



MINI-PROYECTO INICIAL SWARM ROBOTICS

Universidad del Valle de Guatemala
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica
Gabriela Iriarte Colmenares

Guatemala, enero de 2020

Índice

1. Objetivos	3
2. Algoritmos	3
2.1. Particle Swarm Optimization (PSO)	3
2.1.1. Funcionamiento del algoritmo PSO	3
2.1.2. Mejora del algoritmo	4
2.2. Ant Colony Optimization (ACO)	4
2.3. Artificial Bee Colony (ABC)	6
3. Experimentos con PSO	7
3.1. Constantes c_1 y c_2 con w y K constantes	7
3.2. Factor inercial w	8
3.3. Constantes c_1 y c_2 con w lineal y K constante	8
3.4. Coeficiente de restricción K	10
3.5. Experimentos con funciones	11
3.5.1. Rosenbrock	12
3.5.2. Ackley - Varios mínimos locales	12
3.5.3. Rastrigin - Varios mínimos locales	13
3.5.4. Peaks	13
3.5.5. Booth	14
3.6. Implementación de Artificial Potential Fields (APF)	14

1. Objetivos

1. Investigar y trabajar en temas relacionados al proyecto de graduación que realizarán.
2. Definir un cronograma de tareas y completar un proyecto siguiendo dicho cronograma.
3. Aprender sobre manejo de versiones.

2. Algoritmos

2.1. Particle Swarm Optimization (PSO)

Este algoritmo hace referencia al comportamiento de las bandadas de pájaros y cardúmenes de peces al realizar búsquedas óptimas [3]. Las partículas encuentran la solución a partir de su mejor posición y la mejor posición de todas las partículas [4]. Las ecuaciones que modelan esta situación son las siguientes [3], [4]:

$$v_{id}^{t+1} = v_{id}^t + c_1 r_1 (p_{id}^t - x_{id}^t) + c_2 r_2 (p_{gd}^t - x_{id}^t) \quad (1)$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \quad (2)$$

Donde v es la velocidad, x la posición, c son constantes de aceleración, p_{id} es la mejor posición personal, p_{gd} es la mejor posición que tuvo todo el enjambre en la corrida y r son números aleatorios entre 0 y 1. La fase de exploración ocurre cuando se tiene una diferencia grande entre la posición de la partícula y el pbest o gbest. La ventaja de este método contra los que necesitan usar el gradiente es que no se requiere que el problema sea diferenciable y puede optimizarse problemas con ruido o cambiantes. La función que se desea optimizar se llama función de costo [3].

2.1.1. Funcionamiento del algoritmo PSO

Se inicia el programa creando las partículas, dándoles una posición y velocidad inicial. Luego se introducen esos valores a la función de costo, se actualizan los valores de pbest y gbest. Este proceso se repite hasta que se cumpla un número de iteraciones o se logre la convergencia del algoritmo. La convergencia se alcanza cuando todas las partículas son atraídas a la partícula con la mejor solución

(gbest). Un factor que se debe tomar en cuenta es el de restringir los valores que pueden tomar las funciones para que el algoritmo no diverja [4].

2.1.2. Mejora del algoritmo

El algoritmo puede mejorarse a través del incremento de partículas, introducción del peso de inercia (w) o la introducción de un factor de restricción. El factor de inercia controla las fases de explotación y desarrollo del algoritmo. Se sugiere utilizar un valor mayor que y decrementarlo hasta un valor menor a para tener una mayor exploración [3].

$$v_{id}^{t+1} = w v_{id}^{t+1} + c_1 r_1(p_{id}^t - x_{id}^t) + c_2 r_2(p_{gd}^t - x_{id}^t) \quad (3)$$

$$v_{id}^{t+1} = K(w v_{id}^{t+1} + c_1 r_1(p_{id}^t - x_{id}^t) + c_2 r_2(p_{gd}^t - x_{id}^t)) \quad (4)$$

El otro caso es utilizar el factor de restricción K para mejorar la probabilidad de convergencia y disminuir la probabilidad de que las partículas se salgan del espacio de búsqueda [3].

