2018.
- [14] R. C. Eberhart e Y. Shi, “Comparing inertia weights and constriction factors in particle swarm optimization”, en *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, IEEE, vol. 1, 2000, págs. 84-88.
- [15] D. Bingham, *Project Homepage DEAP 1.3.0 documentation*, <https://deap.readthedocs.io/en/master/api/benchmarks.html#deap.benchmarks.sphere>, Accessed: 2019-04-29.
- [16] J. M. Dieterich y B. Hartke, “Empirical review of standard benchmark functions using evolutionary global optimization”, *arXiv preprint arXiv:1207.4318*, 2012.
- [17] R. Siegwart, I. R. Nourbakhsh y D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.
- [18] K. M. Lynch y F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.
- [19] K. Shojaei, A. M. Shahri, A. Tarakameh y B. Tabibian, “Adaptive trajectory tracking control of a differential drive wheeled mobile robot”, *Robotica*, vol. 29, n.º 3, págs. 391-402, 2011.
- [20] F. Martins y A. Brandão, “Velocity-based Dynamic Model and Control”, 2018.
- [21] M. Egerstedt, *Control of Mobile Robots, Introduction to Controls*. Georgia Institute of Technology, 2014.
- [22] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. Springer, 2017, vol. 118.
- [23] S. K. Malu y J. Majumdar, “Kinematics, localization and control of differential drive mobile robot”, *Global Journal of Research In Engineering*, 2014.
- [24] R. M. Murray, *A mathematical introduction to robotic manipulation*. CRC press, 2017.
- [25] M. Hou, G. Duan y M. Guo, “New versions of Barbalat’s lemma with applications”, *Journal of Control Theory and Applications*, vol. 8, n.º 4, págs. 545-547, 2010.
- [26] J. J. Park y B. Kuipers, “A smooth control law for graceful motion of differential wheeled mobile robots in 2d environment”, en *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, págs. 4896-4902.
- [27] J. P. Hespanha, *Linear systems theory*. Princeton university press, 2018.
- [28] T. K. Priyambodo, “Integral Modified Linear Quadratic Regulator Method for Controlling Lateral Movement of Flying Wing in Rotational Roll Mode”, en *Journal of Engineering and Applied Sciences*, IEEE, 2018, págs. 463-471.
- [29] G. Wolberg e I. Alfy, “An energy-minimization framework for monotonic cubic spline interpolation”, *Journal of Computational and Applied Mathematics*, vol. 143, n.º 2, págs. 145-188, 2002.

14.1. Código y simulación utilizado en la investigación

Para descargar el código asociado a esta investigación, puede ingresar al siguiente link: <https://github.com/AldoAguilar001/MPSO-Algorithm.git>.

Aquí encontrará el archivo **Simulacion1.wbt** que contiene el mundo simulado de Webots y que puede cargar en Webots R2019a siguiendo los siguientes pasos:

- Descargue el archivo **Simulacion1.wbt** de Github
- Descargue la plataforma de Webots en <https://cyberbotics.com/download>
- En el IDE de Webots, diríjase a la barra de herramientas y presione File
- Ahora presione Open World
- Elija el archivo descargado de Github para abrir el mundo

