UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Reinforcement y Deep Learning en Aplicaciones de Robótica de Enjambre

Protocolo de trabajo de graduación presentado por Eduardo Santizo,
estudiante de Ingeniería Mecatrónica

Guatemala,

2020

Resumen

El algoritmo de Particle Swarm Optimization (PSO) consiste de un algoritmo de optimización estocástico, nacido a partir de la modificación de un algoritmo de simulación de parvadas. Cuando sus creadores tomaron dicho algoritmo y le retiraron las restricciones de proximidad de las "aves", se percataron que las entidades simuladas (Llamadas partículas) se comportaban conjuntamente como un optimizador.

Luego de la publicación de esta investigación, una gran cantidad de académicos han notado el potencial del método y se han dado a la tarea de mejorarlo y/o utilizarlo como base para nuevos avances. Este es el caso del mega proyecto *Robotat* de la Universidad del Valle de Guatemala, donde el algoritmo se propuso como la base para el sistema de navegación que emplearían los robots diferenciales a utilizar, denominados *Bitbots*.

En particular, se consiguió implementar una modificación del PSO que no solo permitía respetar las limitaciones físicas de los *Bitbots*, sino que también era capaz de esquivar obstáculos. No obstante, los resultados obtenidos eran altamente dependientes de múltiples parámetros específicos que fueron programados manualmente en los robots en base a conocimiento a priori sobre la situación que enfrentarían.

¿Existirá una forma de automatizar el proceso de selección de estos parámetros? En el presente trabajo de investigación se enfoca en esta necesidad, explorando diferentes opciones para poder facilitar esta automatización utilizando inteligencia computacional. Específicamente, se propone la exploración de técnicas como Deep Learning y Reinforcement Learning, las cuales permitirían el entrenamiento previo de los robots, para que estos luego sean capaces de navegar entornos desconocidos con relativa facilidad y sin la necesidad de un operador monitoreando su progreso.

El objetivo es diseñar un algoritmo altamente autónomo, el cual tenga la capacidad de ser acoplado a un robot y a partir de su exploración del espacio de tarea, descubra, en conjunto con otros robots dispuestos a manera de enjambre, la mejor manera de resolver el problema de navegar alrededor de un entorno para llegar a su meta.

Antecedentes

El departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala inició su introducción en el mundo de la *Swarm Intelligence* con la fase 1 del «Megaproyecto Robotat». En este, diversos estudiantes se enfocaron en el diseño de todo el equipo que sería utilizado en años posteriores: Desde el diseño mecánico y electrónico de los «Bitbots»[1], hasta la construcción física de la mesa donde se colocarían los mismos [2, pág. 19].

Aunque este proyecto finalizó con gran parte de su estructura finalizada, muchos aspectos aún requerían de más trabajo. Debido a esto, en 2019 se comenzaron a refinar múltiples aspectos del «Robotat», como el protocolo de comunicación empleado por los *Bitbots* [2] y el algoritmo de visión computacional que se encargaría de detectarlos sobre la mesa en la que se desplazarían [3]. Otra área de gran enfoque dentro de todo este proceso, consistió del algoritmo encargado de controlar el comportamiento de *enjambre* de los robots. En esta área se desarrollaron tres tesis distintas.

Formaciones en Sistemas de Robots Multi-agente

La primera de las mismas, desarrollada por Andrea Peña [4], se enfocó en la utilización de teoría de grafos y control moderno para la creación y modificación de formaciones en conjuntos de múltiples agentes capaces de evadir obstáculos. El algoritmo resultante fue implementado tanto en Matlab como Webots, y aunque los resultados de creación de formaciones no fueron exactamente los deseados[2] el algoritmo de evasión de obstáculos fue mucho más exitoso y marcó un precedente para futuras investigaciones.

