UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Implementación y Validación del Algoritmos de Robótica de Enjambre Particle Swarm Optimization en Sistemas Físicos.

Protocolo de trabajo de graduación presentado por Walter Andres Sierra Azurdia, estudiante de Ingeniería Mecatrónica

Guatemala,

2021

## Resumen

En el siguiente protocolo se presenta la propuesta que se tiene para el trabajo de graduación enfocado en inteligencia computacional de enjambre. La idea principal es la continuación de la fase anterior la cual implica implementar en robots físicos, los algoritmos ya desarrollados (Simple Ant Colony Optimizarion y Ant System) a nivel de simulación. Como metodología se tiene pensado implementar en lenguaje C, en primer lugar el Simple Ant Colony Optimization para luego implementar el Ant System, tambien implementar distintos controladores para realizar las pruebas y verificar como se comportan con los distintos métodos de control y finalmente comprara los resultados con los obtenidos en las simulaciones. Se presenta el cronograma de actividades a seguir.

## Antecedentes

## Implementación de *Bug Algorithm* en un robot E-Puck

En este trabajo desarrollado por Departamento de Ingeniería en sistemas y Control del Instituto Universitario de Automática e Informática Industrial, en la Universidad Politécnica de Valencia. Se implemento en un robot E-Puck un algoritmo para que este avanzara en una trayectoria mientras evadía obstáculos en el camino.

Aunque este tipo de algoritmos (Seguimiento de trayectoria con obstáculos) y entorno de programación para el E-Puck ya existen. Se modelo la cinemática del robot y se simulo la implementación del algoritmo también se realizaron pruebas con el robot físico. Tanto en la simulación utilizando WeBots como en el robot real se obtuvo un buen desempeño, a pesar de los buenos resultados analizaron los efectos al ser expuestos a distintos factores ambientales y como estos afectaban el desempeño de los robots. Uno de los factores que mas afecto en el desempeño del robot fue el color de los obstáculos, ya que el robot detectaba mejor los obstáculos que eran de color rojo. También se pudo observar que el modulo de odometría del robot era menos preciso cuando la distancia recorrida aumentaba o cuando el robot realizaba una cantidad considerable de rotaciones. Finalmente se pudo demostrar no solo la implementación del *Bug Algorithm* sino también la implementación de un sistema de control para mantener la trayectoria de una manera muy precisa [1].

### Implementación de PSO en Robots Diferenciales

En la segunda fase del proyecto Robotat de la Universidad del Valle de Guatemala desarrollado por Aldo Aguilar [2], se utilizo como base el algoritmo clásico *Particle Swarm Optimization* y este se modifico con el fin de poder implementarlo en robots diferenciales reales ya que sin esta modificación, los movimientos que realizan las partículas (robots) con el algoritmo estándar son irregulares y como consecuencia se tendrá una saturación en los actuadores del robot.

El algoritmo modificado emplea un planeador que antes de seguir la posición exacta, utiliza

un sistema de control el cual retraerá la trayectoria determinada y generara velocidades suaves y continuas para los robots diferenciales. Se realizaron distintas pruebas en entorno de simulación de *WebBots* en las cuales se determino que el controlador con mejor desempeño fue el *TUC-LQI*.

### Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre

La tercera fase el proyecto Robotat desarrollado por Eduardo Santizo [3] se proponen dos soluciones para realizar mejoras al algoritmo PSO mediante el uso de técnicas de aprendizaje reforzado y profundo. La primera propuesta es una mejora al algoritmo por medio de redes neuronales recurrentes y la segunda propuesta fue una alternativa al algoritmo de navegación alrededor de un ambiente conocido por medio de programación dinámica.

Para poder mejorar el desempeño del PSO se empleo el *PSO Tuner* que consiste de una red neuronal recurrente que toma diferentes métricas propias de las partículas del PSO y a través de su procesamiento por medio de una red LSTM, GRU o BiLSTM las torna en predicciones de los parámetros que deberá utilizar el algoritmo. Luego de obtener los parámetros adecuados, el PSO Tuner fue capaz de reducir el tiempo de convergencia a mínimos locales, la red utilizada fue la BiLSTM ya que esta resulto ser la mejor opción de las tres propuestas.