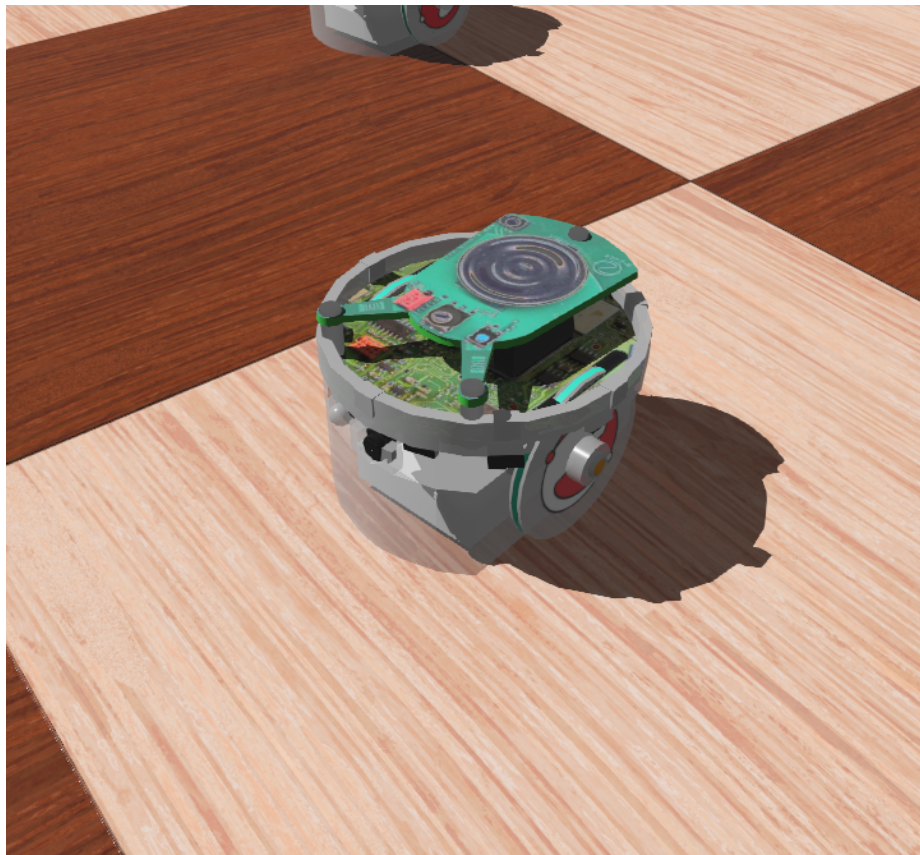
También encontrará el archivo **e-puck_MPSO.c** que contiene el código para implementar los controladores utilizados en esta investigación. Para implementar dicho archivo de controlador al mundo simulado cargado en Webots, siga los siguientes pasos:

- Descargue el archivo **e-puck_MPSO.c** de Github
- En el IDE de Webots, diríjase a la barra de herramientas y presione Wizards
- Elija New Robot Controller
- Presione Siguiente y seleccione lenguaje de programación C
- Elija Webots (gcc Makefile) como compilador del IDE

- Nombre el nuevo controlador como “e-puck_MPSO.c”
- Ya creado el nuevo controlador, copie y pegue el programa del archivo descargado de Github al script del controlador creado en Webots
- Para compilar y cargar controlador a los robots E-Puck de la simulación presione F7 y elija Reload world
- Al presionar el botón de Play inicia la simulación y la puede detener con Pausa
- Para reiniciar simulación, presione Reload world

14.2. Representación de agentes en simulación de Webots

Figura 71: Simulación de robot E-Puck dentro de mundo Webots



14.3. Pseudocódigo de implementación digital de control PID

Implementación de controlador PID digital:

$$e_{k-1} = 0$$

$$E_k = 0$$

...

$$e_k = r - y$$

$$e_d = e_k - e_{k-1}$$

$$E_k = E_k + e_k$$

$$u_k = K_p \cdot e_k + K_i \cdot E_k + K_d \cdot e_d$$

$$e_{k-1} = e_k$$

El controlador PID digital consiste en:

- Ejecutar la resta entre las variables de la señal de referencia del sistema y el estado actual del sistema. El resultado se asigna a la variable de error actual.
- Calcular el error diferencial por medio de la resta entre el error actual y el error de la iteración pasada (si se trata de la primera iteración, el error pasado es inicializado en cero).
- Calcular el error integral por medio de la suma entre el error actual al error integral acumulado en todas las iteraciones (también se inicializa en cero).
- Calcular la salida de control u_k sumando de todos los errores multiplicados por sus respectivos coeficientes (estos absorben el Δt del período de muestreo del programa).
- Actualizar el valor de error de iteración pasada con el valor de error actual.

