

UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Reinforcement y Deep Learning en Aplicaciones de Robótica  
de Enjambre**

Trabajo de graduación presentado por Eduardo Andrés Santizo Olivet  
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2020







UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Reinforcement y Deep Learning en Aplicaciones de Robótica  
de Enjambre**

Trabajo de graduación presentado por Eduardo Andrés Santizo Olivet  
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2020



Vo.Bo.:

(f) \_\_\_\_\_  
Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) \_\_\_\_\_  
Dr. Luis Alberto Rivera Estrada

(f) \_\_\_\_\_  
MSc. Carlos Esquit

(f) \_\_\_\_\_  
MSc. Miguel Zea

Fecha de aprobación: Guatemala, 5 de diciembre de 2020.





Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



<b>Prefacio</b>	<b>v</b>
<b>Lista de figuras</b>	<b>IX</b>
<b>Lista de cuadros</b>	<b>XI</b>
<b>Resumen</b>	<b>XIII</b>
<b>Abstract</b>	<b>XV</b>
<b>1. Introducción</b>	<b>1</b>
<b>2. Antecedentes</b>	<b>3</b>
2.1. Formaciones en Sistemas de Robots Multi-agente . . . . .	3
2.2. Implementación de PSO con Robots Diferenciales Reales . . . . .	4
2.3. PSO y Artificial Potential Fields . . . . .	4
<b>3. Justificación</b>	<b>7</b>
<b>4. Objetivos</b>	<b>9</b>
4.1. Objetivo General . . . . .	9
4.2. Objetivos Específicos . . . . .	9
<b>5. Alcance</b>	<b>11</b>
<b>6. Marco teórico</b>	<b>13</b>
6.1. Particle Swarm Optimization (PSO) . . . . .	13
6.1.1. Orígenes e Implementación Original . . . . .	13
6.1.2. Mejoras Posteriores . . . . .	14
6.2. Deep Learning . . . . .	14
6.3. Reinforcement Learning . . . . .	16
6.3.1. Multi-armed Bandits . . . . .	16
6.3.2. Procesos de Decisión de Markov (MDP's) . . . . .	24

<b>7. Entorno de Simulación (Toolbox) para Entrenamiento y Realización de Pruebas de Algoritmos de Enjambre</b>	<b>31</b>
7.1. Parámetros Generales . . . . .	31
7.2. Obstáculos . . . . .	33
7.3. Tipo de Restricción . . . . .	34
7.4. Colisiones . . . . .	35
7.5. Controladores . . . . .	35
7.6. Criterios de Convergencia . . . . .	36
7.7. Análisis de Resultados . . . . .	37
7.7.1. Evolución del Global Best . . . . .	37
7.7.2. Análisis de Dispersión de Partículas . . . . .	37
7.7.3. Velocidad de Motores . . . . .	38
7.7.4. Suavidad de Velocidades . . . . .	39
7.8. Grabación de Videos / Frames . . . . .	39
<b>8. Obtención de Datos para Entrenamiento de Red Neuronal</b>	<b>41</b>
<b>9. Controladores Basados en Redes Neuronales</b>	<b>43</b>
<b>10. Controlador de Hiperparámetros Basado en Reinforcement Learning</b>	<b>45</b>
<b>11. Validación de Resultados</b>	<b>47</b>
<b>12. Exploración: Aplicación de los Controladores Inteligentes en conjunto con otros Algoritmos de Navegación</b>	<b>49</b>
<b>13. Conclusiones</b>	<b>51</b>
<b>14. Recomendaciones</b>	<b>53</b>
<b>15. Bibliografía</b>	<b>55</b>
<b>16. Anexos</b>	<b>57</b>
16.1. Cosos cosos cosos . . . . .	57

