UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Reinforcement Learning / Deep Learning en aplicaciones de robótica de enjambre

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

Guatemala,

2019

Resumen

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

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* en 2017 con la fase 1 del «Megaproyecto Robotat». En este, diversos estudiantes se enfocaron en el diseño de todo el equipo que sería utilizado por estudiantes en años posteriores: Desde el diseño mecánico y eléctrico 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.

[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.

Implementación de PSO con Robots Diferenciales Reales

En la segunda tesis, desarrollada por Aldo Aguilar [5], se tomó como base el algoritmo de «Particle Swarm Optimization» (PSO) (Ver Marco teórico) y se procedió a modificar el mismo para que este fuera capaz de ser implementado en robots reales con restricciones cinemáticas específicas. Inicialmente, el trabajo se enfocó en implementar el algoritmo PSO, para luego modificar sus parámetros y así obtener el mejor movimiento posible para los robots. En esta versión del algoritmo, cada partícula cuenta con una velocidad específica, la cual se actualiza durante cada iteración siguiendo la siguiente ecuación

$$\vec{V}(t+1) = \chi(\omega\vec{V}(t) + R_1\phi_1(\vec{p_{pos}}(t) - \vec{x}(t)) + R_2\phi_2(\vec{g_{pos}}(t) - \vec{x}(t)))$$

Como se puede observar, en esta ecuación existen 5 parámetros ajenos a los vectores utilizados para posiciones y velocidades: χ, ω, R_1, R_2, ϕ_1 y ϕ_2. R_1 y R_2 consisten de valores aleatorios normalmente distribuidos entre 0 y 1, pero el resto son variables que pueden llegar a ser asignadas por el usuario. Dependiendo de su valor, el algoritmo puede divergir, extender su tiempo de ejecución, limitar el rango de exploración de las partículas o no alcanzar el objetivo deseado. Debido a esto, la primera tarea en [5], consistió en encontrar los valores óptimos para estas variables a manera de asegurar la convergencia del algoritmo sin comprometer la capacidad exploratoria de las partículas.

Previo a iniciar con el proceso de experimentación, se restringieron los valores de ϕ_1, ϕ_2 y χ de acuerdo a las ideas propuestas por Clerc y Kennedy [6], las cuales establecen que se puede asegurar la convergencia del algoritmo mientras $\phi_1 + \phi_2 > 4$ en el siguiente conjunto de ecuaciones

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

$$C_1 = \chi\phi_1 \quad C_2 = \chi\phi_2 \qquad \phi = \phi_1 + \phi_2$$

Una vez se tenía esta restricción, se procedió a experimentar con múltiples combinaciones para ϕ_1, ϕ_2 y ω, cuantificando las capacidades exploratorias de las partículas midiendo la desviación estándar de sus coordenadas X y Y. En el caso particular de ω se experimentó con expresiones dinámicas que cambian según el desarrollo del algoritmo. Se probaron 5 variaciones del coeficiente de inercia: Caótica, random, constante, lineal decreciente y natural exponencial. De estas, aquella que presentó la mejor característica de exploración (A pesar de sufrir un poco en su tiempo de convergencia) fue la inercia natural exponencial. En el caso de ϕ_1 y ϕ_2, se llegó a concluir que los valores que producían los mejores resultados eran 2 y 10 respectivamente. Con esto fuera del camino, se comenzó a considerar la compatibilidad del robot diferencial a emplear (El EPuck) con el algoritmo de PSO.

Considerando que el primer artículo que propone este algoritmo [7], describe a las partículas en el mismo como: «elementos con una arbitrariamente pequeña masa y volumen», se hace rápidamente evidente que los EPucks no podrían llegar a cumplir el papel de partícula en el algoritmo. Entonces se propuso una modificación: Cada uno de los robots no seguirían el movimiento exacto de una partícula, sino que cada uno tomaría la posición de una partícula en la simulación como una *sugerencia o guía* de hacia donde debe desplazarse.

Para evitar confusiones en la terminología, a los robots se les pasó a llamar partículas y a las partículas de la simulación se les llamó «marcadores PSO». Debido a que era necesario probar esta etapa del proyecto en robots físicos, se optó por continuar el proyecto en Webots, donde a cada robot se le programó utilizando C.

Para obtener un control estable y acorde a las especificaciones del EPuck, se inició obteniendo las ecuaciones cinemáticas del robot. Las ecuaciones resultantes eran altamente no lineales, por lo que cualquier técnica de control tradicional no era aplicable. En lugar de optar por linealizar las ecuaciones, se utilizó una técnica que cumple con el mismo objetivo, conocida como un *difeomorfismo*. Empleado para transferir de un espacio vectorial a otro, este difeomorfismo permitió hacer compatible el modelo matemático del robot con una variedad de métodos de control.