14.4. Capturas de pantalla de simulaciones en MATLAB

Figura 72: Simulación con función de costo Rosenbrock

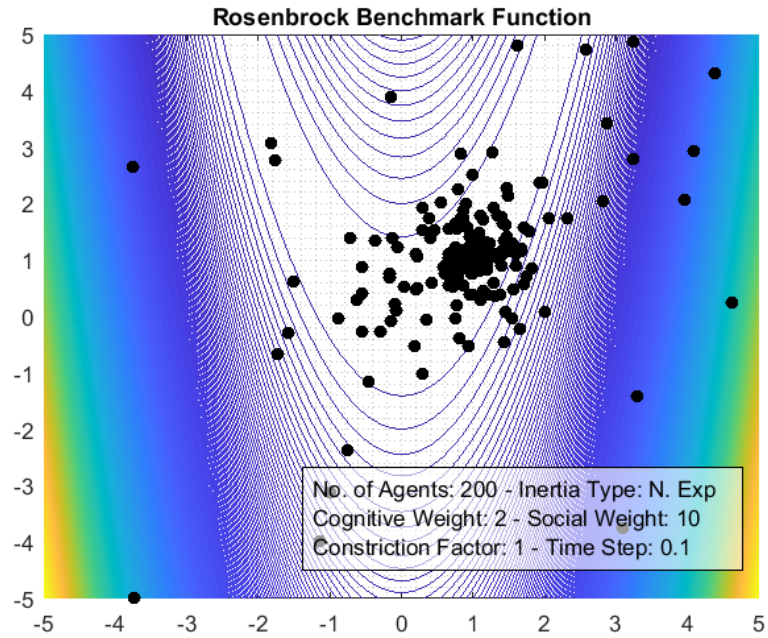


Figura 73: Simulación con función de costo Sphere

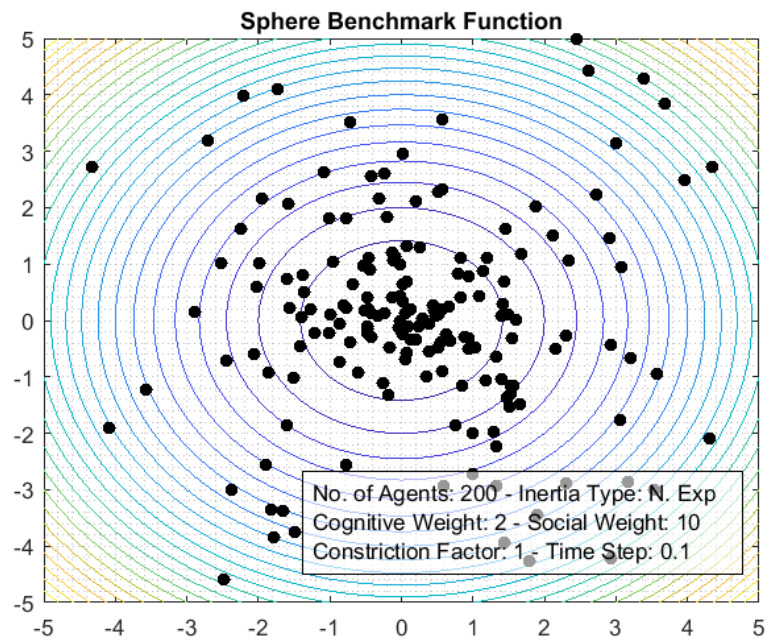


Figura 74: Simulación con función de costo Booth

