Implementación de Robótica de Enjambre en Simuladores con Restricciones Físicas

Implementation of Swarm Robotics in Simulators with Physical Constraints

Marco Izeppi (ize17229@uvg.edu.gt)

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

Resumen

El algoritmo de optimización por enjambre de partículas (Particle Swarm Optimization - PSO) hace referencia a un método heurístico para resolver un tipo de problema computacional que evoca el comportamiento de las partículas en la naturaleza. La idea clásica de este comportamiento se centra en comportamientos de partículas sin masa o dimensiones físicas a diferencia de los robots que se utilizan para la exploración. Tomando esto en consideración, en las fases previas a este proyecto se desarrolló un algoritmo que utiliza métodos de Aprendizaje automático como lo es el aprendizaje profundo y el aprendizaje reforzado con la finalidad de crear un Toolbox en Matlab que optimice la selección de los parámetros que ingresan al sistema para mejorar la velocidad de convergencia de las partículas hacia el mínimo global. Se implemento el algoritmo, previamente diseñado, en un entorno de simulación con restricciones físicas, tomando en cuenta las dimensiones del robot, capaz de evadir obstáculos ubicados dentro del entorno de simulación de forma aleatoria. Este algoritmo busca una rápida convergencia hacia el mínimo global evitando colisiones entre los robots o con los obstáculos. Por otro lado, se determinó que es viable la realización de simulaciones de sistemas multi-robot en el entorno de Robotic Operating System (ROS) brindando muchas herramientas que pueden facilitar el mapeo, navegación, entendimiento, etc.

Palabras clave: multi-robot, restricciones físicas, evasión de obstáculos, robótica de enjambre, controladores.

Abstract

The particle swarm optimization (PSO) algorithm refers to a heuristic method to solve a type of computational problem that evokes the behavior of particles in nature. The classic idea of this behavior focuses on particle behaviors without mass or physical dimensions as opposed to robots that are used for exploration. Taking this into consideration, in the phases prior to this project an algorithm was developed that uses machine learning methods such as deep learning and reinforced learning to create a Toolbox in Matlab that optimizes the selection of the input parameters of the system to improve the speed of convergence of the particles towards the global minimum.

The previously designed algorithm was implemented in a simulation environment with physical restrictions, considering the dimensions of the robot, being able of avoiding obstacles randomly located inside the simulation environment. This algorithm seeks a rapid convergence towards the global minimum avoiding collisions between the robots or with obstacles. On the other hand, it was determined that it is feasible to carry out multi-robot simulations in the Robotic Operating System (ROS) environment, providing many tools that can facilitate mapping, navigation, understanding, etc.

Keywords: multi-robot, physical restraints, obstacle avoidance, swarm robotics, controllers.

Introducción

El algoritmo PSO posee diversas variantes aplicadas a diversos objetivos tales como optimización multiobjetivo, PSO binaria, PSO discreta, PSO combinatoria, etc. Tomando esto en consideración, dentro de la Universidad del valle de Guatemala se inició con una variante aplicada a robots móviles con el mega proyecto Robotat en donde el algoritmo formaba parte del sistema de navegación que utilizarían los robots diferenciales para converger a un punto específico (Castillo, 2019).

Con anterioridad, Aldo Aguilar en su tesis (Aguilar, 2019) buscó la implementación de un algoritmo modificado de optimización de enjambre de partículas utilizando robots diferenciales reales. El objetivo principal de esta tesis era definir un algoritmo que estableciera el comportamiento de enjambre en robots diferenciales con el objetivo de buscar y llegar a una meta en común. Uno de los principales problemas que se presentaron era el acople directo del movimiento de las partículas del PSO debido a que el movimiento era irregular generando la posibilidad de saturación en los actuadores del robot. Para solucionar este problema se planteó que cada robot no iría directamente a la posición dada por el PSO, sino que se utilizaría la posición dada por el PSO como una sugerencia del punto al cual debía desplazarse el robot.

Tomando el modelo realizado por Aldo Aguilar, Juan Cahueque en su tesis (Cahueque, 2019) realizó una implementación de la teoría de enjambre en el área de búsqueda y rescate de personas. Sabiendo que el movimiento de partículas bidimensional se realiza en una superficie tridimensional, siendo la tercera componente el resultado de la función de costo que nos indica cual es el mínimo global, podemos determinar hacia qué punto convergerán las partículas. definió funciones artificiales de potencia con comportamiento multiplicativo y aditivo. Lo cual nos brinda la posibilidad de modelar una función de costo "personalizada" para establecer a que punto convergerán las partículas y hacer que eviten los obstáculos dentro del entorno.

