UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica

Protocolo de trabajo de graduación presentado por Gabriela Iriarte Colmenares, estudiante de Ingeniería Electrónica

Guatemala,

2020

Resumen

En el siguiente protocolo de trabajo de graduación se explicará la propuesta que se tiene sobre inteligencia de enjambre para aplicaciones de robótica como trabajo de graduación. Principalmente se desea comparar el rendimiento entre el algoritmo Particle Swarm Optimization (PSO) y Ant System (o simplemente Ant Colony), que son algoritmos de inteligencia de enjambre con distintos enfoques. Uno de los objetivos es encontrar cuál de estos dos algoritmos es el mejor, para que luego sea implementado en robots físicos similares a un robot E-Puck. Como metodología, primero se implementará el algoritmo Simple Ant Colony, para luego implementar el algoritmo Ant System. Estos algoritmos difieren en la forma en la que toman la decisión de qué camino seguir. Luego de esto se validarán los algoritmos por medio de una simulación en Matlab y finalmente se trasladará el Ant System al lenguaje C para ser implementado en el software Webots. Al final del trabajo se presenta el cronograma de actividades a seguir y un índice preliminar de la propuesta de trabajo de graduación.

Antecedentes

Megaproyecto Robotat - Fase II

En la segunda fase del proyecto Robotat [1] se elaboró un algoritmo basado en el Particle Swarm Optimization (PSO) para que distintos agentes llegaran a una meta definida (el mínimo de una función de costo). El algoritmo modificado elaboraba una trayectoria a partir de la actualización de los puntos de referencia a seguir, generados por el PSO clásico. Asimismo, se realizó distintas pruebas para encontrar los mejores parámetros del algoritmo PSO, probando distintas funciones de costo. Para realizar dicho algoritmo se tomó en cuenta que la velocidad de los robots diferenciales tiene un límite, además de tener que seguir las restricciones físicas de un robot real. Para simular estas restricciones se utilizó el software de código abierto Webots luego de haberlo implementado en Matlab como partículas sin masa ni restricciones.

En Webots también se implementó distintos controladores para que la trayectoria fuera suave y lo más uniforme posible. Entre los controladores que se implementaron están: Transformación de uniciclo (TUC), Transformación de uniciclo con PID (TUCPID), Controlador simple de pose (SPC), Controlador de pose Lyapunov-estable (LSPC), Controlador de direccionamiento de lazo cerrado (CLSC), Transformación de uniciclo con LQR (TUC-LQR), y Transformación de uniciclo con LQI (TUC-LQI). El controlador con el mejor resultado en esta fase fue el TUC-LQI.

Robotarium de Georgia Tech

El Robotarium del Tecnológico de Georgia en Estados Unidos desarrolló una mesa de pruebas para robótica de enjambre para que personas de todo el mundo pudieran hacer pruebas con sus robots. De este modo, ellos están apoyando a que más personas, sin importar sus recursos económicos, puedan aportar a la investigación de robótica de enjambre. Además, eso ayudaría a los investigadores a dejar de simular y probar sus algoritmos en prototipos

reales. Para utilizar la plataforma hay que descargar el simulador del Robotarium de Matlab o Python, registrarse en la página del Robotarium y esperar a ser aprobado para crear el experimento [2].

Wyss Institute Swarm Robots

El Instituto Wyss de Harvard desarrolló robots de bajo costo miniatura llamados *Kilobots* para probar algoritmos de inteligencia computacional de enjambre. Este tipo de robots es desarrollado para potencialmente realizar tareas complejas en el ambiente como lo son polinizar o creación de construcciones. Estos Kilobots cuentan con sensores, micro actuadores y controladores robustos para permitir a los robots adaptarse a los ambientes dinámicos y cambiantes [3].

Aplicaciones de Ant Colony Optimization (ACO)

En [4] puede encontrarse un trabajo similar, donde se aplica el método de ACO en un robot autónomo en una cuadrícula con un obstáculo estático. El robot lo colocan en la esquina inferior izquierda y se pretende que alcance el objetivo en la esquina superior derecha de la cuadrícula. La simulación de este trabajó se realizó en Matlab y se implementó una cuadrícula de 100x100 con la unidad como el tamaño de las aristas. Además, en la cuadrícula los agentes podían moverse norte, sur, este, oeste y en forma diagonal.

También en [5] se realizó una comparación de los algoritmos PSO y ACO, en donde se resaltó algunas ventajas y desventajas de cada uno de los métodos. El PSO es simple pero sufre al quedarse atascado con mínimos locales por la regulación de velocidad. Por otro lado, el ACO se adapta muy bien en ambientes dinámicos, pero el tiempo de convergencia puede llegar a ser muy largo (aunque la convergencia sí está garantizada).