2.2. Ant Colony Optimization (ACO)

Este algoritmo busca imitar el comportamiento que usan las hormigas para la búsqueda de alimento. Cada hormiga deja cierta cantidad de feromonas en el camino que recorre. Sin embargo, esta cantidad se evapora conforme avanza el tiempo. Las demás hormigas eligen la trayectoria que tiene más feromonas. Estas trayectorias son la memoria global de las hormigas. El "deamon" determina si es necesario añadir más feromona al camino para llegar a la convergencia. También se utiliza control descentralizado para tener un algoritmo robusto ante un ambiente dinámico [3].

$$p_{(i,j)}^k(t) = \frac{(\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}}{\sum_{k \in J_k} (\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}} \quad (5)$$

" $p_{(i,j)}$ es la probabilidad de ir del nodo i al j. J_k son los nodos que la hormiga tiene permitidos visitar desde el nodo i. η_{ij} contribuye a la visibilidad entre los nodos i y j. τ_{ij} representa la cantidad de feromona no evaporada entre el nodo i y j en el tiempo t. α y β controlan la influencia de la cantidad de feromona evaporada y la visibilidad entre nodos. Cuando alfa es mayor, el comportamiento de búsqueda

depende más de la cantidad de feromona que hay. Si beta es mayor, se depende más de la visibilidad. Además, cada hormiga tiene una lista taboo para prevenir que las hormigas visiten el mismo nodo dos veces."[3]

Para depositar feromonas se utiliza:

$$\Delta\tau_{ij}^k(t) = \frac{Q}{L_k}(t) \quad (6)$$

Donde Q es una constante y L es el costo del trayecto, como el largo del mismo. El valor representa el cambio de feromonas entre el nodo i y j que la hormiga visitó en la iteración t.

La ecuación que gobierna la evaporación de la feromona en los trayectos es la siguiente:

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \sum_{k=1}^m (\Delta\tau_{ij}^k(t)) \quad (7)$$

Donde m es el número de hormigas, p es el factor de evaporación de feromona. Una ventaja de este método ante el PSO es que siempre converge, mientras que la aleatoriedad del PSO no puede asegurarnos convergencia.

2.3. Artificial Bee Colony (ABC)

Este algoritmo fue propuesto por Dervis Karaboga hace 15 años. Se basa en la búsqueda de alimento que realizan las abejas. Estas abejas se dividen en tres clases: employed, onlooker y scout. La primera se encarga de encontrar una fuente de alimento y guardarla en su memoria. Cada una de estas abejas está asociada solo a una fuente de comida, por lo que habrá la misma cantidad de abejas que de lugares. La abeja onlooker se encarga de recibir la información del primer tipo de abejas para seleccionar una de las fuentes de alimento. Finalmente, el tercer tipo de abeja busca alternativas a la fuente ya seleccionada. Primero se inicializa la población de fuentes de comida, donde l y u son los límites inferior y superior de la posición respectivamente [3].

$$x_i = l_i + rand(0, 1)(u_i - l_i) \quad (8)$$

Luego se va variando la velocidad para llegar a una nueva fuente de comida, y se evalúa la siguiente expresión:

$$v_i = x_i + \emptyset_i(x_i - x_j) \quad (9)$$

Donde x_j es una fuente de comida aleatoria y θ es un número aleatorio entre un rango [-a,a]. El valor de fitness de este algoritmo se calcula como sigue:

$$f(a, b) = \begin{cases} \frac{1}{1+f_i(x_i)} & \text{if } f_i(x_i) \geq 0 \\ 1 + abs(f_i(x_i)) & \text{if } f_i(x_i) < 0 \end{cases} \quad (10)$$

Luego las abejas onlooker por medio de un cálculo de probabilidad eligen sus fuentes de alimento.

3. Experimentos con PSO

Inicialmente las partículas se colocaban en un rango de 0 a 1 en el espacio de dos dimensiones. Sin embargo, se modificó el código para que estas partículas iniciaran esparcidas en todo el espacio de búsqueda. Se encontró que la cercanía de las partículas al lugar objetivo sí importan a la hora de llegar a la convergencia correcta. Para todos los experimentos se utilizó la función de Rosenbrock para un área de búsqueda de 2 dimensiones [1].