Realizando diversas simulaciones se llegó a determinar los mejores parámetros para los controladores en los robots diferenciales físicos. Sin embargo, el problema que este algoritmo presentaba en lo que respecta a evasión de obstáculos era que se debía tener un conocimiento previo del entorno para poder generar la función de costo.

A partir de esto, Eduardo Santizo en su tesis (Santizo, 2021) empleó la metodología de aprendizaje reforzado y aprendizaje profundo para desarrollar un PSO tuner (afinador de parámetros para el algoritmo PSO) y un generador de trayectorias. Para el desarrollo del PSO tuner, se empleó una red neuronal que toma diversas métricas propias de las partículas PSO y realiza una predicción de los valores óptimos para los parámetros relevantes del algoritmo. Entrenando las redes neuronales con simulaciones de un algoritmo estándar de optimización de enjambre se logró que el algoritmo final redujera el tiempo de convergencia y susceptibilidad a mínimos locales en comparación con el PSO original.

Para el generador de trayectorias se utilizó aprendizaje profundo para condicionar el movimiento del robot dentro de un entorno de simulación en Matlab. El robot solo era capaz de moverse hacia arriba, abajo, izquierda o derecha. Al robot se le daban penalizaciones si colisionaba con un obstáculo y se le premiaba si se movía esquivando el obstáculo. Si el robot llegaba a la meta rápidamente recibía un premio aún mayor. Usando este sistema de premios y penalizaciones, se entrenó al algoritmo para predecir de mejor manera la trayectoria.

Cabe destacar que la propuesta de Eduardo Santizo se enfocaba directamente en los algoritmos de aprendizaje reforzado y aprendizaje profundo por lo cual no se llegaron a realizar simulaciones en entornos con restricciones físicas como lo son los simuladores de Webots y ROS. Por lo tanto, se propuso la integración de las herramientas de software PSO Tuner y el Swarm Robotics Toolbox a plataformas de simulación con restricciones físicas.

Webots es una Multi-plataforma Open-Source (Código abierto) utilizada para realizar simulaciones de robots en entornos más cercanos a la realidad. Provee un entorno completo de desarrollo para modelar, programar y simular robots. El programa permite utilizar controladores escritos en C, C++, Java, Python, y MATLAB. Por otro lado, el entorno de ROS es un entorno mas industrial con mayor aplicación en el campo de la robótica y con diversos módulos que realizan diversas acciones dependiendo cual sea el enfoque. ROS cuenta con una gran capacidad para modelar robots de cualquier tipo utilizando archivos URDF y simular condiciones físicas utilizando el motor de simulación de Gazebo. Gazebo utiliza múltiples motores físicos tales como ODE y Bullet para simular los efectos de iluminación, gravedad, inercia, etc.

Materiales y métodos

Desarrollo de algoritmo PSO

La Optimización por Enjambres de Partículas (PSO) es una técnica de optimización/búsqueda, y se inspira en el comportamiento de los enjambres de insectos en la naturaleza. El objetivo de la optimización es encontrar valores de x y y para los que la función $f(x, y)$ sea máxima (o mínima). La idea principal dentro del algoritmo en PSO empieza situando partículas al azar en el espacio de búsqueda, pero dándoles la posibilidad de que se muevan a través de él de acuerdo con unas reglas que tienen en cuenta el conocimiento personal de cada partícula y el conocimiento global del enjambre (Álvarez et al., 2009). Esto se puede describir matemáticamente de la siguiente forma:

$$v_i(t+1) = v_i(t) + c_1 r_1 \left(p_i^{mejor} - p_i(t)\right) + c_2 r_2 (p_g^{mejor} - p_i(t))$$

En donde:

- v_i = la velocidad de la partícula.
- p_i y p_i^{mejor} = es la posición actual de la partícula y la mejor posición por la cual ha pasado dicha partícula.
- p_g^{mejor} = es la mejor posición global de todo el sistema de enjambre.
- c_1 y c_2 = son las constantes de atracción al mejor personal y el mejor global respectivamente.
- r_1 y r_2 = son números aleatorios entre 0 y 1.

