UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Aplicación de Herramientas de Aprendizaje Reforzado y Aprendizaje Profundo en Simulaciones de Robótica de Enjambre con Restricciones Físicas

Protocolo de trabajo de graduación presentado por Marco Antonio Izeppi Rosales, estudiante de Ingeniería Mecatrónica

Guatemala,

2021

Resumen

El algoritmo de Optimización por enjambre de particulas (Particle Swarm Optimization) hace referencia a un método heurístico para resolver un tipo de problema computacional que evoca el comportamiento de las partículas en la naturaleza. En un principio fueron concebidos para elaborar modelos de conductas sociales, como el movimiento descrito por los organismos vivos en una bandada de aves, enjambres de insectos o un banco de peces. Posteriormente el algoritmo se simplificó y se comprobó que era adecuado para problemas de optimización utilizando partículas que simulan un ser individual dentro del enjambre.

El algoritmo PSO posee diversas variantes aplicadas a diversos objetivos tales como optimizacion 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.

Tomando esto en consideración, en las fases previas a este proyecto se desarrolló un algoritmo que utiliza métodos de *machine learning* 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.

En este protocolo se propone un trabajo de graduación que busca una implementación de este algoritmo, previamente diseñado, en un entorno de simulación con restricciones físicas, que tome en cuenta las dimensiones del robot y que sea capaz de evadir obstáculos colocados dentro del entorno de simulación. Este algoritmo ya implementado en un sistema físico busca una rápida convergencia hacia el mínimo global al igual que evitar colisiones entre los robots o con los obstáculos. Por otro lado, se busca evaluar la viabilidad de la realización de simulaciones de sistemas multi-robot en el entorno de *Robotic Operating System* (ROS) debido a que es un *software* popular en el área de robótica y presenta buenas simulaciones de condiciones físicas.

Antecedentes

Optimización de trayectorias de robots móviles en ambientes desconocidos

En este artículo Safa Ziadi, Mohamed Njah y Mohamed Chtourou [1] Utilizaron el enfoque de optimización de enjambre de partículas multiobjetivo (PSO) para optimizar los parámetros del Método Canónico de Campo de Fuerza (CF2) los cuales son P, c,k,Q y ρ_0. El cálculo de los parámetros óptimos se reinicia en cada nueva posición del robot y el PSO se utiliza para minimizar la distancia entre la posición y la meta a la cual se desea llegar y también para maximizar la distancia segura entre la posición actual del robot y los obstáculos cercanos. La eficacia del método se demuestra mediante simulaciones en el entorno de Webots. Las simulaciones se llevan a cabo en varios entornos conocidos y desconocidos. En los entornos conocidos, el robot reconoce la posición del obstáculo al comienzo de la navegación y la planificación de la ruta es global. Pero en los entornos desconocidos, la localización del robot se basa en las lecturas de los sensores y la planificación de la ruta es individual para cada robot, dependiendo de las lecturas de los sensores.

Megaproyecto Robotat

El Megaproyecto Robotat es un proyecto que tenía como objetivo el diseño de robots con aplicaciones de seguimiento de trayectorias y comportamientos de enjambre. A partir de esta idea se realizó el diseño electrónico y mecánico de los "Bitbots". Inicialmente se enfocaron en el diseño de *hardware* de los "Bitbots" y en un algoritmo de visión por computadora para obtener la posición y orientación de los robots en un plano bidimensional. Mas adelante, se empezaron a refinar detalles de *software* tales como el protocolo de comunicación y el algoritmo computacional que sera utilizado por los "Bitbots". Para el desarrollo del algoritmo de control para determinar el comportamiento de los robots se fueron desarrollando diversas tesis mostradas a continuación:

Algoritmo Modificado de Optimización de Enjambre de Partículas

En [2] Aldo Aguilar busca la implementación de Algoritmo Modificado de Optimización de Enjambre de Partículas con robots diferenciales reales. En el desarrollo de esta tesis se buscaba definir un algoritmo para establecer el comportamiento de los robots para que se comporten como un enjambre con la habilidad de búsqueda de metas. 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 que se saturen los actuadores del robot.