---

## Lista de figuras

---

1.	Trayectorias seguidas por E-Pucks en caso A alrededor del obstáculo colocado.	5
2.	(a) Estructura de neurona. (b) Ejemplo de una red neuronal.	15
3.	Representación gráfica de un «Multi-armed bandit»	16
4.	Tres estimados diferentes para el valor de una acción. La línea punteada representa el valor a tomar dado que se trata del límite superior de la incertidumbre	23
5.	Interacción Agente-Ambiente en un proceso de decisión de Markov [13]	24
6.	Dinámica de un MDP	25
7.	Sumar las recompensas de los primeros 3 estados es lo mismo que sumar la serie infinita, ya que luego se pasa al estado de absorción (Cuadro gris).	30
8.	Efectos de alterar el parámetro <i>EnablePucks</i> .	32
9.	Efectos de alterar el parámetro <i>ModoVisualizacion</i> .	32
10.	Creación de obstáculo poligonal.	33
11.	Caso A en tesis de Juan Pablo Cahueque	33
12.	Caso C en tesis de Juan Pablo Cahueque	34
13.	Con solución de colisiones (Izquierda) y sin solución de colisiones (Derecha)	35
14.	Marcadores PSO	36
15.	Evolución de la minimización hacia el global best de la función	37
16.	Dispersión de las partículas sobre el eje X y Y.	38
17.	Velocidad angular observada en los motores del puck con los picos más altos de velocidad en dicha corrida	38
18.	Energía de flexión observada en las velocidades angulares de las ruedas de cada puck.	39



---

## Lista de cuadros

---





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.



The Particle Swarm Optimization (PSO) algorithm is a stochastic optimization algorithm, based off a modified boid simulation algorithm. When the creators of the PSO algorithm took the boid simulation and removed all the constraints related to proximity between boids, they realized that the simulated entities now exhibited an optimizing behavior.

After the publication of this paper, a large section of the academic community took notice of the big potential that this algorithm held, and started to modify it in order to use it for new applications. This is the case for the project *Robotat* developed by the Universidad del Valle de Guatemala, where the algorithm was proposed as the base for a navigation system deployed on group of differential drive robots known as *Bitbots*.

Previous members of this project were able implement a modified PSO algorithm that not only was able to generate actuator speeds according to the physical limitations of the robot, but also managed to avoid certain obstacles. However, some of the results were highly dependant on multiple hyperparameters that were selected based off previous knowledge of the environment in which the robots were going to be deployed.

¿Is there a way to automate the selection process of said hyperparameters? In the following thesis, multiple options are explored, in order to develop an automation algorithm based off machine learning. Specifically, techniques based off deep learning and reinforcement learning are explored. The idea is to enable smart robot navigation in unknown environments in the absence of a human intervening in any way.

The objective is to design an autonomous controller able to be coupled with a robot and without any supervision, discover and develop a policy (In conjunction with other similar robots) that best solves the current objective task or function.



# CAPÍTULO 1

---

## Introducción

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



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»<sup>1</sup>, 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.

### 2.1. 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<sup>2</sup> el algoritmo de evasión de obstáculos fue mucho más exitoso y marcó un precedente para futuras investigaciones.

---

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

<sup>2</sup>Las métricas empleadas indicaron que el control fue capaz de producir las formaciones deseadas un 11 % de las veces.

## 2.2. 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 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.

## 2.3. 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.

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.

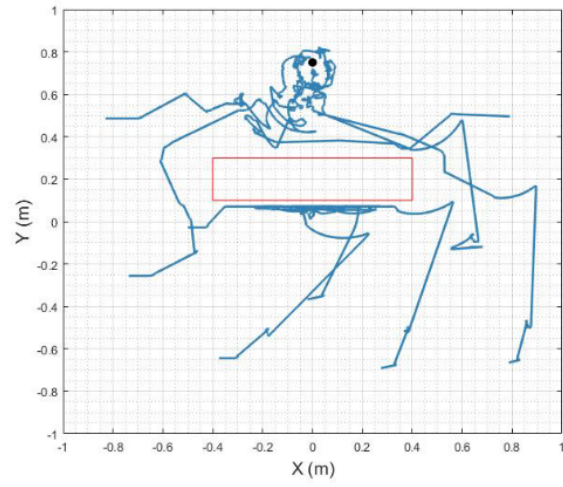


Figura 1: Trayectorias seguidas por E-Pucks en caso A alrededor del obstáculo colocado.



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.



#### 4.1. Objetivo General

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

#### 4.2. 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.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.





## 6.1. Particle Swarm Optimization (PSO)

### 6.1.1. Orígenes e Implementación Original

El algoritmo de Particle Swarm Optimization (PSO) consiste de un método de optimización estocástica<sup>1</sup>, 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» ( $\vec{g}(t)$ ).