Una vez actualizadas las velocidades de todas las partículas utilizando la ecuación anterior, se actualizan sus posiciones siguiendo una ley simple:

$$p_i(t+1) = v_i(t) + P_i(t)$$

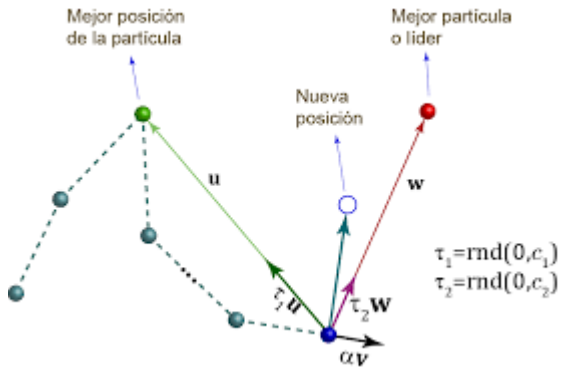

Figura 1. Representación gráfica del comportamiento del algoritmo PSO (Chavarría and Fallas, 2019).

Para el desarrollo del algoritmo PSO dentro del entorno de simulación de Webots se utilizó la posición de los robots diferenciales obtenida por un nodo de GPS que nos brinda la posición y la orientación de los robots en cada momento. A partir de esto, se obtuvieron los datos de la posición del robot para comparar la mejor posición que ha tenido el robot, la mejor posición que ha existido en todo el enjambre y la posición actual del mismo y así calcular la velocidad actualizada de cada uno de los robots (Chi et al., 2010).

Para poder compartir la mejor posición que ha existido en todo el enjambre, se utiliza un nodo transmisor y un nodo receptor para cada uno de los robots. El nodo emisor se utiliza para modelar emisores de radio, serie o infrarrojos. Este tipo de nodos solo es capaz de enviar información, pero no de recibirla. Por otro lado, un nodo receptor también se utiliza para modelar receptores de radio, serie o infrarrojos, pero a diferencia del emisor, este solo es capaz de recibir información. Ambos nodos poseen diversas configuraciones, en este caso nos centraremos en las configuraciones que nos importan para lograr la transmisión de los datos (Álvarez et al., 2009).

En este caso, nos interesa transmitir la mejor posición del robot y verificar si los otros robots han tenido una mejor posición. Para esto, es necesario que cada robot guarde la mejor posición en ambos ejes y el valor del mejor resultado. Cabe destacar que cada robot calcula su propia función del PSO y se comparten esta información para determinar el mejor global. La mejor posición del robot y la posición siguiente son calculadas internamente, mientras que la mejor posición global si es compartida entre todos los robots utilizando los nodos emisores y receptores dentro de cada robot (Chi et al., 2010).

Las constantes del algoritmo fueron obtenidas del PSO tuner dentro del cual se calculan los valores de c_1 y c_2 óptimos para el algoritmo PSO generando una convergencia más rápida por parte del algoritmo hacia el mínimo de la función de costo.

Algoritmo de Braitenberg

El Vehículo de Braitenberg tiene dos ruedas actuadas por motores independientes y varios sensores de luz o distancia. A mayor intensidad de luz o menor distancia, mayor par aplicado a las ruedas y en consecuencia se genera un movimiento de las ruedas a mayor velocidad angular. Este tipo de vehículos se guían por instintos, entendiendo que un instinto se corresponde con un sistema de valores de intensidades variables brindado por los sensores, es decir un sistema de motivaciones y gustos (Braitenberg, 1986).

Suponiendo un robot que funciona con sensores de luz, este puede realizar un movimiento circular alrededor de una fuente de luz externa si la función de activación tiene un máximo en algún punto, es decir si funciona con una polaridad positiva (figura 2a) para valores de intensidad de la luz bajas y con polaridad negativa (figura 2b) para intensidades elevadas. Si la conexión es contra lateral, se acercará a la fuente de luz cuando esté lejos de ella, y se alejará de ella cuando esté más próximo (Braitenberg, 1986).

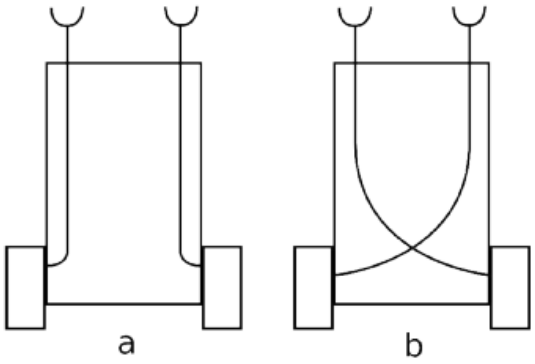

Figura 2. Diagrama de braitenberg (Braitenberg, 1986).

Si el anterior robot dispusiese además de otros sensores con funciones de activación diversas, el conjunto de comportamientos que podrían realizar sería muy variado, hasta el punto de que resultaría difícil, si no imposible, para un observador externo deducir esta variedad de comportamientos a partir del análisis de los elementos del robot. La dificultad de segmentar los comportamientos mediante técnicas analíticas es una de las características principales de los vehículos de Braitenberg (Braitenberg, 1986).