Implementación de PSO con Robots Diferenciales Reales

En la segunda tesis, desarrollada por Aldo Aguilar [5], se tomó como base la versión estándar del algoritmo de «Particle Swarm Optimization» (PSO) y se procedió a modificarla para que fuera capaz de ser implementada en robots diferenciales reales. El problema principal con acoplar directamente el movimiento de las partículas del PSO con la locomoción de los robots es que el movimiento de las partículas es sumamente irregular, por lo que puede causar la saturación de los actuadores de los «Bitbots».

Para combatir este problema se propuso una modificación: Cada uno de los robots no seguirían el movimiento exacto de una partícula del PSO, sino que cada uno tomaría la posición de la misma como una *sugerencia* de hacia donde desplazarse. Esta sugerencia luego sería alimentada a un controlador que se encargaría de calcular la velocidad angular de las dos ruedas del robot diferencial. Se experimentó con 8 diferentes metodologías de control para el movimiento de los robots de forma acorde a sus especificaciones y capacidades.

La efectividad de cada método de control se cuantificó haciendo una interpolación de las

[1] Versiones más económicas del robot empleado con propósitos educativos: E-Puck [1]

[2] Las métricas empleadas indicaron que el control fue capaz de producir las formaciones deseadas un 11 % de las veces.

trayectorias seguidas por el robot y luego calculando *la energía de flexión* de las mismas. Mientras más grande fuera la energía de la trayectoria, mayor sería el esfuerzo realizado por el robot en términos de sus velocidades, por lo que se consideraban como más efectivos aquellos métodos que contaran con los valores de energía más bajos. Aplicando este criterio a las diferentes pruebas realizadas en un entorno de simulación, se llegó a determinar que los dos mejores controladores para los robots consistían de los controladores LQR y LQI.

PSO y Artificial Potential Fields

El movimiento de las partículas en la versión *bidimensional* del algoritmo de PSO proviene del movimiento de las mismas sobre una superficie tridimensional denominada «función de costo». El objetivo de las partículas, es encontrar el mínimo global de esta función o las coordenadas (X,Y) correspondientes a la altura más baja del plano [6].

En la tercera tesis, creada por Juan Cahueque [7], se explora la idea de diseñar y utilizar estas funciones como herramientas de navegación. Llamadas *Artificial Potential Fields* (APF), estas funciones están diseñadas para atraer a las partículas del algoritmo PSO hacia un punto específico del plano mientras esquivan cualquier obstáculo presente en el camino. Para esto, se colocaba un valle de gran profundidad en el punto objetivo y colinas de gran magnitud en el sitio donde existen obstáculos. El resultado: Las partículas tendían a esquivar las grandes alturas, favoreciendo el movimiento coordinado hacia los valles o la meta.

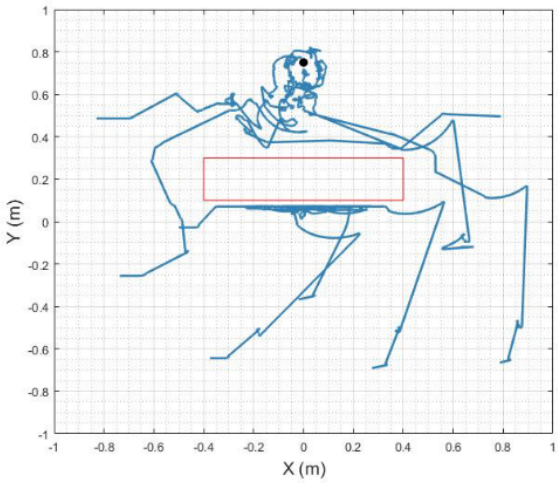

Figura 1: Trayectorias seguidas por E-Pucks en caso A alrededor del obstáculo colocado.