---

<sup>1</sup>Estocástico: Cuyo funcionamiento depende de factores tanto predecibles dado el estado previo del sistema, así como en factores aleatorios

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

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

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  $p_{pos}^{\vec{}}(t)$  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  $g_{pos}^{\vec{}}(t)$  se le denomina «componente social».

### 6.1.2. 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»:

$$\begin{aligned}
\vec{V}(t+1) = & \omega \vec{V}(t) && \text{Término Inercial} \\
& + R_1 C_1 (p_{pos}^{\vec{}}(t) - \vec{x}(t)) && \text{Componente Cognitivo} \\
& + R_2 C_2 (g_{pos}^{\vec{}}(t) - \vec{x}(t)) && \text{Componente Social}
\end{aligned}$$

En este nuevo conjunto de ecuaciones, las variables de restricción agregadas ( $C_1$ ,  $C_2$  y  $\omega$ ) 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:  $\phi_1$ ,  $\phi_2$  y  $\kappa$ . 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  $\kappa > (1 + \phi - 2\sqrt{\phi})|C_2|$ .

## 6.2. 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 ( $\mathbf{x}$ ), los multiplica por una matriz de pesos ( $\mathbf{W}^T$ ) y luego les suma un vector de constantes denominados «biases» ( $\mathbf{B}$ ). El output de esta regresión lineal ( $\mathbf{z}$ ) se introduce en una función denominada «función de activación» ( $\sigma$ ).

$$\mathbf{Z} = \mathbf{W}^T \mathbf{x} + \mathbf{B}$$

$$\mathbf{A} = \sigma(\mathbf{Z})$$

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.

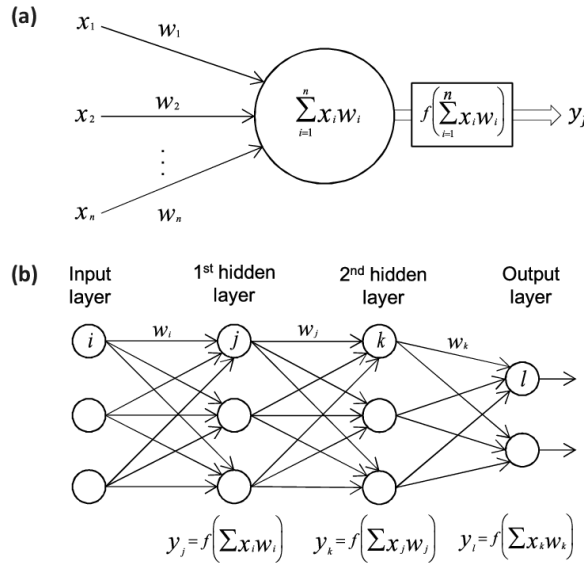


Figura 2: (a) Estructura de neurona. (b) Ejemplo de una red neuronal.

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  $\mathbf{W}$  y  $\mathbf{B}$  de las neuronas individuales. Se continúa este proceso de forma iterativa hasta encontrar el conjunto de vectores  $\mathbf{W}$  y  $\mathbf{B}$  que producen el costo más pequeño [12].

## 6.3. Reinforcement 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].

### 6.3.1. Multi-armed Bandits

#### K-Armed Bandit

Nos podemos imaginar que tenemos una «bandit machine» (Máquinas encontradas en los casinos, de las que uno jala la palanquita) con  $k$  brazos. Cada vez que jalamos uno de los brazos, obtenemos una recompensa (Digamos fichas). El objetivo, es maximizar la recompensa obtenida luego de cierta cantidad de juegos o rondas, por ejemplo, luego de jalar 1000 veces los brazos. La recompensa entregada por cada brazo sigue una distribución probabilística distinta para la entrega de su recompensa. A este problema se le denomina el «K-armed bandit problem»



Figura 3: Representación gráfica de un «Multi-armed bandit»

Cada una de las « $k$ » acciones a tomar tienen una recompensa promedio dada por la

acción tomada. A esta cantidad se le llamará el «valor» de la acción y está dada por la siguiente expresión

$$\begin{aligned} q_*(a) &\doteq \mathbb{E}[R_t \mid A_t = a] \quad \forall a \in \{1, \dots, k\} \\ &= \sum_r p(r \mid a) r \end{aligned} \tag{1}$$

Donde:

$q_*(a)$  : Valor de la acción  
 $a$  : Acción arbitraria  
 $A_t$  : Acción en el tiempo  $t$   
 $R_t$  : Recompensa en tiempo  $t$

En otras palabras, esto se puede re-expresar como la suma de todas las posibles recompensas. Se suman todas las recompensas por la probabilidad de ver dicha recompensa <sup>2</sup>. Si se supiera el valor de cada acción, el problema sería trivial porque siempre tomaríamos la acción «a» con el mayor valor. Comúnmente no conocemos los valores de las acciones con total seguridad, pero podemos estimarlos. A este estimado le llamamos:  $Q_t(a)$ .