Se implementaron 8 métodos de control distintos: Control proporcional de velocidades con saturación limitada, control PID de velocidad angular, control proporcional de velocidad lineal, control simple de pose de robot, controlador de pose con criterio de estabilidad de Lyapunov, control de lazo cerrado de direccionamiento de robot, control LQR y control LQI. Todos se probaron dentro de Webots, un software de simulación de robótica, y a medida que se ejecutó la simulación, se extrajeron datos como la posición, velocidad y aceleración lineal, velocidad y aceleración angular de las ruedas, entre otros.

Los datos fueron luego procesados en Matlab, donde no solo se evaluó la factibilidad de la implementación del controlador, sino que también aspectos como la precisión e incluso la consistencia con la que se generaban los resultados observados. En particular, se colocó especial atención en la energía que empleaba cada robot para llegar a la posición deseada. Esto fue cuantificado haciendo una interpolación de las trayectorias seguidas por el robot y luego calculando *la energía de flexión* de las mismas, como si estas consistieran de vigas. A mayor fuera la energía de la interpolación, mayor sería el esfuerzo realizado por el robot en términos de sus velocidades angulares.

Aplicando esto a las diferentes trayectorias obtenidas en Webots, se llegó a determinar que el mejor controlador en términos de la energía empleada, era el controlador LQR.

PSO y Artificial Potential Fields

El movimiento de las partículas en el algoritmo de PSO proviene del movimiento de las mismas sobre una superficie tridimensional virtual 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. Debido a que el PSO consiste de un algoritmo de optimización, comúnmente las funciones empleadas para probar su efectividad son funciones *benchmark* o funciones estándar de referencia diseñadas para probar cualquier algoritmo de optimización. La idea de los *Artificial Potential Fields* (APF) surge de la siguiente pregunta: ¿Por qué no sustituir estas *Benchmark functions* por una función de costo personalizada diseñada para alejar a las partículas de potenciales obstáculos, mientras alcanzan un objetivo pre-establecido? [8].

De aquí surge la tercera tesis relacionada con *Swarm Intelligence*. Escrita por Juan Cahueque [9], esta se enfoca en el diseño y generación de *Artificial Potential Fields* (APF)

capaces de atraer a las partículas hacia un punto específico del plano, mientras estas esquivan cualquier obstáculo presente en el camino. El contenido de la tesis inicia de manera muy similar a la de Nadalini [5]: Eligen los mejores parámetros para el PSO. La diferencia clave entre ambos, es que Nadalini emplea una versión levemente alterada del PSO estándar, mientras que Cahueque, utiliza un MPSO (Modified PSO) propuesto por Chowdhury [10].

En esta versión modificada del PSO, gran parte de la estructura general del código permanece intacta, el nuevo elemento presente es la regla de actualización de velocidad, cuya ecuación de actualización tiene la siguiente forma

$$V_i^{t+1} = K\left[\alpha V_i^t \beta_l r_1 \left(P_i - X_i^t\right) + \beta_g r_2 \left(P_g - X_i^t\right) + \gamma r_3 \hat{V}_i^t\right]$$

En esta ecuación existen 4 nuevos parámetros de restricción ω, α, β y γ. Para elegir los valores finales, se realizó un barrido para cada parámetro y se tomó nota del número de agentes que cumplían con las condiciones de convergencia o el número de agentes «exitosos». Al finalizar, se tomó la combinación de parámetros que, según las pruebas, generaron el mayor número de agentes exitosos. Esto implicó que $\alpha = 400$, $\beta = 50$, $\gamma = 35$ y $\omega = (1.2, 1.8)$.

Con estos parámetros, se procedió a modelar tres espacios distintos en Matlab, los cuales presentaban una meta como objetivo y múltiples obstáculos intermedios. A estos escenarios se les denominó caso A, B y C. Dentro de Matlab, las simulaciones se corrieron sin llegar a considerar parámetros como el tamaño físico de las partículas. No obstante, luego de finalizar las pruebas en simulación, se procedió a probar el algoritmo, en conjunto con un controlador capaz de tomar en cuenta las restricciones físicas del robot, en un ambiente que simula al físico.

Justificación

Uno de los algoritmos más prominentes 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, el algoritmo de PSO consiste de un método de optimización basado en el movimiento 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 la estructura del algoritmo, colocando mayor énfasis en la modificación de los parámetros asociados al mismo. Según la aplicación, se pueden llegar a favorecer diferentes aspectos como la convergencia, exploración de la superficie de costo, rapidez o precisión.

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 mismo sea más fácil de utilizar en una variedad de aplicaciones, incluyendo, la actual iniciativa de Swarm Robotics presente en la Universidad del Valle.