En total se modelaron 3 entornos, cada uno con una meta y múltiples obstáculos intermedios de diferentes dimensiones. A estos escenarios se les denominó caso A, B y C. La navegación alrededor de estos entornos se simuló tanto en Matlab como en Webots. Para las simulaciones en Webots, se emplearon versiones modificadas de los controladores propuestos por Aldo Aguilar [5], a manera de hacerlos compatibles con un entorno en el que se presentan obstáculos. Al finalizar se llegó a concluir que el controlador PID con filtros «hard-stops» era el más útil al momento de evadir obstáculos pequeños y dispersos en el escenario, mientras que el controlador LQR presentaba mayor versatilidad al momento de esquivar obstáculos de mayor tamaño.

Justificación

Uno de los algoritmos más populares dentro de las áreas de Swarm Robotics y Swarm Intelligence, es el algoritmo de Particle Swarm Optimization o PSO. Basado en un algoritmo previamente desarrollado para simular el vuelo de parvadas aves o escuelas de peces [8], este algoritmo consiste de un método de optimización basado en el movimiento conjunto de partículas sobre una superficie o función de costo.

Dado que este algoritmo fue originalmente propuesto en 1995, actualmente existe una gran cantidad de modificaciones y variaciones del mismo. Debido a su eficiencia computacional, no se tiende a variar en gran medida su estructura, colocando mayor énfasis en la modificación de los parámetros asociados al mismo. Según la aplicación, se pueden llegar a favorecer diferentes aspectos como la rapidez de convergencia, exploración de la superficie de costo o la precisión de las partículas en términos de su capacidad para encontrar el mínimo global de la función de costo.

No obstante, en todos estos casos algo permanece constante: No existe un conjunto de parámetros universales que permitan que el algoritmo se ajuste de manera flexible a cualquier aplicación. Existen variaciones *dinámicas* que varían el valor de los parámetros conforme este se ejecuta, pero su efecto es limitado, comúnmente afectando únicamente propiedades como la exploración y convergencia.

Debido a esto, en este proyecto se desea realizar un conjunto de experimentos que lleven al desarrollo de un selector de parámetros dinámico e inteligente, que sea capaz de ajustar los mismos por si solo, sin mayor intervención por parte del usuario. Esto no solo aportará una versión mucho más completa del PSO al resto de la comunidad científica, sino que también permitirá que el algoritmo sea más fácil de utilizar en una variedad de aplicaciones, incluyendo, la actual iniciativa de Swarm Robotics presente en la Universidad del Valle: El Robotat.

Objetivos

Objetivo General

Optimizar la selección de parámetros en algoritmos de inteligencia de enjambre mediante el uso de Reinforcement Learning y Deep Learning.

Objetivos Específicos

- Combinar los trabajos sobre Artificial Potential Fields (APF) y Particle Swarm Optimization (PSO) desarrollados en fases previas del proyecto Robotat.
- Construir una colección de datos de entrenamiento y validación para usar en métodos de Reinforcement y Deep Learning, a partir de múltiples corridas de algoritmos de robótica de enjambre.
- Desarrollar e implementar un algoritmo que determine automáticamente los mejores parámetros para los algoritmos de robótica de enjambre.

Marco teórico

Particle Swarm Optimization (PSO)

Orígenes e Implementación Original

El algorítmo de Particle Swarm Optimization (PSO) consiste de un método de optimización estocástica[3], basado en la emulación de los comportamientos de animales que se movilizan en conjunto. Sus creadores [9] tomaron el algoritmo de simulación de parvada de [8] y experimentaron con el mismo. Luego de múltiples pruebas, se percataron que el algoritmo presentaba las cualidades de un método de optimización. En base a esto, modificaron las reglas del algoritmo de parvada y propusieron la primera iteración del PSO en 1997.

El algoritmo original propone la creación de un conjunto de «m» partículas, cada una con una posición y velocidad correspondientes. Estas partículas se desplazan sobre la superficie de una función objetivo cuyos parámetros (Variables independientes) son las «n» coordenadas de cada partícula. A dicha función objetivo se le denomina «función de costo» y al escalar que genera como resultado se le denomina «costo».