Para la alternativa al algoritmo se utilizo como base la programación dinámica de *Grid-world*, sin embargo este se modifico para lograr ajustar los problemas con robots. En primer lugar, se incremento el numero de acciones que puede tomar el robot, es decir que ahora el robot se podrá mover en una diagonal a $45^o$. Luego se divide el espacio de trabajo en celdas y es escaneado para determinar las celdas obstáculo y meta. Por ultimo, mediante *Policy Iteration* se genera una acción óptima por estado, para generar una trayectoria a seguir por el controlador, tanto el PSO Tuner como el planificador de trayectorias fueron probados a nivel de simulación utilizando Matlab.

### Implementación del ACO en robots diferenciales

En el trabajo desarrollado por Gabriela Iriarte [4], en el cual se implemento el algoritmo de inteligencia de enjambre *Ant System* o *Ant Colony* como planificador de trayectorias y poder tener otra alternativa al MPSO. Para poder comparar el desempeño de este algoritmo se utilizaron varios controladores y se determino cual de todos resulta ser mas efectivo, luego de realizar las pruebas, el controlador con la respuesta mas suave son el controlador de pose y el controlador de pose de Lyapunov. Tambien se realizaron pruebas con algoritmos genéticos para explorar otras alternativas al MPSO y el AC. El planificador de trayectorias fue realizado en Matlab y para realizar pruebas se realizo una simulación en Webots utilizando como robot el E-Puck.

El alcance tanto de este trabajo como el realizado por Eduardo Santizo [3], fue realizar la implementación de los algoritmos y realizar pruebas con dicho algoritmo todo a nivel de simulación utilizando Matlab y Webots como herramientas para hacer las pruebas.

## Justificación

Dentro de la rama de inteligencia computacional de enjambre, el Ant Colony Optimization (ACO) es uno de los algoritmos mas utilizados, este consiste en un método de optimización basado en el comportamiento de insectos para determinar el camino mas rápido hacia la meta.

En la fase anterior, se planteo el uso del algoritmo Ant Colony como un método alternativo al PSO en el proyecto Robotat de la Universidad del Valle de Guatemala. Este es utilizado para planificar la trayectoria utilizada por el robot para lograr llegar a la meta de la forma mas rápida. También se implementaron distintos controladores para tener una métrica de comparación entre dichos controladores y poder comparar el desempeño con el algoritmo PSO. Los resultados de este planificador de trayectorias fueron probados en un entorno de simulación en Matlab y Webots.

Dado el alcance de la fase anterior, en el cual se llego a implementar el algoritmo exitosamente a nivel se simulación, se busca implementar el planificador de trayectorias en robots diferenciales físicos, para de esta forma poder someter al sistema a factores reales y verificar el desempeño de los robots diferenciales móviles físicos al ser puestos a prueba en ambientes controlados utilizando el algoritmo Ant Colony.

## Objetivos

### Objetivo General

Implementar y validar los algoritmos de robótica en enjambre *Ant Colony Optimization (ACO)* y algunas variabtes desarrollados en años anteriores a nivel de simulación, en robots físicos.

### Objetivos Específicos

- Evaluar distintas opciones de robots, microcontroladores, lenguajes de programación y entornos de desarrollo, y seleccionar los más adecuados para su uso en aplicaciones de robótica de enjambre utilizando el algoritmo ACO.
- Migrar los algoritmos desarrollados anteriormente a los microcontroladores de los robots físicos seleccionados.
- Validar la migración de los algoritmos y verificar el desempeño de los sistemas físicos mediante pruebas simples en ambientes controlados.

# Marco teórico

# Particle Swarm Optimization (PSO)

La optimización por enjambre de partículas (PSO) es un método estocástico de optimización desarrollado por Eberhart y Kennedy en 1995, este esta basado en el comportamiento de enjambres de animales en la naturaleza tales como parvada de aves o cardumen de peces.

El algoritmo consiste en una función $f(x, y)$, que podemos evaluar en los puntos que queramos pero es posible conocer su expresión dado que no existe una solución analítica. El objetivo es el habitual en optimización, encontrar valores de $x$ e $y$ para los que la función $f(x, y)$ sea máxima o mínima, a esta función se le suele llamar función fitness o función de costo la cual determinara que tan buena es la posición actual para cada partícula.