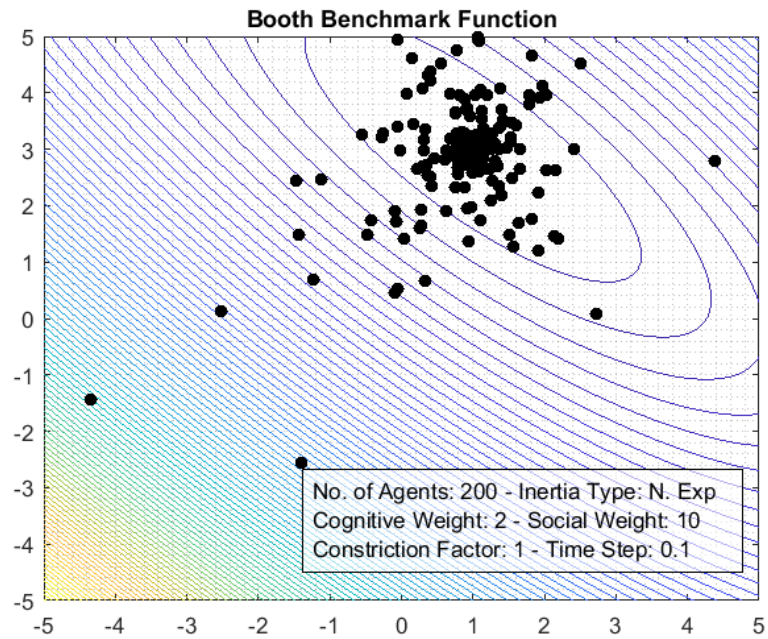


Figura 75: Simulación con función de costo Himmelblau

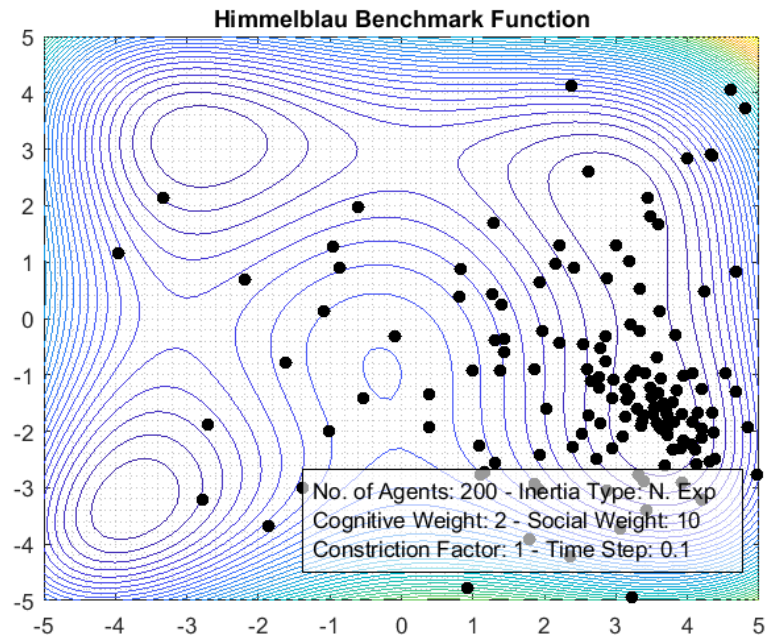


Figura 76: Simulación con función de costo Mínimos Múltiples

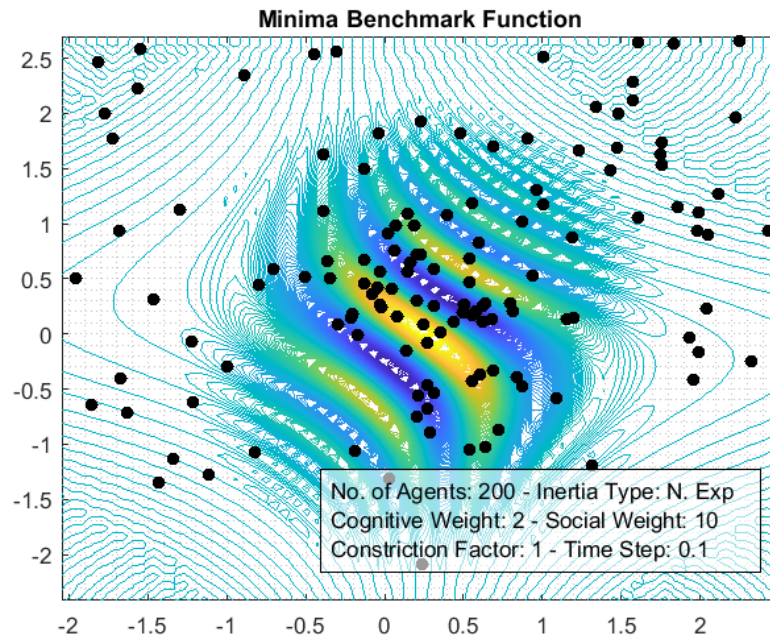


Figura 77: Simulación con función de costo Schaffer F6