El objetivo de las partículas es encontrar un conjunto de coordenadas que generen el valor de costo más pequeño posible dentro de una región dada. Para esto, las partículas se ubican en posiciones iniciales aleatorias y proceden a calcular el valor de costo correspondiente a su posición actual.

Si el costo actual es inferior al de su posición previa, se dice que la partícula ha encontrado un nuevo «personal best» ($\vec{p}(t)$). Este proceso se repite para cada partícula en el enjambre, por lo que al finalizar cada iteración del algoritmo se contará con «m» estimaciones de $\vec{p}(t)$. El valor mínimo de todas estas estimaciones se le conoce como «mínimo global». Si el mínimo global actual es inferior al de la iteración previa, se dice que se ha encontrado un nuevo «global best» ($\vec{g}(t)$).

Para la actualización de su posición y velocidad actual, las partículas utilizan el siguiente conjunto de ecuaciones:

$$\begin{aligned}
\vec{V}(t+1) = \quad & \vec{V}(t) & \\
& + C_1(\vec{p_{pos}}(t) - \vec{x}(t)) & \textit{Componente Cognitivo} \\
& + C_2(\vec{g_{pos}}(t) - \vec{x}(t)) & \textit{Componente Social} \\
\vec{X}(t+1) = \quad & \vec{X}(t) + \vec{V}(t+1) &
\end{aligned}$$

Debido a que el «personal best» proviene de la memoria individual de cada partícula sobre su mejor posición hasta el momento, a la sección de la ecuación de la velocidad que utiliza $\vec{p_{pos}}(t)$ se le denomina el «componente cognitivo». Por otro lado, debido a que el «global best» proviene de la memoria colectiva sobre la mejor posición alcanzada hasta el momento, a la sección de la ecuación de la velocidad que utiliza $\vec{g_{pos}}(t)$ se le denomina «componente social».

[3]Estocástico: Cuyo funcionamiento depende de factores tanto predecibles dado el estado previo del sistema, así como en factores aleatorios

Mejoras Posteriores

El algoritmo PSO propuesto por [10], presentaba un comportamiento oscilatorio o divergente en ciertas situaciones, por lo que en 2002, [11] se dio a la tarea de diseñar múltiples métodos para restringir y asegurar la convergencia del algoritmo. Uno de los más utilizados hasta la actualidad consiste de una modificación a la regla de actualización de la velocidad conocida como «modelo tipo 1'»:

$$\begin{aligned} \vec{V}(t+1) = \quad & \omega\vec{V}(t) && \textit{Término Inercial} \\ & +R_1C_1(\vec{p_{pos}}(t) - \vec{x}(t)) && \textit{Componente Cognitivo} \\ & +R_2C_2(\vec{g_{pos}}(t) - \vec{x}(t)) && \textit{Componente Social} \end{aligned}$$

En este nuevo conjunto de ecuaciones, las variables de restricción agregadas (C_1, C_2 y ω) están dadas por las siguientes expresiones:

$$\begin{aligned} \omega =& \chi & \chi =& \frac{2\kappa}{|2-\phi-\sqrt{\phi^2-4\phi}|} \\ C_1 =& \chi\phi_1 & \phi =& \phi_1+\phi_2 \\ C_2 =& \chi\phi_2 \end{aligned}$$

Como se puede observar, bajo estas modificaciones, la velocidad es ahora dependiente de tres variables nuevas: ϕ_1, ϕ_2 y κ. Los autores de la modificación sugieren que $\phi_1 = \phi_2 = 2.05$ y $\kappa = 1$, aunque como regla general, para asegurar la convergencia del algoritmo se debe cumplir con que $\kappa > (1 + \phi - 2\sqrt{\phi})|C_2|$.

Deep Learning

En la actualidad, los términos «Machine Learning» y «Deep Learning» se han convertido en sinónimos de inteligencia artificial, pero en muchas ocasiones se desconoce la diferencia entre ambos. De acuerdo con [12], el Machine Learning es un sistema que produce reglas a partir de datos y respuestas, en lugar de producir respuestas a partir de reglas y datos, como es el caso de un programa tradicional.