El PSO posee ciertas ventajas en comparación con otros métodos de optimización clásica ya que estos utilizan métodos secuenciales, los cuales permiten explorar el espacio de solución solamente en una dirección a la vez en cambio el PSO puede explorar el espacio de la solución en múltiples direcciones simultáneamente. Si un camino no conduce al óptimo, el algoritmo fácilmente elimina tal camino y continúa buscando otros mejores mediante un paralelismo intrínseco. Otra fortaleza de los algoritmos de optimización es su buen desempeño en problemas cuyo espacio de solución presenta múltiples mínimos locales, otros métodos por lo regular quedan atrapados en estos mínimos locales. Ya que no se tiene una solución analítica, el métodos es capaz de lidiar las desventajas de las técnicas heurísticas porque no existe un parámetro de comparación.

## Funcionamiento

Se comienza situando partículas al azar en el espacio de búsqueda, asignándoles una posición y velocidad inicial pero dándoles la posibilidad de que se muevan a través de él de acuerdo a unas reglas que tienen en cuenta el conocimiento personal de cada partícula y el conocimiento global del enjambre. Lo interesante de este algoritmo es que dándoles a las partículas una capacidad simple de movimiento por este espacio de búsqueda y de comunicación entre ellas pueden llegar a descubrir valores particularmente altos para f(x,y) gastando pocos recursos computacionales (cálculos, memoria y tiempo).

Cada partícula (individuo) tiene una posición P en el espacio de búsqueda y una velocidad, v, que determina su movimiento a través del espacio. Además, como partículas de un mundo real físico, tienen una cantidad de inercia, que los mantiene en la misma dirección en la que se movían, así como una aceleración, que depende principalmente de dos características: cada partícula es atraída hacia la mejor localización que ella, personalmente, ha encontrado en su historia (mejor personal) y cada partícula es atraída hacia la mejor localización que ha sido encontrada por el enjambre en el espacio de búsqueda (mejor global), estos dos parámetros son llamados coeficientes de aceleración ($C_1$ y $C_2$). Este modelo esta

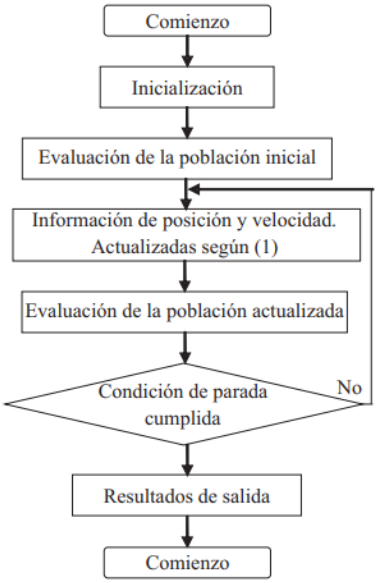



Figura 1: Diagrama de flujo, PSO. [5].

definido por dos ecuaciones, ecuación de posición y de velocidad [5] .

$$v_i(t+1) = v_i(t) + c_1 r_1 (P_i^{best} - P_i(t)) + c_2 r_2 (P_{global}^{best} - P_i(t)) \tag{1}$$

$$P_i(t+1) = v_i(t+1) + P_i(t) \tag{2}$$

Donde $v_i(t)$ representa la velocidad de la partícula en el instante $i$, $r_1$ y $r_2$ son números aleatorios en el rango de [0,1]. $P_i^{best}$ es la mejor posición definida por la partícula y $P_{global}^{best}$ es la mejor posición global de toda la población. $C_1$ y $C_2$ son las constantes de aceleración y representan respectivamente las constantes de atracción al mejor personal y mejor global. Para la implementación del algoritmo se deberá seguir el diagrama de flujo mostrado en la Figura 1.

## Parámetros

Las constantes $w$ (inercia), $C_1$ y $C_2$ afectan el desempeño del algoritmo y depende de estas que el algoritmo logre encontrar ya sea el máximo o el mínimo global. Uno de los propósitos de los dos coeficientes de aceleración aparte de determinar la atracción hacia las posiciones personal y global, es determinar el tipo de búsqueda, unos coeficientes cercanos a cero producirán una búsqueda fina en una región, mientras coeficientes cercanos a uno permitirán a la partícula una búsqueda mas amplia.