Idealmente el valor de  $Q_t(a)$  debería de ser lo más cercano posible a  $q_*(a)$ . Durante cada «time step» ( $t$ ), existe al menos una acción cuyo valor es el más alto. Estas acciones se denominan «greedy actions». Si se toman estas acciones, se dice que se está explotando el conocimiento actual del valor de las acciones. Si se toman las acciones restantes, se dice que se está explorando, porque esto ayuda a mejorar el estimado de las «non-greedy actions».

«Exploitation» maximiza la recompensa obtenida en un único «time step», pero «Explorar» puede llegar a producir mayor recompensa a largo plazo. En explotación, podríamos decir que se toman las acciones inmediatas con el mejor valor, la exploración permite descubrir otras opciones, permitiendo que a largo plazo se tome una decisión mucho mejor informada en base a todas las opciones disponibles.

No es posible explorar y explotar de manera simultánea tomando una única acción. Debido a esto, comúnmente se habla de un «conflicto» entre exploración y explotación. Existen métodos complejos para balancear ambas etapas, pero están basadas en formulaciones extensas y conocimiento a priori sobre el problema. En «Reinforcement Learning, An Introduction» [13], se proponen algunos métodos simples.

## Métodos de Valor-Acción (Método de promedio de muestreo)

A los métodos para estimar el valor de una acción (Calcular  $Q_t(a)$ ) y luego utilizar este estimado para decidir entre un grupo de acciones se les denomina *Action-value methods*. El valor real de una acción consiste del promedio de la recompensa obtenida cuando esa acción es seleccionada. Una forma de estimar esto es obtener el promedio de las recompensas obtenidas hasta el time step  $t - 1$ .

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{2}$$

---

<sup>2</sup>Si se trata un caso continuo, la suma se puede sustituir por una integral.

Donde:

$\mathbf{1}_{\text{predicate}}$  es una variable aleatoria igual a 1 si el predicado es *verdadero* y 0 si es *falso*.

Si el denominador es 0 (Nunca se ha tomado la acción), entonces asignamos un valor por defecto a  $Q_t(a)$ , como 0. A medida que el denominador tiende a infinito (Se ha tomado muchas veces una acción)  $Q_t(a)$  converge a  $q_*(a)$ . A este método para estimar valores se le denomina «sample-average» o promedio muestral, ya que toma el promedio de un conjunto de muestras de las recompensas disponibles.

#### Métodos para Elegir Acciones

**Greedy Action Selection:** Se selecciona la acción con el valor estimado más alto (Con el  $Q_t(a)$  más alto).

$$A_t \doteq \arg \max_a Q_t(a) \quad (3)$$

**Epsilon Greedy:** Alternativa simple para actuar avariciosamente buena parte del tiempo, pero cada cierto tiempo (Con una probabilidad  $\epsilon$ ) se selecciona aleatoriamente una acción de todas las disponibles, independientemente del valor de sus estimados <sup>3</sup>. La ventaja de estos métodos es que a medida que el número de «time steps» incrementa, cada acción será muestreada un número infinito de veces, asegurando que  $Q_t(a)$  converja a  $q_*(a)$

$$A_t \leftarrow \begin{cases} \arg \max_a Q_t(a) & \text{with probability } 1 - \epsilon \\ a \sim \text{Uniform}(\{a_1 \dots a_k\}) & \text{with probability } \epsilon \end{cases} \quad (4)$$

Esto implica que la probabilidad de seleccionar la acción óptima converge a un valor mayor a  $1 - \epsilon$ , o se traduce a una selección casi segura. A pesar de esto, todas estas consisten de garantías cuando la ejecución del programa tiende a infinito, no es posible concluir sobre la efectividad práctica de los métodos.