Para solucionar este problema se planteó que cada robot no iría directamente a al 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. Durante este proceso se realizaron pruebas con 8 controladores diferentes, los cuales son: 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). Al finalizar las pruebas con estos 8 controladores se determinó que los mejores resultados en cuanto a la generación de trayectorias y velocidades continuas reguladas para los robots diferenciales era el controlador TUC-LQI y el TUC-LQR.

Algoritmo de optimización de enjambre y Artificial Potential Fields

En [3] Juan Cahueque realiza 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.

Teniendo esto bien definido, se definen funciones artificiales de potencia de Choset y Kim, Wang y Shin 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. Se realizaron tres modelos, cada uno contaba con una meta y diversos obstáculos, los cuales tena que evitar el robot para llegar a la meta.

Para lograr que estos modelos fueran implementables en robots diferenciales físicos. se emplearon los controladores propuestos por Aldo Aguilar [2] con algunas modificaciones para hacerlos compatibles con esta nueva metodología. Realizando diversas simulaciones se llego a determinar los mejores parámetros para los controladores en los robots diferenciales físicos. Cabe destacar que se tuvieron que realizar diversas pruebas con Webots, el cual es un software enfocado en simulaciones de robótica con restricciones físicas. Estas pruebas mostraron que los parámetros establecidos en Matlab eran idealizados y al ser implementados en Webots no brindaban el comportamiento esperado para el comportamiento del robot.

Algoritmo PSO modificado con aprendizaje reforzado y aprendizaje profundo

En [4] Eduardo Santizo Empleo la metodología de aprendizaje reforzado y aprendizaje profundo para desarrollar un *PSO tuner* y un generador de trayectorias. Para el desarrollo del PSO tuner, se empleo una red neuronal recurrente 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 7,700 simulaciones de un algoritmo estándar de PSO se generan predicciones de carácter dinámico, Causando que el algoritmo final redujera el tiempo de convergencia y susceptibilidad a mínimos locales del PSO original.

Para el generador de trayectorias se utilizó aprendizaje profundo. En las pruebas iniciales se utilizó una cuadrícula simulando el *gridworld* de Webots en el cual el robot solo tenia la posibilidad de moverse en 4 direcciones: arriba, abajo, izquierda o derecha. Al robot se le daban penalizaciones si chocaba contra 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. Luego se modificó el algoritmo para permitir el movimiento en diagonales a 45°, Por lo que se tienen 8 posibles movimientos y se siguió la misma metodología anteriormente descrita. Obteniendo finalmente el modelo propuesto por Eduardo Santizo [4]. 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 con restricciones físicas.

Justificación

El algoritmo de PSO nos permite optimizar un problema (función de costo) a partir de una población de soluciones candidatas o 'partículas' que se mueven por todo el espacio de búsqueda según diversas reglas matemáticas que tienen en cuenta la posición y la velocidad en ese instante de las partículas. El movimiento de cada partícula se ve influido por la mejor posición que ha encontrado hasta el momento, así como por las mejores posiciones globales encontradas por las demás partículas a medida que recorren el espacio de búsqueda.

En la fase previa, Se desarrolló un algoritmo que se 'adapta' a la función de costo para obtener los parámetros óptimos y brindar un mejor comportamiento de las partículas para lograr una convergencia rápida y efectiva hacia el mínimo o máximo del a función. Sin embargo, no se han realizado pruebas en entornos físicos tomando en consideración los controladores a utilizar y la validación del comportamiento esperado por parte de los robots con condiciones físicas.

Debido a esto, en este proyecto se busca la integración de simuladores de condiciones físicas junto con el algoritmo adaptado con aprendizaje profundo y aprendizaje reforzado para la optimización de funciones de costo para la obtención de los parámetros óptimos en un entorno estático. Se busca verificar y validar los resultados obtenidos en entornos físicos con los resultados obtenidos a partir de simulaciones ideales del comportamiento de los robots, sentando las bases para futuras experimentaciones que resuelvan cada vez problemas más complejos brindando resultados satisfactorios.

Se busca evaluar la viabilidad de la simulación de dicho algoritmo en el entorno de ROS debido a que es un *software* popular en el área de robótica por sus buenas simulaciones de condiciones físicas como lo son la fricción, la interacción de las fuerzas, etc. Además, cuenta con diversos modelos de robots, actuadores y sensores los cuales son actualizados por las mismas empresas o algunos diseñadores para brindar un simulador muy completo.