Para el algoritmo de vehículos de braitenberg se utilizó robótica basada en comportamiento para generar el movimiento de evasión. La robótica basada en comportamiento es un enfoque de la robótica que se centra en aquellos robots capaces de exhibir comportamientos de apariencia compleja a pesar de pequeñas variables del estado interno para modelar su entorno, sobre todo corrigiendo gradualmente sus acciones a través de enlaces sensoriomotores. Por lo tanto, mientras el robot no detecte ningún obstáculo cercano a él, seguirá moviéndose en la dirección del punto que sea nuestra meta.

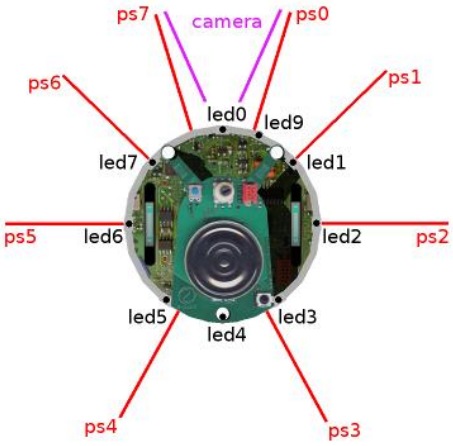

Figura 3. Ubicación de los sensores del robot e-puck (Cyberbotics Ltd, 2021)

Como se puede observar en la figura 3, el robot e-puck cuenta con 8 sensores de distancia. Tomando en consideración que el giro del robot depende de la velocidad de giro de las ruedas (velocidad angular). Se puede establecer que si $V_r > V_l$ entonces se realizara un giro a la izquierda y si en caso contrario, $V_l > V_r$ se realizara un giro a la derecha, por último, si $V_r = V_l$ el robot se moverá hacia adelante o hacia atrás. Bajo esta definición, se puede establecer el comportamiento del robot como:

$$\|V_r\| = \|V_r\|_0 + \sum_{k=1}^{2} \frac{\alpha_{1k}(\|s_K\|)}{\rho}$$

$$\|V_l\| = \|V_l\|_0 + \sum_{k=1}^{2} \frac{\alpha_{2k}(\|s_K\|)}{\rho}$$

En donde:

- $\|V_r\|_0$ y $\|V_l\|_0$ = corresponden a velocidades iniciales constantes para la rueda derecha e izquierda respectivamente.
- ρ = rango de los sensores que se estén utilizando (generalmente se normaliza el valor).
- α = impacto del valor de cada uno de los sensores sobre cada motor.
- s_K = corresponde a el valor medido por los sensores.

Teniendo esto definido, se puede emplear producto matricial para encontrar la forma compacta del controlador de braitenberg, finalmente la ecuación queda de la siguiente manera:

$$\begin{bmatrix} \|V_r\| \\ \|V_l\| \end{bmatrix} = \begin{bmatrix} \|V_r\|_0 \\ \|V_l\|_0 \end{bmatrix} + A * s$$

En donde:

- $matriz\ A$ = impacto del valor de cada uno de los sensores sobre cada motor.
- s = corresponde a el valor medido por los sensores dividido por su rango ($\frac{s_K}{\rho}$)

Asumiendo las posiciones de los sensores con respecto a la figura 3, se puede observar que los sensores traseros no se utilizan, debido a que se busca que el robot avance hasta una meta en específico. Los sensores laterales tendrán un impacto mínimo ya que no queremos que el robot gire bruscamente al detectar algo en estos sensores, mas bien queremos que mantenga una dirección paralela al obstáculo. Para los sensores frontales, se requiere que los mas cercanos al centro presenten un mayor impacto en el giro ya que se quiere evitar la colisión del robot con los obstáculos u otros robots. Por lo que finalmente, tomando esto en consideración, se define la matriz A como:

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{21} & \cdots \\ \alpha_{12} & \alpha_{22} & \ldots \end{bmatrix} = \begin{bmatrix} 0.95 & 0.80 & 0.40 & 0 & 0 & -0.20 & -0.35 & -0.45 \\ -0.50 & -0.40 & -0.20 & 0 & 0 & 0.40 & 0.75 & 0.90 \end{bmatrix}$$

Controlador PID

Un control PID lee un valor de entrada o del sensor de entrada, aplica los algoritmos de control, definidos y produce una salida específica como señal actuante o como entrada a un actuador. La salida del sistema es medida por el sensor de entrada, y el proceso se repite indefinidamente (Lorandi et al., 2011). El controlador PID clásico tiene la forma:

$$u = k_p \left(e + \frac{1}{T_i} \int_{x=0}^{t} e dt + T_d \frac{de}{dt} \right)$$

En este tipo de controlador, la ganancia K_p que afecta el componente proporcional, reduce el tiempo de crecimiento y elimina parte del error estacionario, $K_i = \frac{1}{T_i}$ que afecta el componente integral, elimina el error en estado estacionario, pero puede afectar de la respuesta transitoria y $K_d = T_d$ que afecta el componente derivativo, reduce el sobre pico y mejora la respuesta transitoria (Lorandi et al., 2011).