A partir del desarrollo de este método se han realizado una gran cantidad de trabajos para mejorar la versión original modificando los 3 factores de interés, utilizando la definición los coeficientes de constricción definida por Eberhart & Shi el cual utiliza los parámetros de *kappa*, *chi* y *phi*, para una buena configuración de los parámetros [5].

$$\phi = \phi_1 + \phi_2 \tag{3}$$

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{4}$$

$$w = \chi \tag{5}$$

$$c_1 = \chi * \phi_1 \tag{6}$$

$$c_2 = \chi * \phi_2 \tag{7}$$

# Ant Colony Optimization (ACO)

## Inspiración

La inspiración de este algoritmo viene del comportamiento natural en una colonia de hormigas, al rededor de los cuarentas y cincuentas del siglo 20, el entomólogo Pierre-Paul Graseé observo en una colonia de termitas lo que el denomino como "*Estímulos Significativos*". En sus observaciones pudo ver los efectos de las reacciones tanto en el insecto que produjo como en la colonia y utilizo el termino de "*estigmergía*"para describir este tipo de comunicación en la cual los trabajadores son estimulados por el desempeño que logran. En estas observaciones se determino que las hormigas poseen la habilidad de encontrar el camino mas corto entre la fuente de alimento y su colonia mediante el uso de feromonas.

A partir de estas observaciones en 1992 Marco Dorigo desarrollo un algoritmo con base al comportamiento de las hormigas denominado *Ant System (AS)*, a partir de este algoritmo se desarrollaron muchos híbridos hasta que en 1999 Dorigo junto a Di Caro and Gambardella basados en un método metaheurístico, en donde se ven las hormigas como base del algoritmo denominado ACO, para resolver problemas discretos de optimización [6].

## Funcionamiento

Cuando una hormiga encuentra alimento, esta deja un rastro de feromonas en el camino a la colonia, el resto de compañeras detectan este rastro y lo siguen ya que estas saben que encontraran alimento al final del camino, mientras mas hormigas pasan por el camino el rastro se hace cada vez mas fuerte. Este es un método eficiente ya que aunque existan dos caminos diferentes, mientras mas corto sea el camino, las hormigas que vienen detrás pueden detectar la feromonoas dejadas por sus compañeras mas rápido, aumentando las posibilidades que escogan el camino mas corto. Este comportamiento se ve reflejado en el experimento de "*doble puente"* realizado por Deneuborg, donde se colocan 2 caminos para

llegar a una fuente de alimento, en un principio los dos caminos tenían la misma distancia, al empezar el experimento el comportamiento de las hormigas fue aleatorio pero al final todas seleccionaron un solo camino, luego uno de los caminos se hizo mas corto que el otro, al igual que en el experimento anterior, el comportamiento de la colonia fue aleatorio, pero mientras pasaba el tiempo, los insectos escogieron el camino mas corto, a partir de este experimento se desarrollo un modelo dado por la Ecuación 8 [6].

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \tag{8}$$

Donde, $m_1$ y $m_2$ representan la cantidad de hormigas que han pasado por el camino 1 y el camino 2 respectivamente, $k$ y $h$ son parámetros que se ajustan mediante experimentación y $p_1$ representa la probabilidad que la hormiga escoga el camino 1.

## Algoritmos desarrollados

Existen diversas variantes para el algoritmo ACO, entre las mas exitosas son el *Ant System*, cuya principal característica es que en cada iteración los valores de feromonas son actualizados mediante la cantidad de hormigas $m$ que han construido una solución. El modelo que define la cantidad de feromoa $\tau_{ij}$ es el siguiente:

$$\tau_i j = (1 - \rho) * \tau_i j + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{9}$$

Donde, $\rho$ es la taza de evaporación de la feromona, $m$ es el numero de hormigas y $\Delta\tau_{ij}^k$ es la cantidad de feromona que hay en las aristas.

$$\Delta\tau_{ij}^k = \left\{ \begin{array}{r} \frac{Q}{L_k},\ \text{k utilizo aristas (i,j)} \\ 0,\ \text{otro caso} \end{array} \right\} \tag{10}$$

Donde, $Q$ es una constante y $L_k$ es la longitud del recorrido construido por la hormiga k.

$$P_{i,j}^k = \left\{ \begin{array}{r} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum \tau_{il}^\alpha \eta_{il}^\beta}, \text{sic}_{ij} \in N(s^p) \\ 0,\ \text{otro caso} \end{array} \right\} \tag{11}$$