El tipo de método a utilizar cambia según la aplicación. Para recompensas que tengan una alta varianza, por ejemplo, el método «Epsilon Greedy» obtendrá mejores resultados que la «greedy action selection». Si la varianza es 0, sería ineficiente implementar un «Epsilon greedy», ya que tomando acciones avariciosas se estimaría el valor de la decisión en el primer intento. A pesar de esto, encontrar un escenario sin varianza es muy extraño. Muy comúnmente los escenarios son no-estáticos, o escenarios donde la toma de diferentes acciones cambia dependiendo del tiempo. Esto causa que el valor real de una acción cambie de manera constante, por lo que un «Epsilon greedy» que también incluya exploración, es necesario para poder tomar estos cambios en cuenta.

---

<sup>3</sup>Se elige la acción «greedy» con frecuencia  $1 - \epsilon$ , y se elige una de las demás acciones de manera aleatoria y con misma probabilidad usando frecuencia  $\epsilon$

## Estimación Incremental

Sabemos que podemos estimar el valor de una acción utilizando el promedio de las recompensas obtenidas. ¿Cómo implementamos esto con memoria y un tiempo por «time step» constantes?

Iniciamos analizando el estimado para el valor de la acción  $n$  ( $Q_n$ ).  $R_i$  consiste de la recompensa obtenida al seleccionar la recompensa la «i-ésima» vez. Es claro que para la acción  $n$  tomaremos en cuenta todas las acciones previas, que en total serían  $n - 1$  acciones. Por lo tanto, el estimado será igual a:

$$Q_n \doteq \frac{R_1 + R_2 + \dots + R_{n-1}}{n - 1} \quad (5)$$

Para obtener esta implementación, podríamos guardar el valor de todas las recompensas obtenidas y luego estimar el valor. No obstante, si se hace esto, los requerimientos computacionales crecerían con el tiempo (Las dimensiones del array crecerían de manera dinámica y Matlab comenzaría a tirar warnings de «you should pre-allocate for speed»). Para solucionar esto, se tomó la fórmula para estimar el valor de la acción y se manipuló para que la actualización de la estimación requiera del menor tiempo de computación posible.

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\ &= \frac{1}{n} \left( R_n + \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left( R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} (R_n + (n-1)Q_n) \\ &= \frac{1}{n} (R_n + nQ_n - Q_n) \\ &= Q_n + \frac{1}{n} [R_n - Q_n] \end{aligned} \quad (6)$$

Donde:

$Q_n$  : Estimado previo  
 $n$  : Número de estimado  
 $R_n$  : Recompensa recibida previamente

Otra forma de colocar esta última fórmula escrita es la siguiente:

$$\text{Nuevo Estimado} \leftarrow \text{Viejo Estimado} + \text{Step Size} \underbrace{(\text{Objetivo} - \text{Viejo Estimado})}_{\text{Medida de Error}} \quad (7)$$

Usando esta regla de actualización incremental y una selección de acción « $\epsilon$  greedy» se puede escribir un algoritmo bandit completo de la siguiente forma.

---

**Algorithm 1:** Un Algoritmo de Bandit Simple

---

**Inicializar:** para  $a = 1$  hasta  $k$

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

**Loop:**

$A \leftarrow \begin{cases} \arg \max_a Q(a) & \text{con probabilidad } 1 - \epsilon \quad (\text{breaking ties randomly}) \\ \text{Una acción aleatoria} & \text{con probabilidad } \epsilon \end{cases}$

$R \leftarrow \text{bandit}(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$

---

Nota: La función *bandit* consiste de una función que toma una acción y retorna una recompensa según lo dicten las reglas del sistema<sup>4</sup>.

## Tracking en un Problema No Estacionario

El planteamiento previo es útil para problemas estacionarios, donde la probabilidad de recompensa no cambia con el tiempo. No obstante, gran parte de los problemas encontrados en el mundo real consisten de problemas no estacionarios. En estos casos, tiene más sentido ponerle mayor prioridad a recompensas recientes, que a recompensas recibidas en el pasado lejano. Una forma popular de conseguir esto, es utilizar un time-step constante .

$$Q_{n+1} \doteq Q_n + \alpha [R_n - Q_n] \quad (8)$$

Al realizar esta modificación la actualización del estimado actual se puede definir como el promedio ponderado de las recompensas actuales y el estimado inicial  $Q_1$ .