3.1. Constantes c_1 y c_2 con w y K constantes

Para esta parte se dejó constante K y w (con valor igual a 1). Se elaboró un código para correr varias veces hasta con 500 iteraciones para probar distintos valores de c_1 y c_2 . Estos valores se variaron desde 0.1 hasta 4 en un step de 0.1 pues se recomienda que el valor esté entre ese rango [3]. Se comparó la distancia euclíadiana entre el último vector de posición de cada partícula y la posición objetivo. Para esto se utilizó un 1% de error en el área (circular). Se contó la cantidad de partículas que entraron dentro del error y se guardó en una celda. El procedimiento anterior se repitió 10 veces y luego estas celdas fueron exportadas a Excel. En excel se anotó cuales combinaciones tuvieron más casos de éxito y se apuntaron en otra hoja de Excel, donde se sacó la moda de estas combinaciones. Los valores más repetidos fueron $c_1 = 0,1$ y $c_2 = 0,9$. Una observación interesante es que el resultado es mejor cuando las c (speed) son más pequeñas (menores a 1). Al graficar este caso se encontró que el algoritmo no converge puesto que luego de 300 iteraciones las partículas siguen buscando aunque el global best ya fue encontrado correctamente. Para arreglar esto se buscará encontrar los mejores valores para los demás parámetros.

3.2. Factor inercial w

Se volvió a modificar el código para realizar varias corridas, pero ahora variando el factor inercial. Se varió con 3 casos distintos:

$$w = \frac{\maxgen - \text{gen}}{\maxgen} \quad (11)$$

$$w = e^{\frac{\maxgen - \text{gen}}{\maxgen} - 1} \quad (12)$$

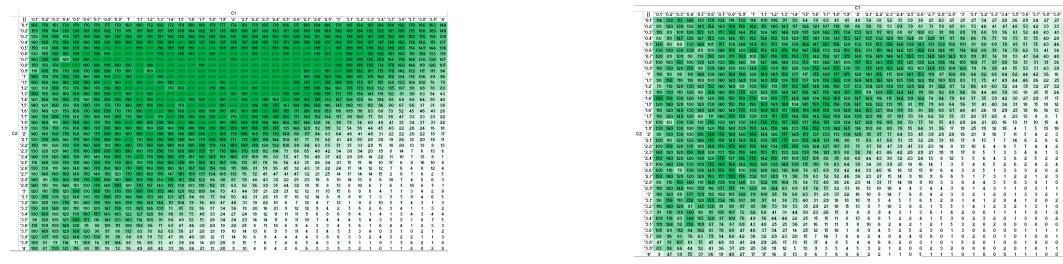
$$w = 1 \quad (13)$$

Donde gen es la iteración actual y maxgen la cantidad máxima de iteraciones, que en este caso, fue de 500. Se realizó 120 corridas de las cuales en cada corrida se evaluaba una sola X inicial de las partículas y en las demás se iba variando para evaluar la misma posición en cada w distinto pero en 120 posiciones distintas. Esto se repitió para un 5% y 1% de error. En cada corrida se guardó el valor de la cantidad de partículas dentro de este radio de error y se guardó en una celda que luego fue exportada a un archivo de excel (pruebas_w.xls). En ambos casos de error pudo observarse que el que tuvo mejor respuesta fue la lineal con un 20% de error en el caso de 5% y 25.8% de error en el caso de 1% (tomando como base el caso ideal de que las 10 partículas llegaran justo al objetivo las 120 veces). Sin embargo, el caso de w constante tuvo un pésimo desempeño, con un 85.6% y 97.3% de error. Por consiguiente, considero que es necesario repetir el experimento de las constantes c_1 y c_2 pues el factor w tiene mucha influencia en la exactitud de las partículas al llegar al objetivo.

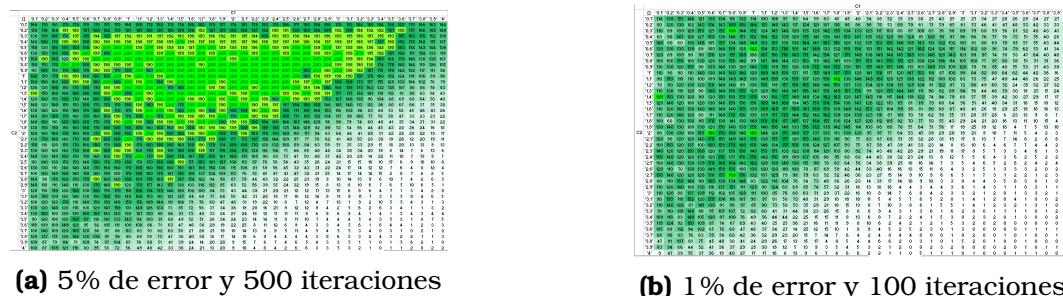
3.3. Constantes c_1 y c_2 con w lineal y K constante