Más formalmente, Machine Learning se puede definir como la búsqueda de representaciones útiles de un conjunto de datos de entrada dentro de un espacio de posibilidades, utilizando una señal de retroalimentación (Datos de entrenamiento) como guía. El Deep Learning entonces, consiste de un sub-área del Machine Learning donde se obtienen nuevamente representaciones útiles de datos, pero colocando particular énfasis en el aprendizaje por medio de *capas* apiladas de representaciones cada vez más complejas [12].

Los modelos utilizados para crear estas capas apiladas de representaciones se les denomina redes neuronales y similar al cerebro humano, la unidad fundamental de una red es la neurona. Si el número de capas y neuronas del modelo es muy numeroso (Ejemplos modernos comunes utilizan cientos de capas sucesivas para sus modelos) el modelo puede ser considerado Deep Learning. De lo contrario, el modelo se clasifica dentro del área de Shallow Learning.

En el contexto de Deep Learning, una neurona consiste de una regresión lineal modificada que toma los outputs de todas las neuronas de la capa previa ($\boldsymbol{x}$), los multiplica por una matriz de pesos (W^T) y luego les suma un vector de constantes denominados «biases» (B). El output de esta regresión lineal (z) se introduce en una función denominada «función de activación» (σ).

$$\boldsymbol{Z} = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{B}$$
$$\boldsymbol{A} = \sigma(\boldsymbol{Z})$$

Las reglas que rigen a una neurona, como es posible observar, son sumamente simples. La complejidad de un modelo de Deep Learning, proviene de colocar múltiples neuronas en cada capa, y múltiples capas de las mismas dentro de la red.

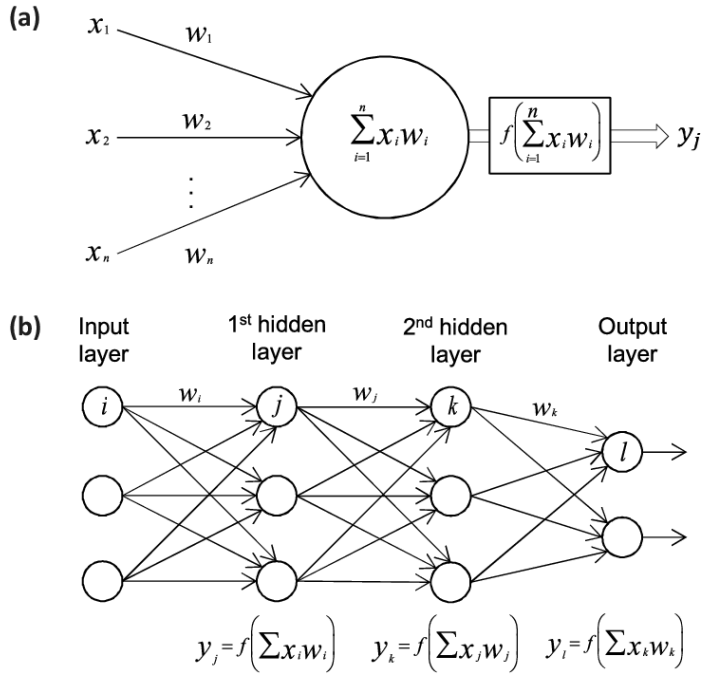

Figura 2: (a) Estructura de neurona. (b) Ejemplo de una red neuronal.