Donde, $N(s^p)$, es el set de componentes factibles, es decir las aristas (i,l) donde l son las aristas que no ha sido visitado por la hormiga k. Los parámetros $\alpha$ y $\beta$ controlan la importancia de la feromona dada por la información heurística $\eta_{ij}$ que esta dada por:

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{12}$$

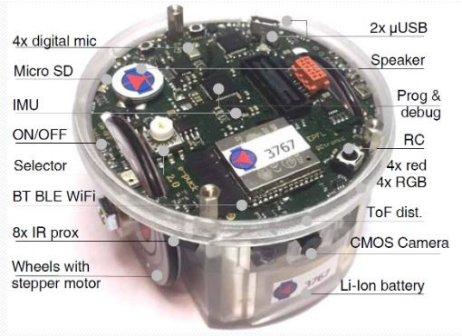



Figura 2: Sensores del robot E-Puck [7].

Donde, $d_{ij}$ es la distancia entre la arista i y j [6]. .

## E-Puck

El E-Puck es un robot diferencial móvil con llantas desarrollado por Dr. Fancesco Mondada y Michel Bonani en el 2006 en *EPFL, (Federal Swiss Institute of Technology in Lausanne.* El robot consiste de dos llantas con actuadores que permiten girar en ambas direcciones para cambiar la dirección del robot. Además tiene diversos sensores que permiten hacer mediciones muy buenas de distancia con objetos a su alrededor. El robot utiliza un procesador dsPIC que funciona como un microcontrolador [1].

## Raspberry Pi

La Raspberry Pi es un ordenador pequeño y de bajo coste, al cual se puede conectar un monitor y un teclado para interactuar con ella como cualquier otra computadora. Fue desarrollado como una organización caritativa de la Fundación Raspberry Pi en 2009 cuyo objetivo era animar a los niños a aprender informática en las escuelas.

Con Raspberry Pi esto es mucho más sencillo y abre las puertas de la experimentación y el aprendizaje en distinetos ambitos. Puede usarlo para aprender a programar, crear proyectos de electrónica, y para muchas de las cosas que hace cualquier PC de escritorio, como hojas de cálculo, procesamiento de texto, navegar por Internet y jugar ciertos videojuegos. También reproduce videos de alta definición. La Raspberry Pi está siendo utilizada para aprender de programación y creación digital [8].

La Raspberry Pi es la placa de un ordenador simple compuesto por un SoC, CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, reloj, una toma para la alimentación, conexiones para periféricos de bajo nivel, reloj. Se tiene que conectar conectar periféricos de entrada y salida para poder interactuar con la Raspberry, instrumentos como un monitor, un ratón y un teclado y grabar un sistema operativo para Raspberry Pi en la tarjeta SD.

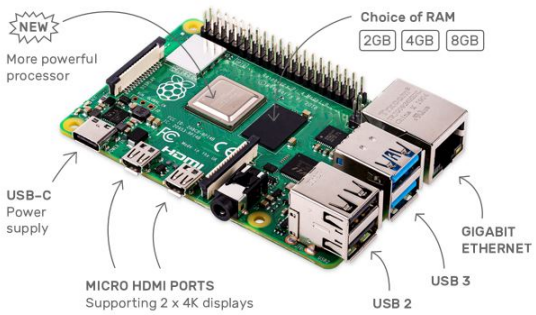



Figura 3: Raspberry Pi [8].

## Metodología

Para evaluar las distintas opciones de robots, el microcontrolador y lenguaje de programación a utilizar para la implementación del algoritmo, se propone:

- Investigar plataformas y lenguajes de programación para la implementación, esto para facilitarnos la tarea de conseguir las herramientas necesarias.
- Luego de investigar las distintas opciones que se encuentre, determinar el microcontrolador a utilizar.
- Con el microcontrolador escogido, determinar el lenguaje de programación a utilizar.
- Al escoger las herramientas a utilizar, se procederá a familiarizarse con el entorno escogido, realziando diversas pruebas para aprender a programar en la plataforma.