Justificación

En la segunda fase del proyecto Robotat de la Universidad del Valle de Guatemala se desarrolló un algoritmo basado en el algoritmo Particle Swarm Optimization para planificar una trayectoria óptima para que los robots llegaran a una meta determinada. También se elaboró el diseño de distintos controladores para asegurar que los robots recibieran velocidades coherentes con respecto a sus limitantes físicas. Ya que este algoritmo y controlador fueron exitosos en simulación, se desearía comparar con otros algoritmos para que, de este modo, el mejor algoritmo sea finalmente implementado en los robots físicos.

Como propuesta de algoritmo se tiene el Ant Colony Optimization(ACO), pues es otro de los más utilizados de la rama de inteligencia computacional de enjambre. Se debe de encontrar los mejores parámetros para que el algoritmo converja en un tiempo similar al logrado con el PSO de la fase II del proyecto Robotat. Asimismo se busca implementar los mismos controladores que en el proyecto mencionado anteriormente para que la comparación sea equitativa.

Objetivos

Objetivo General

Implementar y verificar algoritmos de inteligencia de enjambre y computación evolutiva como alternativa al método de Particle Swarm Optimization para los Bitbots de UVG.

Objetivos Específicos

- Implementar el algoritmo Ant Colony Optimization (ACO) y encontrar el valor de los parámetros de las ecuaciones del ACO que permitan a los elementos de la colonia encontrar una meta específica, en un tiempo finito.
- Validar los parámetros encontrados por medio de simulaciones computarizadas que permitan la visualización del comportamiento de la colonia.
- Adaptar los modelos del movimiento y de la cinemática de los Bitbots, desarrollados en la fase anterior del proyecto, al algoritmo ACO.
- Validar el algoritmo ACO adaptado implementándolo en robots físicos simulados en el software WeBots, y comparar su desempeño con el del Modified Particle Swarm Optimization (MPSO).

Marco teórico

Particle Swarm Optimization (PSO)

Este algoritmo hace referencia al comportamiento de las bandadas de pájaros y cardúmenes de peces al realizar búsquedas óptimas [6]. Las partículas encuentran la solución a partir de su mejor posición y la mejor posición de todas las partículas [7]. Las ecuaciones que modelan esta situación son las siguientes [6], [7]:

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

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \tag{2}$$

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 [6].

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 [7].

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 constricción. 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 [6].

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

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

El otro caso es utilizar el factor de constricción K para mejorar la probabilidad de convergencia y disminuir la probabilidad de que las partículas se salgan del espacio de búsqueda [6].

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 [6].

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

«$p_{(i,j)}$ 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» [6].

Para depositar feromona se utiliza:

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

Donde Q es una constante y L es el costo del trayecto, como el largo del mismo. El valor representa el cambio de feromona 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:

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

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.

Metodología

Para implementar el algoritmo Ant Colony Optimization (ACO) y encontrar el valor de los parámetros de las ecuaciones del ACO que permitan a los elementos de la colonia encontrar una meta específica, en un tiempo finito se propone:

1. Implementar el algoritmo Simple Ant Colony (SACO) en Matlab de forma básica utilizando pseudocódigos encontrados en libros y otras fuentes.
2. Implementar el algoritmo Ant System a partir del SACO en Matlab para tener más flexibilidad en los parámetros.
3. Realizar el barrido de los parámetros ρ, α y β al principio, con la cantidad de iteraciones y costo final del camino encontrado como métrica de desempeño. Mientras se hace esto, los demás parámetros se mantendrán constantes.
4. Realizar el barrido de la cantidad de hormigas y parámetro Q con la misma métrica que el inciso anterior (iteraciones y costo final).

Para validar los parámetros encontrados por medio de simulaciones computarizadas que permitan la visualización del comportamiento de la colonia se propone:

1. Realizar una gráfica del camino final, elegido por las hormigas como el óptimo[1] al final del algoritmo.

[1]Como condición de paro se tiene que al menos el 90 % de las hormigas sigan el mismo camino o que se llegue a un máximo de 50 iteraciones.

2. Realizar una simulación animada donde pueda observarse a través del tiempo cómo es que las hormigas están depositando feromona. En otras palabras, que sea posible evidenciar que efectivamente el algoritmo está convergiendo a un camino óptimo [2] que lleva del nodo origen al destino.

3. Realizar otra gráfica donde se registre la evolución del costo final (por iteración) a través del tiempo (con respecto al número de iteraciones) para evaluar la correcta convergencia del algoritmo.