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

- Construir un prototipo que incorpore elementos de la tesis de Aldo Nadalini y Juan Pablo Cahueque. Esto consistiría de una simulación con Artificial Potential Fields y robots capaces de realizar acciones acordes a sus dimensiones y capacidades físicas.
- Explorar múltiples variaciones de PSO para estudiar cuales son las ventajas y desventajas de cada método y de esta manera construir un dataset de entrenamiento/validación para la red neuronal.
- Implementar una modificación de PSO capaz de inteligentemente variar los parámetros del algoritmo utilizando técnicas de Deep Learning, Machine Learning y Reinforcement Learning.
- Aplicar el algoritmo de PSO inteligente modificado, en un enjambre de robots físicos dispuestos sobre una superficie de costo creada utilizando Artificial Potencial Fields
- Reforzar potenciales debilidades en el algoritmo que emplea Deep Learning complementándolo con técnicas de Reinforcement Learning (Q-Learning).

Marco teórico

Particle Swarm Optimization (PSO)

Orígenes e Historia

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. Los inicios del algoritmo se remontan a 1987, cuando Reynolds **Reynolds**

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

desarrolla un método para animar una parvada de «boids», objetos similares a aves cuyo comportamiento individual se ve regido por 3 simples reglas: Cada «boid» solo es capaz de adquirir información de sus vecinos próximos, estos evitaran colisiones con otros «boids» y cada uno adquirirá la velocidad promedio de sus vecinos sin alejarse de la parvada.

Se dice que Kennedy y Eberhart [7] tomaron el algoritmo de Reynolds y comenzaron a experimentar con el mismo. Luego de múltiples pruebas y observaciones, estos se percataron que el algoritmo no solo simulaba un comportamiento natural, sino que presentaba las cualidades de un método de optimización. En base a esto, modificaron las reglas del algoritmo original (Por ejemplo, suprimiendo la regla de la proximidad hacia otras entidades) y propusieron dos variaciones el algoritmo de PSO: La implementación GBEST y LBEST.

La implementación GBEST asume la existencia de un ente global que es capaz de monitorear, recordar y transmitir la posición y velocidad de todas las partículas a todas las demás. La implementación LBEST se asemeja mucho más al enfoque local tomado por Reynolds, donde cada entidad solo es capaz de conocer información procedente de sus vecinos próximos.

Cabe mencionar que se le denominó «Particle Swarm Optimization» en lugar de «Particle Flock Optimization» ya que dentro de las simulaciones realizadas por Kennedy y Eberhart, se pudo observar que el comportamiento conjunto de las partículas se asemejaba más al movimiento de un enjambre que al de una parvada o banco [11].

Funcionamiento

El algoritmo propone la creación de un conjunto de «m» partículas, cada una con un vector de posición y velocidad correspondiente. 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».

$$\vec{x_i} = \begin{bmatrix} x_{i1} & x_{i2} & x_{i3} & \dots & x_{in} \end{bmatrix}$$
$$Costo = f(\vec{x}) = f(x_{i1}, x_{i2}, \dots, x_{in})$$

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 se permite que las mismas «exploren» de manera iterativa hasta encontrar el mínimo global de la región.

Para conseguir esto, en cada iteración cada partícula calcula el valor del costo correspondiente a su posición actual y compara con el costo obtenido en la posición previa. Si el costo actual es inferior, se dice que la partícula ha encontrado un nuevo «personal best» ($\vec{p}(t)$), por lo que se almacena en memoria tanto el valor de costo como la posición que lo ha generado.

$$\vec{p_{pos}}(t) = \begin{bmatrix} x_{min1} & x_{min2} & x_{min3} & \dots & x_{min"n"} \end{bmatrix}$$
$$\vec{p_{cost}}(t) = f(\vec{p_{pos}}(t))$$

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 el «mínimo global» de dicha iteración. Si este mínimo es inferior al de la iteración previa, se dice que se ha encontrado un nuevo «global best» ($\vec{g}(t)$). Nuevamente, se almacena tanto el valor del costo, como la posición que ha generado dicho valor.

Para la actualización de la posición, las partículas suman a su posición actual una nueva velocidad calculada utilizando su velocidad actual ($\vec{V}(t)$), las posiciones asociadas al personal y global best ($\vec{p_{pos}}(t)$, $\vec{g_{pos}}(t)$), y dos escalares aleatorios uniformemente distribuidos (R_1, R_2). 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».

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

$$\vec{X}(t+1) = \vec{X}(t) + \vec{V}(t+1)$$

Cabe mencionar que la implementación detallada previamente consiste de la implementación GBEST. La implementación LBEST no se describe en mayor detalle por los autores debido a que es muy similar a la anterior. La única diferencia con respecto al GBEST, es que en esta no existe un único global best. Debido a que cada partícula es capaz de comunicarse con sus vecinos próximos, pueden llegar a existir múltiples versiones del global best.