Para migrar el algoritmo ACO desarrollado anteriormente por Gabriela Iriarte [4], se propone:

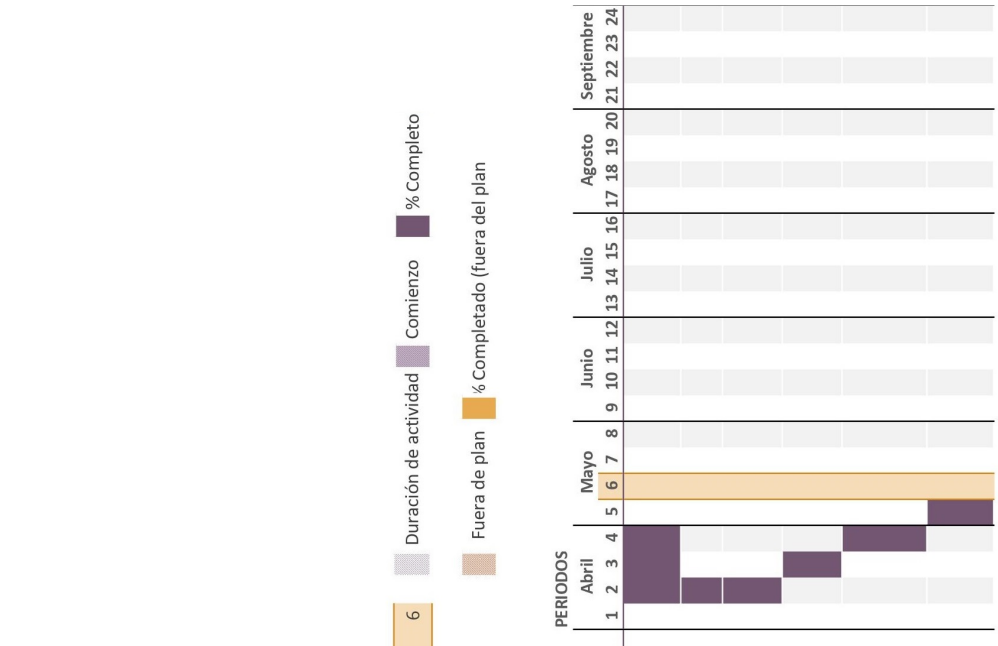
- Sacar medidas de la mesa de pruebas para la implementación y ajustar el código ya hecho a las nuevas medidas.
- Comenzar implementando un algoritmo mas sencillo como lo es el Simple Ant Colony Optimization.
- Luego al ya tener la experiencia del algoritmo mas simple, implementar el Ant System.
- Al lograr migrar el algoritmo, se implementaran los diferentes métodos de control a utilizar ya que se planea utilizar varios para lograr un mejor rendimiento.

Para validar la migración de los algoritmos al controlador, se propone:

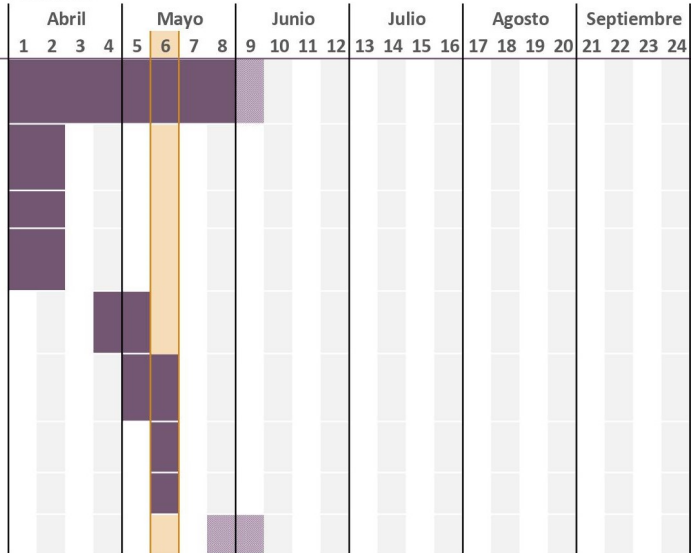
- Simular un método de comunicación entre las "hormigas"para empezar a probar el funcionamiento.

- Comenzar a realzar distintos experimentos en mesa de pruebas, como lograr llegar a una meta especifica o recibir la información de las "feromonasz procesarla.
- Al tener todo funcionando, implementar comunicación entre dos o mas controladores y que estos envíen y reciban información exitosamente.