Las neuronas de la red «aprenden» alimentando una serie de datos en las neuronas de entrada o la «input layer» y propagando los datos a través de toda la red hasta finalmente obtener una salida «y». Ahí, la salida es introducida en una función de costo, en conjunto con las salidas deseadas dados los datos de entrada, produciendo un escalar que indica el error inherente a la salida o «costo». El objetivo es minimizar el valor del costo, por lo que este se utiliza como guía para propagar cambios a los vectores $\boldsymbol{W}$ y $\boldsymbol{B}$ de las neuronas individuales. Se continúa este proceso de forma iterativa hasta encontrar el conjunto de vectores $\boldsymbol{W}$ y $\boldsymbol{B}$ que producen el costo más pequeño [12].

Reinforcement Learning o Q-Learning

Al igual que el Deep Learning, el Reinforcement Learning también consiste de una subrama del Machine Learning, ya que este tipo de aprendizaje es capaz de generar representaciones útiles de datos. La diferencia con el Deep Learning radica en la forma en la que

genera los modelos para estas representaciones. En el caso del Reinforcement Learning, al sistema no se le dice como operar, sino que este debe descubrir cuales son las acciones que producen la mejor recompensa probando las diferentes opciones disponibles [13].

Este tipo de aprendizaje aplica ideas de control óptimo sobre procesos incompletamente-conocidos de Markov. Lo que implica esto, es que un agente sujeto a este proceso de aprendizaje debe ser capaz de hacer 3 cosas: Sensar el estado del ambiente, tomar acciones para cambiar dicho estado y contar con un objetivo relacionado con la estructura de dicho ambiente.

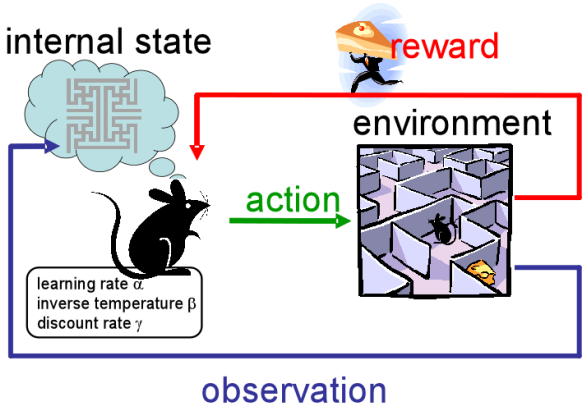

Figura 3: Estructura básica de un problema de Reinforcement Learning

En muchos casos, esto se puede confundir con aprendizaje supervisado[4], pero la diferencia es que los datos utilizados para entrenar un algoritmo de Reinforcement Learning, consisten de ejemplos de comportamientos deseados por parte del sistema dados por un agente externo que supervisa el problema.

El objetivo de este tipo de aprendizaje, es que el agente sea capaz de extrapolar o generalizar comportamientos y respuestas, para que este sea capaz de responder de forma acorde a situaciones que no se presentaron durante su entrenamiento. En particular, una de las características más interesantes de este tipo de aprendizaje, es que el agente, dado un entrenamiento correcto, es capaz de considerar un problema como parte de un sistema de condiciones mucho mayor, adquiriendo la capacidad de incluso considerar cómo decisiones presentes podrían llegar a afectar la futura obtención de recompensas [13].

[4]Aprendizaje en el que se le introducen datos a un sistema y explícitamente se le indica cuales son las salidas deseadas dados los datos de entrada.

Metodología

El primer paso a tomar consistirá en tomar los prototipos previamente trabajados por Aldo Nadalini [5] y Juan Pablo Cahueque [7], y combinarlos para poder generar un script base que sea capaz no solo de tomar en cuenta las dimensiones y capacidades físicas del EPuck, sino que también pueda esquivar obstáculos a través del modelado de funciones de costo personalizadas empleando Artificial Potential Fields.

Esta tarea no solo implicará la combinación de los scripts de Webots, sino que también de la combinación de los scripts de Matlab empleados para poder seleccionar los parámetros óptimos para el MPSO empleado. Por lo tanto, se realizará una Toolbox, que incluirá la capacidad de replicar todas las pruebas y experimentos que los dos predecesores de esta tesis habían previamente realizado, todo debidamente documentado para que cualquier usuario futuro sea capaz de retomar la tarea en el menor tiempo posible.