Mejoras Posteriores

Debido a que la primera implementación del PSO consistía de la modificación de un algoritmo pre-existente basado en evidencia puramente experimental, este carecía de fundamentos teóricos que justificaran su funcionamiento y fallas. Con el objetivo de solucionar esto, Clerc y Kennedy [12] se dieron a la tarea de tomar un enfoque más riguroso para el análisis del algoritmo.

A partir de su análisis se llegó a determinar que la razón por la que el algoritmo presentaba un comportamiento oscilatorio o divergente en ciertas situaciones era porque no existía control sobre la influencia que tenía cada uno de los componentes de $\vec{V}(t+1)$ sobre el valor final de este vector. Tomando en cuenta esto, se diseñaron múltiples métodos para restringir y asegurar la convergencia del algoritmo. Uno de los métodos más comúnmente implementados 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, las nuevas variables de restricción agregadas 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. Clerc y Kennedy 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 $\kappa > (1 + \phi - 2\sqrt{\phi})|C_2|$. Puede que otras combinaciones generen convergencia, pero esto se torna en un resultado altamente dependiente de la disposición aleatoria inicial de las partículas.

Redes Neuronales

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Metodología

El primer paso a tomar consistirá en tomar los prototipos previamente trabajados por Aldo Nadalini [5] y Juan Pablo Cahueque [9], 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 especie suite, 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. Además de esto, se tomarán los scripts creados en el software de Webots y se unificarán para nuevamente realizar pruebas ahí.

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 en 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 machine y deep learning que puedan utilizar estos datos para poder a un control óptimo, capaz de generar resultados consistentes, precisos y rápidos, todo sin la necesidad de que un humano se introduzca en mayor medida a modificar los parámetros deseados. La idea es entrenar a la estructura diseñada, de tal manera que la operación del algoritmo sea casi autónoma, haciendo mucho más accesible su implementación a futuros usuarios.

Finalmente, se colocarán todos los elementos diseñados, 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. Aunque esto resulta interesante, no es una prioridad, por lo que se denomina como opcional.

Cronograma de actividades

Semana 1 **Comprensión de código y tesis de Aldo Nadalini:** Comprensión de todos los elementos que empleó en sus scripts tanto de Matlab como de Webots. Ser capaz de

modificar diferentes parámetros a manera de generar nuevos resultados.

Semana 2 **Comprensión de código y tesis de Juan Pablo Cahueque:** Comprensión de todos los elementos que empleó en sus scripts tanto de Matlab como de Webots. Ser capaz de modificar diferentes parámetros a manera de generar nuevos resultados.

Semana 3 **Implementación del «Diferential Robot Toolbox»:** Implementar el Toolbox en cuestión, en conjunto con los diferentes parámetros y tipos de PSO provenientes de las tesis previas con el objetivo de acelerar el proceso de prototipado en Matlab. El Toolbox debería de ser compatible al menos con la versión base del algoritmo base del PSO. A manera de hacer dicha simulación más realista, también se buscará implementar algún motor de física rudimentario con el objetivo de evaluar eventos como choques.

Semana 4 **Unificación de Toolbox y elementos de Tesis Previas (1):** Con el algoritmo base de PSO corriendo nativamente en Matlab, se procederá a incluir todos los elementos presentes en las tesis de Juan Pablo y Aldo. Esto implica la programación del APF y los controladores para tomar en cuenta las dimensiones y capacidades físicas del robot.

Semana 5 **Experimentación con constantes de PSO:** Se tomará la herramienta creada y los criterios de desempeño de tesis anteriores, y se comenzarán a realizar diversas variaciones a los parámetros para determinar cuales son los que mejor desempeño generan. Se deberá generar una cantidad de datos sustancial a manera de utilizar estos como el dataset para los algoritmos posteriores.

Semana 6 **Experimentación con constantes de controladores:** Se tomarán los diferentes controladores empleados para las diferentes tesis previas y se procederá a variar sus diferentes parámetros para obtener un dataset correspondiente a los parámetros de control que generan los mejores resultados.

Índice preliminar

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

[6] 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.

[7] 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.

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

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

[10] S. Chowdhury, W. Tong, A. Messac y J. Zhang, «A mixed-discrete Particle Swarm Optimization algorithm with explicit diversity-preservation», *Structural and Multidisciplinary Optimization*, vol. 47, n.º 3, págs. 367-388, 2013, ISSN: 1615147X. DOI: 10.1007/s00158-012-0851-z.

[11] E. García-Gonzalo y J. L. Fernández-Martínez, «A Brief Historical Review of Particle Swarm Optimization (PSO)», *Journal of Bioinformatics and Intelligent Control*, vol. 1, n.º 1, págs. 3-16, 2013, ISSN: 23267496. DOI: 10.1166/jbic.2012.1002.

[12] 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.