Tomando esto en consideración, se puede concluir que es un *software* de gran utilidad para evaluar el comportamiento de robots en condiciones físicas y definir controladores apropiados que nos brinden el comportamiento esperado. De ser viable, se busca dejar modelos de ejemplo en los diferentes entornos para que llegue a ser utilizado en cursos dentro de la Universidad del Valle de Guatemala y así lograr que las futuras generaciones de graduandos tengan un entendimiento mas profundo sobre los diferentes entornos que existen y cuales son las ventajas y desventajas que presenta cada uno de ellos.

Objetivos

Objetivo General

Integrar las herramientas de *software PSO Tuner* y *Swarm Robotics Toolbox*, desarrolladas en la fase anterior, a plataformas de simulación con restricciones físicas.

Objetivos Específicos

- Evaluar entornos de simulación con motores de física que permitan simulaciones de robótica de enjambre más realistas.
- Realizar simulaciones en el entorno seleccionado y obtener datos de entrenamiento para el *PSO Tuner* y *Swarm Robotics Toolbox*.
- Utilizar los parámetros obtenidos por el *PSO Tuner* y el *Swarm Robotics Toolbox*, luego del entrenamiento, en nuevos escenarios, y verificar el rendimiento del sistema.
- Evaluar la viabilidad del uso de ROS para aplicaciones de robótica de enjambre en combinación con las herramientas *PSO Tuner* y *Swarm Robotics Toolbox*.

Marco teórico

Particle Swarm Optimization (PSO)

La Optimización por Enjambres de Partículas (conocida como PSO, por sus siglas en inglés, Particle Swarm Optimization) es una técnica de optimización/búsqueda. Este método fue descrito alrededor de 1995 por James Kennedy y Russell C. Eberhart (Kennedy, J. & Eberhart, R. (1995), 'Particle swarm optimization', Neural Networks, 1995. Proceedings., IEEE International Conference), y se inspira en el comportamiento de los enjambres de insectos en la naturaleza. Formalmente hablando, se supone que tenemos una función desconocida, $f(x, y)$, que podemos evaluar en los puntos que queramos, pero a modo de caja negra, por lo que no podemos conocer su expresión. El PSO es fácil de implementar y ha sido ampliamente usado en variadas aplicaciones con resultados excelentes resolviendo problemas reales de optimización. Este algoritmo puede ser computacionalmente ineficiente por que puede quedar atrapado fácilmente en óptimos locales cuando resuelve problemas cuyo espacio de solución es multimodal; estas debilidades han hecho que el campo de aplicación de la metodología esté un poco restringido [5].

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, o bien verifica alguna relación extremal respecto a alguna otra función). La idea que vamos a seguir en PSO comienza de forma similar, 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 [5].

Cada partícula (individuo) tiene una posición, p (que en 2 dimensiones vendrá determinado por un vector de la forma $[x, y]$), en el espacio de búsqueda y una velocidad, v (que en 2 dimensiones vendrá determinado por un vector de la forma $[v_x, v_y]$), 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 (cambio de velocidad), que depende principalmente de dos características:

- Cada partícula es atraída hacia la mejor localización que ella, personalmente, ha encontrado en su historia (mejor personal).
- Cada partícula es atraída hacia la mejor localización que ha sido encontrada por el conjunto de partículas en el espacio de búsqueda (mejor global).

Formalmente, lo podemos escribir como:

$$v_i(t+1) = v_i(t) + C_1 r_1(p_i^{mejor} - p_i(t)) + c_2 r_2(p_g^{mejor} - p_i(t)) \tag{1}$$

En donde:

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

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

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

un conjunto de partículas representa potenciales soluciones, donde cada partícula i está asociada a dos vectores, el vector de velocidades y a la posición del vector. La velocidad y la posición de cada partícula son inicializadas por vectores aleatorios con sus correspondientes rangos [6].

Aprendizaje Profundo

En el enfoque Aprendizaje Profundo se usan estructuras lógicas que se asemejan en mayor medida a la organización del sistema nervioso de los mamíferos, teniendo capas de unidades de proceso (neuronas artificiales) que se especializan en detectar determinadas características existentes en los objetos percibidos. La visión artificial es una de las áreas donde el Aprendizaje Profundo proporciona una mejora considerable en comparación con algoritmos más tradicionales. Existen varios entornos y bibliotecas de código de Aprendizaje Profundo que se ejecutan en las potentes GPUs modernas tipo CUDA, como por ejemplo NVIDIA cuDNN [7].