Para adaptar los modelos del movimiento y de la cinemática de los Bitbots, desarrollados en la fase anterior del proyecto, al algoritmo ACO se propone:

1. Implementar el algoritmo en el lenguaje C para su uso en el software Webots o alternativamente utilizar el código de Matlab y de forma offline[3] calcular el camino óptimo para cada robot.

2. Estudiar formas nuevas de *pathplanning* que sean de utilidad para varios robots en simultáneo.

3. Implementar los controladores realizados por la fase anterior del proyecto Robotat UVG.

Para validar el algoritmo ACO adaptado implementándolo en robots físicos simulados en el software WeBots, y comparar su desempeño con el del Modified Particle Swarm Optimization (MPSO) se propone:

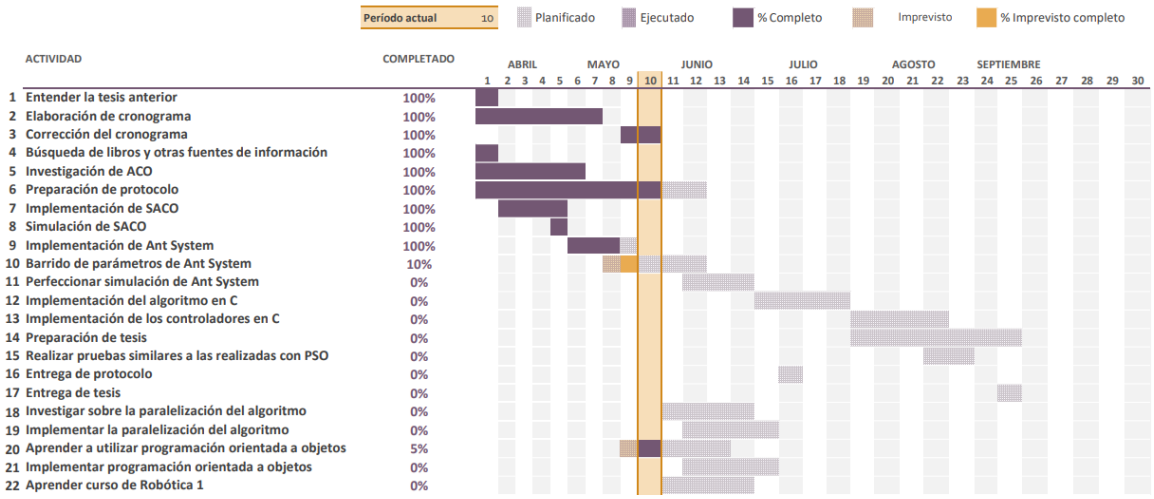
1. Realizar las mismas pruebas que las que se realizaron en la fase anterior del proyecto Robotat (suavidad de trayectoria, velocidad de ruedas, desempeño de cada controlador, etc.)

2. Utilizar las pruebas generadas anteriormente para realizar la comparación entre los algoritmos MPSO y AS eligiendo al mejor dependiendo de la velocidad de convergencia (cantidad de iteraciones), suavidad de la trayectoria y esfuerzo de los actuadores del robot (motores).

[2] En este momento se calcularía el costo óptimo como el camino en diagonal pues se le asignó un costo menor de forma arbitraria

[3] Evaluar paralelización y programación orientada a objetos.

Cronograma de actividades



Descripción de las actividades

1. Leyendo acerca del tema, preguntando al autor y replicando gráficas de su trabajo de graduación.

2. Anotar de forma semanal los avances que se realizan.

3. Aplicar las correcciones dadas por el catedrático de Diseño e Innovación.

4. Investigar sobre libros, artículos científicos, páginas de internet y otras fuentes que puedan servir para la investigación. Principalmente se hizo una búsqueda amplia de información en la primera semana, pero si no llegara a bastar se podría ampliar el tiempo de búsqueda.

5. Investigar en libros y artículos ya recopilados sobre el algoritmo Ant Colony Optimization.

6. Se tiene planificado avanzar en el protocolo conforme se va haciendo la investigación y el proyecto. Es decir, cuando se realizan los resultados de una vez plasmarlos en el protocolo. Asimismo, cuando se vaya investigando ir documentándolo en el protocolo. Luego de la semana 10 aplicar las últimas correcciones dadas por el catedrático.

7. De los algoritmos que componen la optimización por hormigas se encuentra el simple (SACO) y una variante más compleja (AS). Por lo tanto, primero se implementará el algoritmo más sencillo para comprender los conceptos básicos del algoritmo para luego hacer modificaciones y llevarlo a un modelo más complejo.

8. Cuando se haya terminado (o durante) la codificación del algoritmo, se realizará una simulación sencilla en el software Matlab para validar el funcionamiento del mismo.