Una desventaja de este proceso, donde parte del desarrollo se realiza en Webots y otra parte en Matlab, es que muchas de las ideas que pueden llegar a ser fácilmente prototipadas en Matlab, requieren de mucho más esfuerzo para convertirse al lenguaje de programación «C». Considerando que el tema de la tesis consiste en la utilización de redes neuronales y machine learning (Algo no tan sencillo de implementar en un lenguaje como C) se propone la creación de una simulación simplificada de los EPucks dentro de Matlab. De esta manera, el proceso de creación de prototipos se acelerará significativamente.

Con este entorno de simulación listo, se procederá a realizar un gran número de experimentos a partir de los diferentes parámetros y variaciones de PSO presentes en las tesis pasadas, así como en nuevas potenciales investigaciones. Las métricas recolectadas consistirán de factores como el número de *partículas exitosas* (O partículas que cumplen con determinadas condiciones de convergencia), la energía empleada por los robots durante sus trayectorias, el tiempo de convergencia y las capacidades exploratorias de las partículas. A partir de esto, se ensamblará un dataset con los parámetros que generaron las corridas más exitosas.

Una vez los datos de entrenamiento y validación se encuentren listos, se procederá a estructurar una variedad de metodologías de Deep Learning y Reinforcement Learning que puedan utilizar estos datos para obtener un control óptimo, capaz de incrementar la robustez de los métodos de navegación previamente diseñados. La idea es entrenar al algoritmo, de tal manera que la operación del mismo sea casi autónoma, haciendo mucho más accesible su implementación a futuros usuarios.

Finalmente, se colocarán todos los elementos diseñados dentro de un paquete de software, se documentarán propiamente para el uso de futuros usuarios y se implementarán de manera que sean fácilmente manejables por personas nuevas al entorno de machine learning.

En caso llegue a presentarse la oportunidad, incluso se podrían llegar a incluir elementos de la tésis presentada por Andrea Peña: Formaciones en sistemas de robots multi-agente. También se podría intentar implementar los controladores sobre otros algoritmos de optimización distintos al PSO para así probar la efectividad de los mismos como una solución «universal» de control. Aunque estas propuestas resultan interesantes, no consisten de una prioridad, por lo que se denominan como opcionales.

Cronograma de actividades

	Tarea / Implementación	Semanas																							
		Abril				Mayo				Junio				Julio				Agosto				Septiembre			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	25
1	Algoritmo básico PSO																								
2	Funciones de costo "benchmark"																								
3	Colisiones físicas de robots diferenciales con su ambiente																								
4	Parámetros de constricción por Nadalini A.																								
5	Controladores LQR y LQI por Nadalini A.																								
6	Creación de APF's con cualquier geometría																								
7	Controladores LQR y LQI modificados para esquivado de obstáculos por Cahueque J.																								
8	Estadísticas para medir la efectividad de un controlador																								
9	Curso sobre Deep Learning por Andrew NG																								
10	Investigación sobre Reinforcement Learning																								
11	Exploración de alternativas para la aplicación conjunta de Reinforcement Learning, Deep Learning y PSO.																								
12	Migración de scripts desarrollados a POO																								
13	Creación de datos de entrenamiento para red neuronal																								
14	Controlador inteligente (Basado en Deep Learning) para los parámetros de constricción del algoritmo de PSO																								
15	Controlador inteligente (Basado en Deep Learning) para las velocidades angular y lineal de los robots diferenciales																								
16	Pruebas sustituyendo el mecanismo de "Back propagation" tradicional de la red neuronal, por uno de carácter evolutivo																								
17	Creación de un "playground" útil para pruebas asociadas a Q-Learning																								
18	Controlador para la navegación de entornos con obstáculos utilizando Q-Learning																								
19	Calificación de los diferentes métodos de control implementados para elegir aquel con mejor rendimiento																								
20	Validación de los mejores resultados obtenidos en Webots.																								
21	Unión de todos los scripts creados en una Toolbox																								
22	Redacción de documentación de proyecto																								
23	Redacción de reporte final de tesis																								