Con el mismo código de la sección anterior sobre las constantes de aceleración se corrió 20 veces el algoritmo para probar todas las combinaciones de c_1 y c_2 . Luego se realizó la suma de las mismas posiciones de las celdas para obtener una celda con la sumatoria total para comparar cuáles tuvieron el mejor desempeño. El valor ideal en este caso sería de 200 pues se desearía que en las 20 iteraciones los 10 individuos del enjambre estuvieran dentro del 5% con un máximo de 500 iteraciones. Esta última celda fue exportada a Excel (pruebas_c.xls) y se le colocó un formato condicional donde mientras más alto es el valor, más oscuro es el verde, como puede observarse en la siguiente imagen:

Luego a la misma matriz se le aplicó otro formato condicional donde a los valores más altos se les colocó el verde fosforecente y a los valores que aún están dentro

**Figura 1:** Mapa de colores de los resultados.

de un 5% de error se les colocó un verde limón. En el caso de las corridas con 100 iteraciones no se logró obtener un resultado tan alto como el anterior. En esta corrida solo se logró obtener una suma de 183, por lo tanto los valores mayores o iguales a 180 fueron resaltados.

**Figura 2:** Mapa con los mejores resultados resaltados.

Al hacer un match entre ambas figuras se encontró que los mejores valores para las constantes son:

$$\begin{aligned} c_1 &= 1,8 \\ c_1 &= 1 \end{aligned}$$

$$\begin{aligned} c_2 &= 1,1 \\ c_2 &= 0,6 \end{aligned}$$

3.4. Coeficiente de restricción K

Eberhart y Shi introdujeron el concepto de coeficiente de restricción K para mejorar el comportamiento del algoritmo. El coeficiente puede calcularse a través de las siguientes ecuaciones [2]:

$$K = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \quad (14)$$

$$\phi = \phi_1 + \phi_2 > 4 \quad (15)$$

$$\phi > 4 \quad (16)$$

Según Eberhart lo más común es utilizar w como una función lineal que decrece de 0.9 a 0.4, por lo que se modificó esto en el algoritmo. También se agregó los valores explicados anteriormente y se bajó el número de iteraciones a 100 para comparar si en realidad estas mejoras hacen que decrezca el tiempo de convergencia. Además, el mismo autor recomienda que además de hacer un clamping de los valores de la posición también se haga uno de los valores de la velocidad como el rango que se tiene dividido entre 5. El procedimiento en Matlab fue recorrer los valores de 0 a 4 para ϕ_1 y ϕ_2 pero 20 veces, guardando estos valores en un vector en cada elemento de la celda. Luego se guardó en otra celda el valor promedio de cada vector en cada celda inicial.