Para los parámetros de el sistema dentro de la implementación en webots se tomaron los parámetros establecidos en el Swarm Robotics Toolbox en el cual los parámetros son:

$$K_p = 0.5 \qquad K_i = 0.1 \qquad K_d = 0.001$$

Regulador cuadrático lineal (LQR)

El regulador cuadrático lineal es un controlador por retroalimentación que busca llevar al sistema a un estado determinado de forma rápida y reduciendo la cantidad de control utilizado. El controlador LQR debe utilizar un proceso de optimización que determine el desempeño de este (Beauchamp and Batista, 2016). la función que se utiliza es el índice de desempeño dada por:

$$J = \int_{t_0}^{\infty} (x^T Q x + u^T R u)\, dt$$

Si se considera que la ecuación diferencial de Riccati tiene una solución en estado estacionario, el problema ahora se convierte en un problema de horizonte en infinito en donde la ecuación diferencial de Riccati (DRE) se torna en una ecuación algebraica de Riccati (ARE) (Beauchamp and Batista, 2016). Finalmente, la acción de control para el controlador LQR con horizonte en infinito en sistema a lazo cerrado es:

$$\dot{x}(t) = Ax(t) - BKx(t)$$

En donde:

- Matriz A = matriz de estado del sistema
- Matriz B = matriz de parámetros de entrada del sistema
- Matriz K = matriz de parámetros encontrados para el controlador LQR

Por lo tanto, luego de realizar el análisis, se determina que los valores para la matriz K son:

$$K_{lqr} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Integrador cuadrático lineal (LQI)

Básicamente es similar al LQR, sin embargo, en este controlador, la ley de control se expresa de la siguiente forma:

$$u = -K\begin{bmatrix} x \\ x_i \end{bmatrix}$$

En donde x_i es la salida del integrador. Esta ley de control asegura que la salida "y" siga el comando de referencia o punto de consigna, el número de integradores es igual a la dimensión de la salida "y" (Chalán, 2020). Esta ley de control adicionalmente minimiza las funciones de costo:

$$J(u) = \sum_{n=0}^{\infty} \{z^T Q z + u^T R u + 2 z^T N u\}$$

En tiempo discreto, el algoritmo LQI calcula la salida del integrador x_i usando la fórmula de avance de Euler. Discretizando el algoritmo se calculan los valores de las matrices del sistema LTI y se plantea el error del sistema calculando la diferencia entre la referencia y la salida del sistema. Finalmente, se puede utilizar el mismo cálculo del regulador lineal cuadrático (LQR) para obtener el valor de la matriz K tomando los nuevos valores de las matrices A y B del sistema (Chalán, 2020). Obteniendo como resultado la siguiente ecuación:

$$u = -Kx = \begin{bmatrix} K_{m\times n} \\ K_{im\times p} \end{bmatrix}\begin{bmatrix} x_{n\times 1} \\ x_{ip\times 1} \end{bmatrix} = -Kx - K_i x_i$$

A continuación, se procede a definir los valores de las matrices K y K_i. Los cuales fueron obtenidos del Swarm Robotics Toolbox y tienen los siguientes valores:

$$K = \begin{bmatrix} 0.2127 \\ 0.2127 \end{bmatrix} \qquad K_i = \begin{bmatrix} -0.0224 \\ -0.0224 \end{bmatrix}$$

Simulaciones dentro del entorno de ROS

Dentro del entorno de simulación de ROS se tienen diversos nodos o módulos que realizan diversas acciones entre las cuales se encuentra obtener la pose del robot, obtener la orientación del robot, darle una velocidad al robot, obtener información de sensores, etc. ROS cuenta con una gran variedad de nodos que son suscriptores y publicadores, sin embargo, un nodo no puede escribirse a si mismo. Los nodos suscriptores

deben obtener la información de un nodo publicador y un nodo publicador solo puede transmitir información a un nodo suscriptor.