Clave de Color

- Completado
- Por Hacer

Índice preliminar

13. Controladores Basados en Redes Neuronales

    13.1. Obtención de Datos de Entrenamiento

    13.2. Arquitectura

    13.3. Controlador de Parámetros de Constricción de PSO

    13.4. Controlador de Velocidad Lineal y Angular de los Robots Diferenciales

    13.5. Sustitución de Back Propagation por Algoritmo Evolutivo

    13.6. Resultados de Entrenamiento y Oportunidades de Mejora

14. Controlador de Hiperparámetros Basado en Reinforcement Learning

    14.1. Creación de "Playground" para los E-Pucks

    14.2. Estructura del Sistema de Recompensas

    14.3. Combinación con Controladores Basados en Redes Neuronales

    14.4. Resultados del Proceso de Aprendizaje

15. Validación

    15.1. Ventajas y Desventajas de los Controladores Implementados

    15.2. Selección del Controlador con el Mejor Desempeño

    15.3. Validación en Webots

16. Exploración: Aplicación de los Controladores Inteligentes en conjunto con otros Algoritmos de Navegación.

    16.1. Ant Colony

    16.2. A*

    16.3. D*

17. Conclusiones

18. Recomendaciones

19. Bibliografía

20. Anexos

Referencias

[1] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-c. Zufferey, D. Floreano y A. Martinoli, «The e-puck, a robot designed for education in engineering», en *Proceedings of the 9th conference on autonomous robot systems and competitions*, vol. 1, 2009, págs. 59-65.

[2] J. Castillo, «Diseñar e implementar una red de comunicación inalámbrica para la experimentación en robótica de enjambre», Tesis doct., 2019, pág. 54.

[3] A. Hernández, «Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre», Tesis doct., 2019.

[4] A. Maybell y P. Echeverría, «Algoritmo de sincronización y control de sistemas de robots multi-agente para misiones de búsqueda», Tesis doct., Universidad del Valle de Guatemala, 2019.

[5] A. Nadalini, «Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO)», Tesis doct., Universidad del Valle de Guatemala, 2019.

[6] K. F. Uyanik, «A study on Artificial Potential Fields», vol. 2, n.º 1, págs. 1-6, 2011.

[7] J. Cahueque, «Implementación de Enjambre de Robots en Operaciones de Búsqueda y Rescate», Tesis doct., Universidad del Valle de Guatemala, 2019.

[8] C. W. Reynolds, «Flocks, herds, and schools: A distributed behavioral model», *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*, vol. 21, n.º July, págs. 25-34, 1987. DOI: 10.1145/37401.37406.

[9] R. Eberhart y J. Kennedy, «New optimizer using particle swarm theory», en *Proceedings of the International Symposium on Micro Machine and Human Science*, 1995, págs. 39-43, ISBN: 0780326768. DOI: 10.1109/mhs.1995.494215.

[10] M. Clerc, «The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization», en *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, vol. 3, 1999, págs. 1951-1957, ISBN: 0780355369. DOI: 10.1109/CEC.1999.785513.

[11] M. Clerc y J. Kennedy, «The Particle Swarm — Explosion , Stability , and Convergence in a Multidimensional Complex Space», *IEEE Transactions on Evolutionary Computation*, vol. 6, n.º 1, págs. 58-73, 2002.

[12] F. Chollet, *Deep Learning With Python*. Manning, 2018, pág. 373, ISBN: 9781617294433.

[13] R. S. Sutton y A. G. Barto, *Reinforcement Learning, An Introduction*, Second, F. Bach, ed. Cambridge, Massachusetts: MIT Press, 2018, pág. 400, ISBN: 9780262039246.