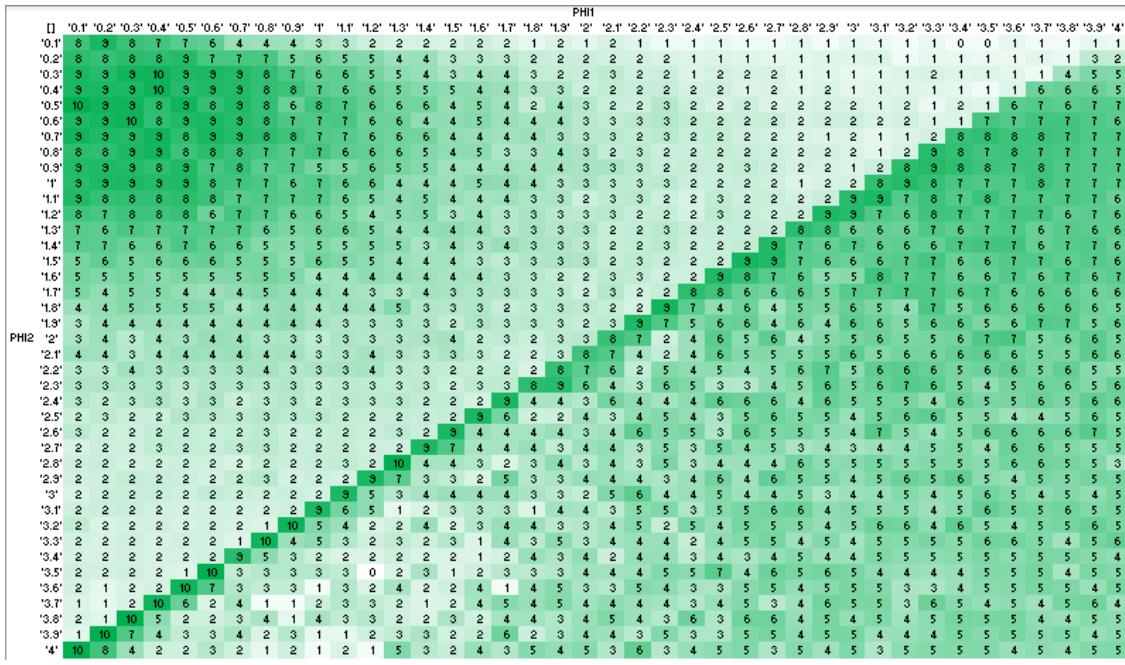


Figura 3: Valores promedio de ϕ_1 y ϕ_2 en 20 iteraciones.

Puede observarse en la figura 3 que sí mejoró la respuesta comparado al número de iteraciones, pues fueron solo 100 con un 1% de error en las distancias. Además, también se pudo comprobar que los valores más altos son producidos por los números que suman 4.1, tal como lo pide Ebenhart. Existen algunos casos atípicos en la esquina superior izquierda, pero considero que es mejor apoyarnos a lo expuesto por el autor. La gráfica de las partículas realizando la búsqueda se pueden observar en la siguiente sección.

3.5. Experimentos con funciones

Como se explicó anteriormente, todos los experimentos fueron probados con la función de costo de Rosenbrock (en forma de valle). Sin embargo, también se realizó otras pruebas con 4 funciones más. Se modificó el código de Matlab para aceptar como parámetro el nombre de la función de costo a utilizar. Las funciones disponibles son: Banana (Rosenbrock), Ackley, Rastrigin, Booth y la función Peaks de Matlab. En este apartado solo se mostrarán los resultados para las 5 funciones con los mejores parámetros encontrados en las secciones anteriores. En [1] se explica que existen 5 tipos básicos de funciones de optimización para hacer pruebas: con varios mínimos locales, forma de tazón, forma de placa, forma de valle y forma de gotas. De estas se evaluó 3 tipos.

3.5.1. Rosenbrock

Esta función cuenta con una forma de valle, por lo que generalmente sí logra llegar al mínimo global.

Figura 4: Animación de la búsqueda con Rosenbrock.

3.5.2. Ackley - Varios mínimos locales

Esta función cuenta con varios mínimos locales, por lo que algunas veces el algoritmo converge a estos en lugar de al mínimo global.

Figura 5: Animación de la búsqueda con Ackley

3.5.3. Rastrigin - Varios mínimos locales

Esta función al igual que la anterior cuenta con varios mínimos locales, por lo que algunas veces el algoritmo convergía a estos en lugar de al mínimo global.

Figura 6: Animación de la búsqueda con Rastrigin

3.5.4. Peaks

Esta función al igual que las anteriores cuenta con varios mínimos locales, por lo que algunas veces el algoritmo convergía a estos en lugar de al mínimo global.

Figura 7: Animación de la búsqueda con Peaks

3.5.5. Booth

Esta función cuenta con una forma de placa, por lo que generalmente sí logra llegar al mínimo global.

Figura 8: Animación de la búsqueda con Booth

3.6. Implementación de Artificial Potential Fields (APF)

Se implementó el algoritmo APF utilizando un círculo como obstáculo y la distancia euclíadiana como medida. Tuve problemas con la suavidad de la caída pero logré arreglarlo modificando el Q^* , puesto que esto regula qué tan estricto es el algoritmo en cuanto a la repulsión de los obstáculos.

Figura 9: Animación de la búsqueda con APF

Referencias

- [1] D. Bingham. (2017). Optimization Test Problems, dirección: <https://www.sfu.ca/~ssurjano/optimization.html>.
- [2] R. Eberhart e Y. Shi, «Comparing inertial weights and Constriction factor in particle swarm optimization», vol. 1, feb. de 2000, 84-88 vol.1, ISBN: 0-7803-6375-2. doi: 10.1109/CEC.2000.870279.
- [3] M. N. A. Wahab, S. Nefti-Meziani y A. Atyabi, «A Comprehensive Review of Swarm Optimization Algorithms», *PLoS ONE*, vol. 10, n.º 5, 2015. doi: <https://doi.org/10.1371/journal.pone.0122827>.
- [4] Y. Zhang, S. Wang y G. Ji, «A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications», *Hindawi*, vol. 2015, n.º 931256, 2015. doi: <http://dx.doi.org/10.1155/2015/931256>.