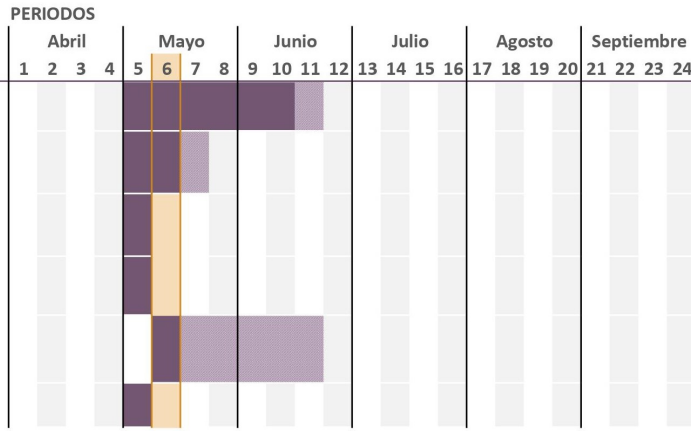
## Cronograma de actividades



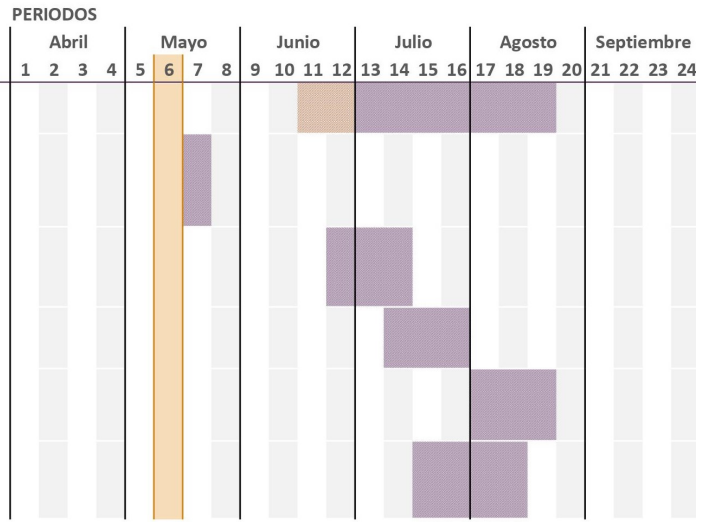
| | ACTIVIDAD | INICIO | DURACIÓN | INICIO REAL | DURACIÓN REAL | PORCENTAJE COMPLETADO |
|---|---|---|---|---|---|---|
| 2 | **Elaboración del Protocolo** | 1 | 9 | 1 | 9 | 90% |
| 2.1 | Elaboración del marco teórico | 1 | 2 | 1 | 2 | 100% |
| 2.2 | Redacción de objetivos | 1 | 2 | 1 | 2 | 100% |
| 2.3 | Redacción de los antecedentes | 1 | 2 | 1 | 2 | 100% |
| 2.4 | Redacción de la justificación | 4 | 2 | 4 | 2 | 100% |
| 2.5 | Redacción de la metodología | 5 | 2 | 5 | 2 | 100% |
| 2.6 | Elaboración del cronograma | 6 | 1 | 6 | 1 | 100% |
| 2.7 | Redacción del resumen | 6 | 1 | 6 | 1 | 100% |
| 2.8 | Corrección del Protocolo | 8 | 2 | 8 | 2 | 0% |