El Aprendizaje Profundo representa un acercamiento más íntimo al modo de funcionamiento del sistema nervioso humano. Los modelos computacionales de Aprendizaje Profundo imitan estas características arquitecturales del sistema nervioso, permitiendo que dentro del sistema global haya redes de unidades de proceso que se especialicen en la detección de determinadas características ocultas en los datos. El Aprendizaje Profundo lleva a cabo el proceso de *Machine Learning* usando una red neuronal artificial que se compone de un número de niveles jerárquicos. En el nivel inicial de la jerarquía la red aprende algo simple y luego envía esta información al siguiente nivel. El siguiente nivel toma esta información sencilla, la combina, compone una información algo un poco más compleja, y se lo pasa al tercer nivel, y así sucesivamente. Destaca porque no requiere de reglas programadas previamente, sino que el propio sistema es capaz de "aprender´´ por sí mismo para efectuar una tarea a través de una fase previa de entrenamiento [7].

Los algoritmos que componen un sistema de aprendizaje profundo se encuentra en diferentes capas neuronales compuestas por pesos (números). El sistema está dividido principalmente en 3 capas:

- **Capa de entrada (Input Layer):** Está compuesto por las neuronas que asimilan los datos de entrada, como por ejemplo imagen o una tabla de datos.
- **Capa oculta (Hidden Layer):** Es la red que realiza el procesamiento de información y hacen los cálculos intermedios. Cada más neuronas en esta capa haya, más complejos son los cálculos que se efectúan.
- **Salida (Output Layer):** Es el último eslabón de la cadena, y es la red que toma la decisión o realiza alguna conclusión aportando datos de salida.

Aprendizaje Reforzado

El Aprendizaje por refuerzo es un área del aprendizaje automático inspirada en la psicología conductista, cuya ocupación es determinar qué acciones debe escoger un agente de software en un entorno dado con el fin de maximizar alguna noción de recompensa.º premio acumulado. Intenta conseguir que una inteligencia artificial aprenda a decidir mediante su propia experiencia. Es decir, que ante una situación determinada, sea capaz de seleccionar por sí misma la mejor acción a ejecutar en ese momento mediante un proceso interactivo de prueba y error a base de reforzar positivamente cada vez que se aproxima o logra objetivo. Por eso, con el Aprendizaje reforzado una máquina puede tomar decisiones aunque no almacene un conocimiento a priori del entorno o de las variables que se están dando, y realizar de manera satisfactoria cuestiones abstractas más avanzadas[8].

La aplicación de ese aprendizaje les permite ya reconocer caras, clasificar secuencias de ADN, conducir vehículos, o hacer diagnósticos médicos. En la actualidad compañías tecnológicas punteras como Google, Apple o IBM, están invirtiendo en investigación para entrenar robots que realicen sencillas tareas mediante esta técnica. El aprendizaje por refuerzo es especialmente adecuado para los problemas que incluyen un razonamiento a largo plazo frente a uno a corto plazo. Se ha aplicado con éxito a diversos problemas, entre ellos el control de robots, telecomunicaciones, backgammon y damas [8].

Dos componentes hacen aprendizaje por refuerzo de gran alcance: El uso de muestras para optimizar el rendimiento y el uso de la función de aproximación para hacer frente a entornos de gran tamaño. El problema de aprendizaje reforzado requiere mecanismos de exploración inteligente. Seleccionar al azar acciones, sin hacer referencia a una distribución de probabilidad estimada, que se conoce para dar lugar a un rendimiento muy pobre [8].

Webots

Webots es una Multi-plataforma Open-Source (Código abierto) utilizada para realizar simulaciones de robots en entornos mas cercanos a la realidad. Provee un entorno completo de desarrollo para modelar, programar y simular robots. Esta plataforma fue desarrollada por el Dr. Oliver Michel del Instituto Federal Suizo de Tecnología para fines educativos. Utiliza el Open Dynamics Engine que Permite la simulación de colisiones y la dinámica de cuerpos rígidos creando así un entorno mas real. El programa permite utilizar controladores escritos en C, C++, Java, Python, MATLAB y ROS [9].