Dentro del entorno de ROS nos encontramos con herramientas muy útiles como lo es Gazebo y Rviz. Gazebo es un simulador 3D mientras que ros sirve como una interfaz para los robots. Combinando ambos se obtiene un simulador muy completo y versátil. Con gazebo se pueden crear escenarios en 3 dimensiones con robots, obstáculos y muchos otros objetos. Gazebo utiliza múltiples motores físicos tales como ODE y Bullet para iluminación, gravedad, inercia, etc.

Por otro lado, la herramienta de Rviz es una poderosa herramienta de visualización 3D para ROS. Permite al usuario ver el modelo de robot simulado, registrar la información de los sensores del robot y reproducir la información del sensor registrada. Al visualizar lo que el robot ve, piensa y hace, el usuario puede depurar una aplicación de robot desde las entradas del sensor hasta las acciones planificadas.

En el caso de la implementación del algoritmo de PSO dentro del entorno de ROS, se utilizó el lenguaje de Python y se importó el modelo del robot utilizado desde el github oficial de ROS. A partir de esto, se utilizo el nodo de odometría para obtener la posición del robot dentro del entorno de simulación de Gazebo. Ya teniendo la posición, se realizaron los cálculos descritos con anterioridad para encontrar la nueva posición a la cual se debe de dirigir el robot utilizando el algoritmo PSO.

Ya teniendo la nueva posición, se realizan los cálculos de la pose al igual que la implementación de un PID para realizar el movimiento del robot hacia la posición deseada. Durante toda la trayectoria, los robots calculan la función de costo para obtener el mejor valor global. Todos los robots lo envían a un nodo y este nodo se encarga de tomar todos los datos y solo transmitir el valor mínimo entre todos los datos que recibe. Finalmente, cada robot está constantemente leyendo este nodo para tener el mejor costo y las mejores posiciones globales de todo el enjambre de robots.

Resultados y discusión

Selección de controlador

Se realizaron pruebas con algoritmos básicos de PSO y controladores PID, LQI y LQR para establecer el comportamiento que deberían de tener los robots y el tiempo que le toma al algoritmo converger en cada caso. En la siguiente tabla se pueden observar los tiempos promedio de las 250 simulaciones que se realizaron con diversos mapas y variando el controlador utilizado:

Controlador	Tiempo Promedio (s)
PID	29.0632
LQR	10.4672
LQI	34.8332

Tabla 2. Tiempos de simulación promedio por controlador

Se realizaron pruebas con diversas funciones de costo para evaluar la velocidad de convergencia de los robots hacia el mínimo local de cada una de ellas. Cabe destacar que todos los controladores son capaces de converger hacia el mínimo global de la función de costo. La única diferencia es el tiempo el cual les tomo a los robots en converger hacia este punto.

Como se puede observar en la tabla 2, el controlador LQR es el controlador con el menor tiempo de convergencia. Por otro lado, se puede observar que los tiempos del controlador PID y el controlador LQI son similares siendo la diferencia 5 segundos entre uno y el otro.