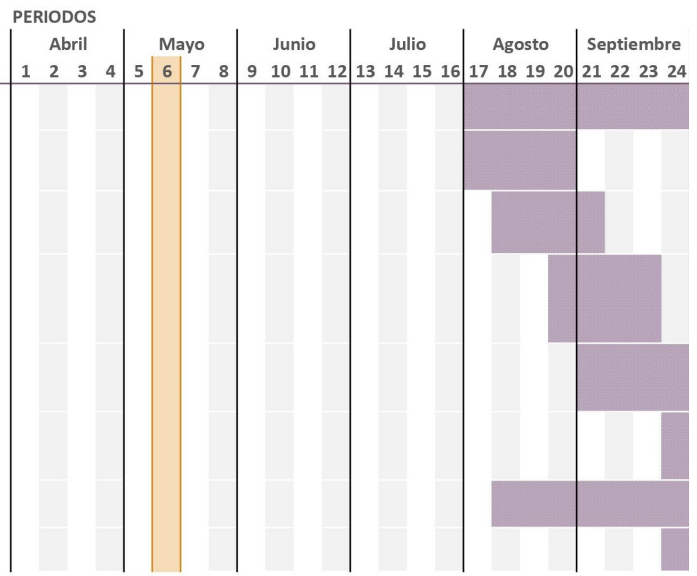
| | ACTIVIDAD | INICIO | DURACIÓN | INICIO REAL | DURACIÓN REAL | PORCENTAJE COMPLETADO |
|---|---|---|---|---|---|---|
| 3 | Herramienta a utilizar | 5 | 7 | 5 | 7 | 90% |
| 3.1 | Investigar plataformas y lenguajes accesibles | 5 | 3 | 5 | 3 | 80% |
| 3.2 | Determinar el controlador a utilizar | 5 | 1 | 5 | 1 | 100% |
| 3.3 | Determinar el lenguaje de programación | 5 | 1 | 5 | 1 | 100% |
| 3.4 | Familiarizarse con el entorno escogido | 6 | 6 | 6 | 6 | 20% |
| 3.5 | Realizar una cotización. | 5 | 1 | 5 | 1 | 100% |



| | ACTIVIDAD | INICIO | DURACIÓN | INICIO REAL | DURACIÓN REAL | PORCENTAJE COMPLETADO |
|---|---|---|---|---|---|---|
| 4 | **Migrar el algoritmo ACO** | 13 | 7 | 11 | 9 | 0% |
| 4.1 | **Sacar medidas de la mesa de pruebas para la implementación** | 7 | 1 | 7 | 1 | 0% |
| 4.2 | **Implementar el Simple Ant Colony Optimization** | 12 | 3 | 12 | 3 | 0% |
| 4.3 | **Implementar el Ant System** | 14 | 3 | 14 | 3 | 0% |
| 4.4 | **Perfeccionar los algoritmos** | 17 | 3 | 17 | 3 | 0% |
| 4.5 | **Implementar los diferentes métodos de control** | 15 | 4 | 15 | 4 | 0% |



| | | ACTIVIDAD | INICIO | DURACIÓN | INICIO REAL | DURACIÓN REAL | PORCENTAJE COMPLETADO |
|---|---|---|---|---|---|---|---|
| 5 | | **Validar los resultados** | 17 | 8 | 17 | 8 | 0% |
| | 5.1 | Simular el método de comunicación | 17 | 4 | 17 | 4 | 0% |
| | 5.2 | Implementar en mesa de pruebas. | 18 | 4 | 18 | 4 | 0% |
| | 5.3 | Implementar comunicación entre dos controladores | 20 | 4 | 20 | 4 | 0% |
| | 5.4 | Determinar los mejores métodos de control | 21 | 4 | 21 | 4 | 0% |
| 6 | | Evaluar implementación de algoritmos genéticos | 24 | 1 | 24 | 1 | 0% |
| 7 | | Redacción de tesis | 18 | 7 | 18 | 7 | 0% |
| 8 | | Entrega de tesis | 24 | 1 | 24 | 1 | 0% |



## Índice preliminar

## Referencias


[1] L. Marín, M. Vallés, Á. Valera y P. Albertos, "Implementation of a bug algorithm in the e-puck from a hybrid control viewpoint," *2010 15th International Conference on Methods and Models in Automation and Robotics, Miedzyzdroje, Poland*, págs. 174-179, 2010.

[2] A. Aguilar, "Algoritmo Modificado de Optimización de Enjambre de Particulas (MPSO)," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2019.

[3] E. Santizo, "Aprendizaje Reforzado y Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2020.

[4] G. Iriarte, "Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2020.

[5] R. C. Eberhart e Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," *Proceedings of the 2000 Congress on Evolutionary Computation (Cat. No.00TH8512), La Jolla, CA, USA*, vol. 1, n.º doi: 10.1109/CEC.2000.870279. Págs. 84-88, 2000.

[6] M. Dorigo, M. Birattari y T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, n.º 4, doi: 10.1109/MCI.2006.329691. Págs. 28-39, 2006.

[7] GCtronic, *E-Puck Education Robot*, `http://www.e-puck.org/`, Accessed: 2018-02-23, 2018.

[8] Raspberry Pi Foundation, *Raspberry Pi documentation*, `https://www.raspberrypi.org/documentation/faqs/#introduction`, 2020.