Robot Diferencial E-Puck

El E-Puck es un robot diferencial móvil desarrollado por la Escuela Politécnica Federal de Lausana (EPFL). Este robot esta incluido en la librería de Webots para realizar simulaciones. Este robot cuenta con una gran variedad de sensores que lo hacen una buena opción para su implementación con el algoritmo PSO. Es un robot pequeño y su programación es amigable con el usuario, por lo cual ha sido utilizado como herramienta de aprendizaje [10].

Robot Operating System (ROS)

Robot Operating System (ROS) es una colección de marcos para el desarrollo de software de robots. ROS se desarrolló originariamente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR2). Desde 2008, el desarrollo continuó principalmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado [11].

Los procesos ROS se representan como nodos (ROS NODES) en una estructura gráfica, conectados por bordes llamados ROS topics. [65] Los ROS NODES pueden pasar mensajes entre sí a través de los ROS topics, realizar llamadas de servicio a otros nodos, proporcionar un servicio para otros nodos o establecer o recuperar datos compartidos de una base de datos común llamada servidor de parámetros. Un proceso llamado ROS Master hace que todo esto sea posible al registrar nodos para sí mismo, configurar la comunicación de nodo a nodo para los temas y controlar las actualizaciones del servidor de parámetros [12].

A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del *hardware*, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lu-

gar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros [12].

La librería está orientada para un sistema UNIX (Ubuntu -Linux-). ROS tiene dos partes básicas: la parte del sistema operativo (ros) y Ros-pkg. Ros-pkg consiste en un conjunto de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados pilas o en inglés stacks) que implementan las funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc. Debido a esto es un sistema bastante completo para la realización de diversas simulaciones implementando diversos componentes los cuales nos brindan resultados muy confiables y similares a lo que se vería en la vida real [11].

Metodología

El primer paso a realizar consiste en tomar un algoritmo básico de un algoritmo PSO tomando como referencia el algoritmo realizado por Eduardo Santizo [4] e implementarlo en Webots (simulador con restricciones físicas), sin colocar ningún obstáculo. Esto nos sera de gran utilidad para obtener parámetros base que nos brinden un punto de referencia a la hora de verificar la velocidad de convergencia y comportamiento del algoritmo PSO modificado con deep learning.