Simulación de algoritmo Braitenberg y PSO en Webots

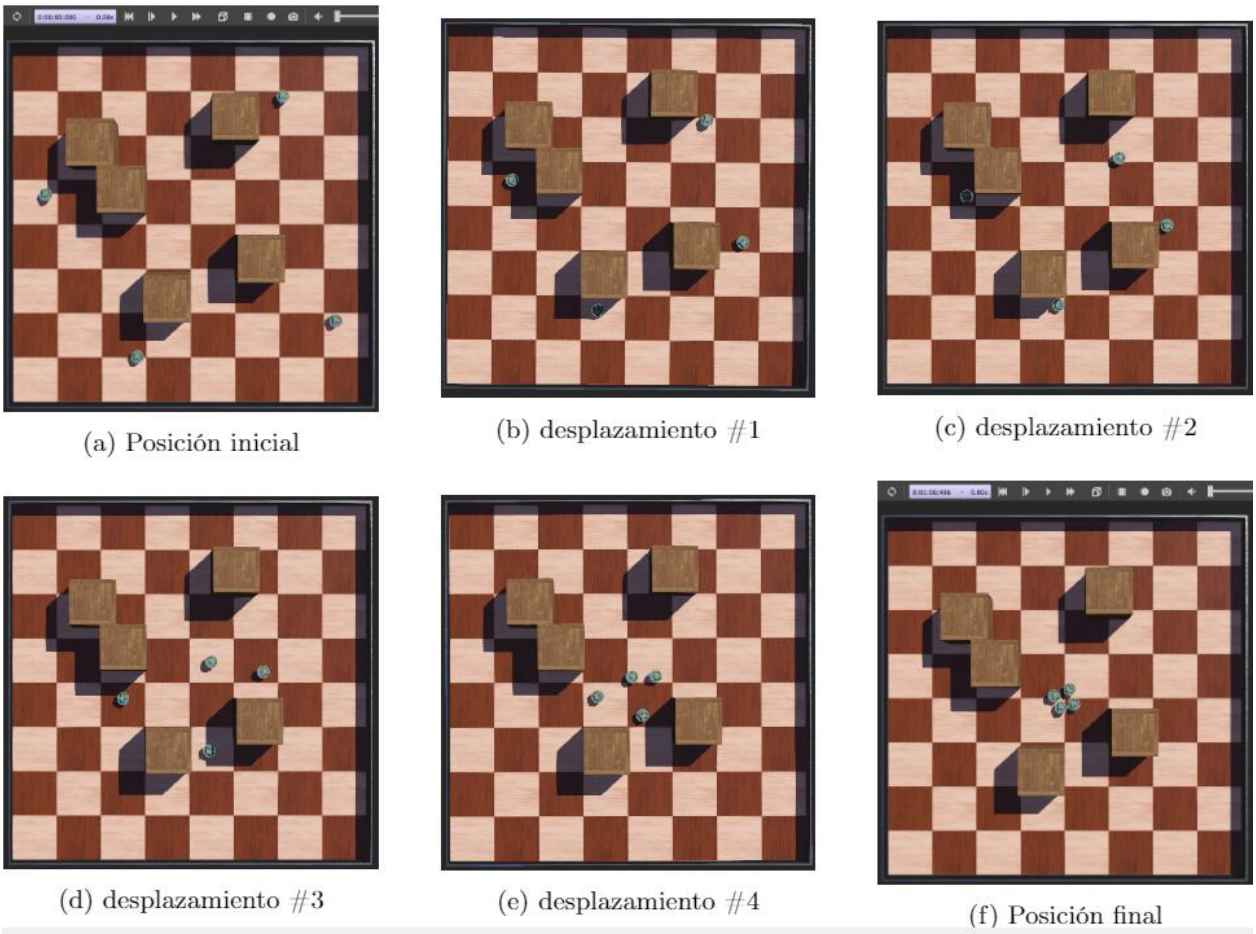

Figura 4. Trayectoria de los robots utilizando el algoritmo de braitenberg y algoritmo PSO

Como se puede observar en la figura 4, el algoritmo PSO es capaz de converger al mínimo global evadiendo los obstáculos dentro del entorno. El sistema es capaz de detectar cuando el obstáculo está cerca y realiza las maniobras pertinentes para lograr evadir el obstáculo y dirigirse hacia el punto establecido por el algoritmo PSO. En el caso de las imágenes mostradas, se utiliza un controlador LQR debido a que es el controlador que obtuvo mejores resultados en lo que respecta a la velocidad de convergencia del algoritmo al igual que la forma en la cual converge.

Simulación de algoritmo PSO en ROS

Se realizaron diversas simulaciones dentro del entorno de ROS para verificar si la implementación de este algoritmo era viable. Los robots y las simulaciones fueron programados utilizando el lenguaje Python. Inicialmente se definió la cantidad de robots que estarían dentro del entorno y sus posiciones iniciales. Luego de esto se dejó correr el algoritmo hasta obtener una convergencia por parte de los robots al mínimo local de la función de costo establecida. En este caso, solo se utilizó la función de la esfera para realizar las pruebas de funcionamiento. Los resultados obtenidos se pueden observar en la figura 5.