9. En esta etapa se implementará las modificaciones sobre el SACO para realizar el algoritmo más complejo AS en Matlab.

10. Se buscará encontrar los mejores parámetros para encontrar la mejor solución en términos de número de iteraciones y costo final de la solución.

11. Se intentará implementar el algoritmo en Matlab utilizando programación orientada a objetos, paralelización (para aprovechar todos los cores de la computadora) y técnicas de pathfinding y generación de nodos vistos en robótica 1. Además de esto, se desea encontrar una mejor manera de presentar la simulación gráfica (probablemente utilizando la función quiver de Matlab).

12. Se evaluará la implementación del algoritmo en C para utilizar el software Webots, pues así se realizó en la fase pasada del proyecto Robotat.

13. Se evaluará la implementación de controladores en C para utilizar el software Webots, pues así se realizó en la fase pasada del proyecto Robotat.

14. Se tiene planificado avanzar en el trabajo de graduación conforme se va haciendo la investigación y el proyecto. Es decir, cuando se realizan los resultados de una vez plasmarlos en el trabajo de graduación. Asimismo, cuando se vaya investigando ir documentándolo en el trabajo escrito. Luego de la semana 24 aplicar las últimas correcciones dadas por el catedrático.

15. Replicar pruebas realizadas en la fase anterior del proyecto Robotat para hacer la comparación entre los algoritmos.

16. Entregar el protocolo según las fechas asignadas por la Universidad.

17. Entregar el trabajo de graduación según las fechas asignadas por la Universidad.

18. Investigar sobre la implementación y el funcionamiento de la paralelización con parallel pool en Matlab.

19. Comenzar con la implementación de la paralelización del algoritmo en Matlab.

20. Investigar acerca de la programación orientada a objetos utilizando libros y videotutoriales en python y Matlab.

21. Implementar el algoritmo utilizando conceptos de programación orientada a objetos en Matlab (o quizas python dependiendo de la conveniencia).

22. Utilizar el contenido del curso para auto-enseñarme el curso de robótica 1, pues algunos conceptos pueden ser útiles para la generación de trayectorias.

Índice preliminar

Prefacio v

Lista de figuras ix

Lista de cuadros xi

Resumen xiii

Abstract xv

1. Introducción 1

2. Antecedentes 3
- 2.1. Megaproyecto Robotat - Fase II . . . 3
- 2.2. Robotarium de Georgia Tech . . . 3
- 2.3. Wyss Institute Swarm Robots . . . 4
- 2.4. Aplicaciones de Ant Colony Optimization (ACO) . . . 4

3. Justificación 5

4. Objetivos 7
- 4.1. Objetivo General . . . 7
- 4.2. Objetivos Específicos . . . 7

5. Alcance 9

6. Marco teórico 11
- 6.1. Inteligencia Computacional Swarm . . . 11
- 6.2. Particle Swarm Optimization (PSO) . . . 11
  - 6.2.1. Funcionamiento del algoritmo PSO . . . 12
  - 6.2.2. Mejora del algoritmo . . . 12
- 6.3. Ant Colony Optimization (ACO) . . . 12
  - 6.3.1. Simple Ant Colony Optimization (SACO) . . . 13
  - 6.3.2. Ant System . . . 14

6.4. Robots diferenciales 15
6.5. Controladores de posición y velocidad de robots diferenciales 15
6.6. Programación paralela 15

7. Estableciendo el valor correcto de los parámetros para AS 17
7.1. Simulación simple de AS 17
7.2. Simulación compleja de AS 17

8. Estructuración del algoritmo para simulación en Webots 19

9. Planificación de trayectorias 21

10. Implementación de controladores para simulación en Webots 23

11. Conclusiones 25

12. Recomendaciones 27

13. Bibliografía 29

14. Anexos 31
14.1. Repositorio de Github 31

15. Glosario 33

Referencias

[1] A. S. A. Nadalini, «Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO)», Universidad del Valle de Guatemala, UVG, nov. de 2019.

[2] *Robotarium | Institute for Robotics and Intelligent Machines*. dirección: `http://www.robotics.gatech.edu/robotarium`.

[3] *Programmable Robot Swarms*, en-US, ago. de 2016. dirección: `https://wyss.harvard.edu/technology/programmable-robot-swarms/` (visitado 06-04-2020).

[4] S. P. ROUL, «Application of Ant Colony Optimization for finding Navigational Path of mobile robot», National Institute of Technology, Rourkela, 2011.

[5] *Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques V.Selvi Lecturer, Department of Computer Science, Nehru Memorial College,*

[6] 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`.

[7] 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`.