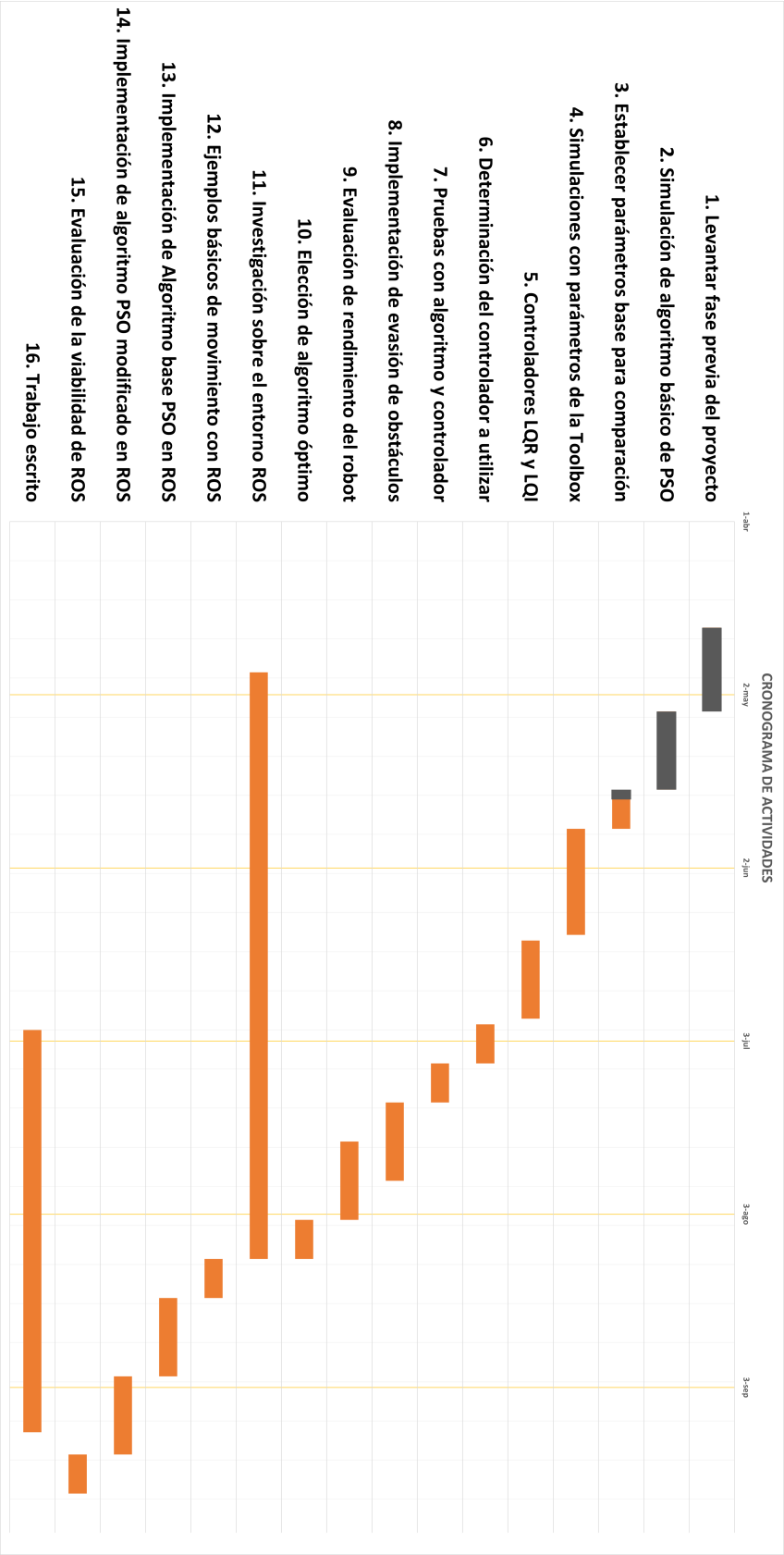
Se iniciar'a con simulaciones sencillas realizando pequeñas modificaciones al algoritmo priorizando el correcto funcionamiento del mismo y validando que en cada prueba el algoritmo converja al punto deseado. Este es un proceso que tomara tiempo debido a que el algoritmo PSO es sensible a los pequeños cambios dentro de su programación al igual que en los valores (constantes) que entran al sistema.

Terminada la fase anterior, se empezarán a agregar obstáculos modificando el algoritmo para que sea capaz de reconocer la ubicación de estos obstáculos y evadirlos de forma certera. Cabe destacar que el robot debe tomar en consideración sus propias dimensiones y la posición de los otros robots para evitar una colisión entre los mismos.

Ya teniendo un algoritmo funcional en el simulador de Webots, se evaluará la viabilidad de la implementación de dicho algoritmo dentro de el entorno de ROS. Se iniciarán con programas base para luego tratar de implementar un algoritmo PSO funcional en el entorno de ROS. Luego del desarrollo del algoritmo PSO se buscará la implementación del algoritmo de *Machine Learning* para verificar el comportamiento que tiene el robot. Se realizará una comparación entre los resultados, tanto de algoritmo PSO básico como del algoritmo modificado con aprendizaje profundo y aprendizaje reforzado, lo cual nos permitirá evaluar el alcance de nuestro algoritmo y verificar si el comportamiento es similar en ambos casos o si presentan alguna discrepancia.

En el caso de la programación en Webots, será realizada en lenguaje C para la facilidad de implementación en la mayoría de sistemas embebidos ya que es el lenguaje mas común para su implementación. En el caso de la programación en ROS se deberá realizar una migración de lenguaje C a Python, C++ o Lisp debido a que son los lenguajes con los cuales trabaja el entorno de ROS. En el caso del robot que se utilizará para las simulaciones dentro del entorno de ROS, se deberá encontrar un Robot con capacidades similares a las del E-puck en Webots. Esto se debe a que el modelo del E-puck no esta disponible dentro de la librería de ROS. Finalmente se deberá evaluar inicialmente la implementación de un algoritmo PSO básico en el entorno de ROS para determinar si el sistema es viable para la implementación del algoritmo modificado.

Cronograma de actividades



1. *Levantar Fase previa del proyecto:* En esta etapa se busca levantar y entender el funcionamiento al igual que como se compone la fase previa del proyecto.