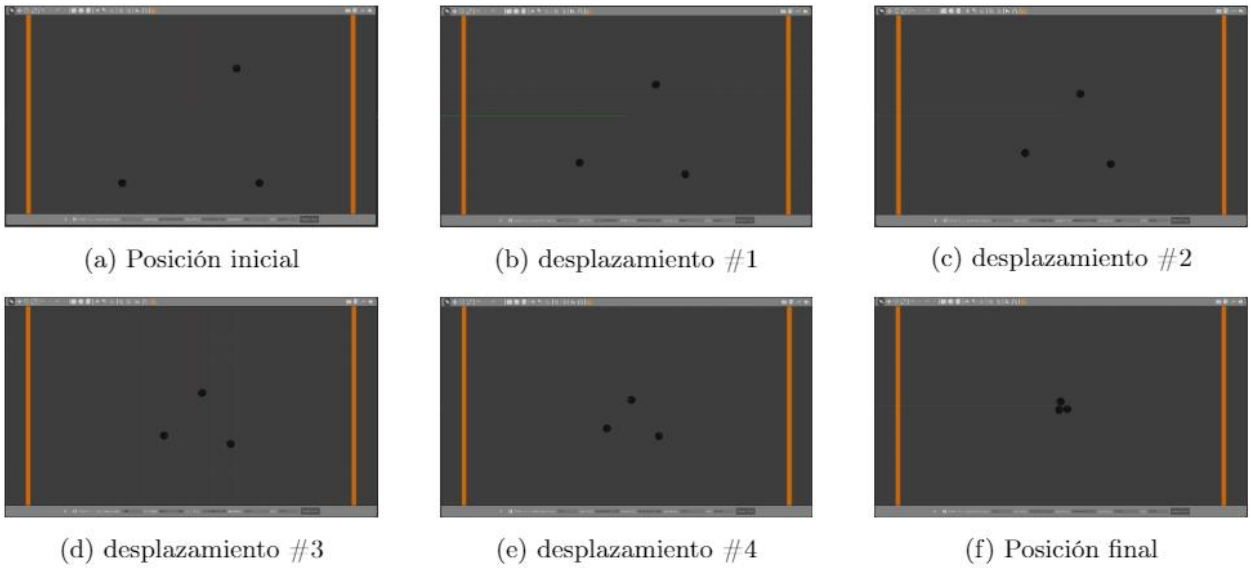

Figura 5. Trayectoria de los robots utilizando Gazebo y algoritmo PSO

Se realizaron diversas simulaciones en el entorno de ROS y el tiempo promedio al cual convergen los robots es de 12 segundos aproximadamente. Cabe destacar que, debido a la definición del modelo del robot, las simulaciones en ROS no contienen giro para la rueda izquierda o derecha, mas bien utilizan el modelo de roll, pitch, yaw. En este caso, se modificaba la dimensión de "yaw" para realizar el giro en el eje z hasta que la parte frontal del robot estuviera apuntando hacia el punto deseado, luego de esto se le daba una velocidad de avance al robot para que llegar a al punto deseado.

Conclusiones

Se determino que los simuladores con motores de física presentan un comportamiento más parecido a una aplicación física real. El sistema se comporta diferente al modelo idealizado dentro de las simulaciones del toolbox debido a que, al contener restricciones de dimensiones, gravedad y fricción, estas cambian el comportamiento del sistema causando una convergencia más lenta y

en funciones con varios mínimos locales se presenta el caso de que los robots se quedan en el mínimo local y no convergen al mínimo global. Por otro lado, se determinó que era viable la utilización de ROS para simulaciones del algoritmo PSO.

Bibliografía

- Aguilar, A., 2019. Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO). Universidad del Valle de Guatemala.
- Álvarez, D., Toro, E., Ramón, G., 2009. Algoritmo de Optimización cumulo de partículas aplicado en la solución de problemas de empaquetamiento óptimo bidimensional con y sin rotación. Scientia et Technica 10–16.
- Beauchamp, G., Batista, R., 2016. Aplicación de Técnicas de Control Óptimo a una plataforma estacionaria cuatrimotor. Revista de Ingeniería Electrónica, Automática y comunicaciones XXXVII, 34–49.
- Braitenberg, V., 1986. Vehicles: Experiments in Synthetic Psychology. MIT Press, Londres.
- Cahueque, J., 2019. Implementación de enjambre de robots en operaciones de búsqueda y rescate. Universidad del Valle de Guatemala.
- Castillo, M., 2019. Diseñar e implementar una red de comunicación inalámbrica para la experimentación en robótica de enjambre (Tesis de licenciatura). Universidad del Valle de Guatemala.
- Chalán, V., 2020. Desarrollo de in controlador optimo LQR utilizando herramientas IOT para un sistema de presión constante controlado remotamente (Master's Thesis). Universidad politécnica salesiana, Quito, Ecuador.
- Chavarría, J., Fallas, J., 2019. El algoritmo PSO aplicado al problema de particionamiento de datos cuantitativos. Matemática, Educación e Internet 19, 1–13.
- Chi, Z., Hai-bing, G.A.O., Liang, G.A.O., Wan-guo, Z., 2010. Particle Swarm Optimization(PSO) Algorithm (Master's Thesis). Huazhong University of Science & Technology, China.
- Cyberbotics Ltd, 2021. Webots.
- Lorandi, A., Hermida, G., Durán, E., Hernández, J., 2011. Controladores PID y Controladores Difusos. Revista de la Ingenieria Industrial 5, 2–6.
- Santizo, E., 2021. Aprendizaje Reforzado y Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre. Universidad del Valle de Guatemala.