2. *Simulación de algoritmo básico de PSO:* En esta etapa se busca realizar simulaciones con algoritmos básicos de PSO sin ningún tipo de optimización.

3. *Establecer Parámetros base para comparación:* A partir de las simulaciones de la etapa anterior, se busca determinar y establecer parámetros base de comparación.

4. *Simulaciones con parámetros de la toolbox:* Realizar simulaciones utilizando los parámetros y algoritmos brindados por la toolbox.

5. *Controladores LQI y LQR:* Realización de simulaciones utilizando controladores LQI y LQR para determinar velocidades de convergencia

6. *Determinación del controlador a utilizar:* A partir de las simulaciones anteriores, se determina el mejor controlador para el algoritmo.

7. *Pruebas con algoritmo y controlador:* Realización de pruebas con el algoritmo optimizado y el controlador seleccionado.

8. *Implementación de evasión de obstáculos:* Desarrollar un sistema de evasión de obstáculos ya con el algoritmo optimizado y el controlador seleccionado.

9. *Evaluación de rendimiento del Robot:* Se evalúa el comportamiento del robot y su capacidad de evadir efectivamente los obstáculos que se le presentan.

10. *Elección de algoritmo óptimo:* A partir de los resultados, se escoge la mejor combinación de parámetros.

11. *Investigación sobre el entorno ROS:* Investigar sobre el sistema ROS y su funcionamiento

12. *Ejemplos básicos de movimiento con ROS:* Realizar simulaciones básicas de movimiento en el entorno ROS para dar un primer vistazo de como funcionan.

13. *Implementación de Algoritmo base PSO en ROS:* Implementación de un algoritmo básico de PSO dentro del entorno de ROS.

14. *Implementación de algoritmo PSO modificado en ROS:* De ser posible, implementar el algoritmo modificado dentro del entorno de ROS.

15. *Evaluación de viabilidad de ROS:* Evaluar que tan viable es la implementación de algoritmos PSO dentro de el entorno de ROS.

16. *Trabajo escrito:* Redacción del documento de tesis

Índice preliminar

No. Página

Referencias

[1] S. Ziadi, M. Njah y M. Chtourou, «PSO optimization of mobile robot trajectoriesin unknown environments,» *International Multi-Conference on Systems, Signals & Devices*, vol. 13, págs. 774-782, 2016.

[2] A. Aguilar, «Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO),» Tesis de licenciatura, Universidad del Valle de Guatemala, 2019.

[3] J. Cahueque, «Implementación de enjambre de robots en operaciones de búsqueda y rescate,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2019.

[4] E. Santizo, «Aprendizaje Reforzado y Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[5] D. Álvarez, E. Toro y G. Ramón, «Algoritmo de Optimización cumulo de partículas aplicado en la solución de problemas de empaquetamiento óptimo bidimensional con y sin rotación,» *Scientia et Technica*, págs. 10-16, 2009.

[6] Z. Chi, G. Hai-bing, G. Liang y Z. Wan-guo, «Particle Swarm Optimization(PSO) Algorithm,» Tesis de mtría., Huazhong University of Science & Technology, China, 2010.

[7] A. Gonzáles, «Aplicaciones de técnicas de inteligencia artificial basadas en aprendizaje profundo (deep learning) al análisis y mejora de la eficiencia de procesos industriales,» Tesis de mtría., Universidad de Oviedo, España, feb. de 2018.

[8] E. Morales, «Aprendizaje por Refuerzo,» *Instituto Nacional de Astrofísica, Óptica y Electrónica.*, págs. 3-12, 2011.

[9] Cyberbotics Ltd, *Webots*, `https://cyberbotics.com`, 2021.

[10] C. Cianci, X. Raemy, J. Pugh y A. Martinoli, «Communication in a swarm of miniature robots: The e-puck as an educational tool for swarm robotics,» *International Workshop on Swarm Robotics*, págs. 103-115, 2006.

[11] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler y A. Ng, «ROS: an open-source Robot Operating System,» *Computer Science Department, University of Southern California*, págs. 1-6, 2009.

[12] T. Foote, «tf: The transform library,» en *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ép. Open-Source Software workshop, abr. de 2013, págs. 1-6. DOI: `10.1109/TePRA.2013.6556373`.