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha [R_n - Q_n] \\ &= \alpha R_n + (1 - \alpha) Q_n \\ &= \alpha R_n + (1 - \alpha) [\alpha R_{n-1} + (1 - \alpha) Q_{n-1}] \\ &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\ &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \\ &\quad \dots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\ &= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha (1 - \alpha)^{n-i} R_i \end{aligned} \quad (9)$$

Como se puede observar, el peso  $\alpha(1 - \alpha)^{(n-i)}$  asignado a la recompensa  $R_i$  depende de hace cuantas recompensas atrás  $(n - i)$  esta recompensa fue experimentada.  $(1 - \alpha) < 1$ , por

---

<sup>4</sup>Para este algoritmo existe un ejemplo programado en Matlab dentro de la carpeta de *Reinforcement Learning Coursera - Ejercicios*. El archivo en cuestión es: *Capitulo2\_TenArmTestbed.mlx*.



lo que el peso asignado a cada recompensa disminuye de forma exponencial a manera que incrementa el número de recompensas recibidas (A medida que el exponente de  $(1 - \alpha)$  aumenta). Esto es comúnmente llamado: «Exponential recency-weighted average» o promedio exponencial ponderado por su carácter reciente.

En algunas situaciones es conveniente variar el step-size  $\alpha$  a lo largo de la ejecución del programa. A este  $\alpha$  variante se le denomina  $\alpha_n$ . En el caso del método incremental, por ejemplo,  $\alpha_n = 1/n$ . Algo interesante es que la convergencia de  $Q$  no está asegurada para toda  $\alpha_n$ . Para asegurar la convergencia (Con probabilidad 1 o total seguridad) se debe cumplir que

$\sum_{n=1}^{\infty} \alpha_n(a) = \infty$	Requerida para garantizar que los pasos son lo suficientemente grandes para eventualmente sobrepasar cualquier transiente inicial
$\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$	Garantiza que los pasos o steps se tornan lo suficientemente pequeños como para asegurar convergencia.

El  $\alpha_n = 1/n$  del método incremental cumple con estas condiciones, pero el  $\alpha_n$  con parámetro constante, no. Esto implica que los estimados este último método nunca convergen completamente ya que cambian según las recientemente obtenidas recompensas. Esto no es siempre malo, ya que como ya fue mencionado previamente, este tipo de adaptación dinámica es necesaria para poder adaptarse a ambientes no estacionarios. Existen funciones  $\alpha_n$  que cumplen con las condiciones, pero muy raras veces se utilizan fuera de entornos académicos.

## Valores Iniciales Optimistas

Como podemos observar, todos los métodos previamente explicados dependen de la estimación inicial del valor de las acciones  $Q_1$ .

$$\begin{aligned}
 Q_{n+1} &= Q_n + \alpha [R_n - Q_n] \\
 &= \alpha R_n + (1 - \alpha) Q_n \\
 &= \alpha R_n + (1 - \alpha) [\alpha R_{n-1} + (1 - \alpha) Q_{n-1}] \\
 &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\
 &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \\
 &\quad \dots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\
 &= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha (1 - \alpha)^{n-i} R_i
 \end{aligned}$$

$$\begin{aligned}
 Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\
 &= \frac{1}{n} \left( R_n + \sum_{i=1}^{n-1} R_i \right) \\
 &= \frac{1}{n} \left( R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\
 &= \frac{1}{n} (R_n + (n-1) Q_n) \\
 &= \frac{1}{n} (R_n + n Q_n - Q_n) \\
 &= Q_n - \frac{1}{n} [R_n - Q_n]
 \end{aligned}$$

Para  $Q_{n+1} = Q_2$  se emplea el estimado previo  $Q_n = Q_1$

En estadística, se puede decir que estas fórmulas tienen un «bias» o sesgo dado por sus valores iniciales. Para el método de sample-average (Con un step size que decrece), el efecto del sesgo eventualmente desaparece, pero para  $\alpha_n = \alpha$  (Constante) el sesgo no desaparece. Esto no es un problema, de hecho puede llegar a ser utilizado para informar al agente sobre el tipo de recompensas que puede esperar. El único problema, es que este estimado inicial consiste de un nuevo parámetro a elegir.

Por ejemplo, si le colocamos un valor alto para la recompensa inicial (Mucho más grande que lo que eventualmente va a conseguir), los métodos de acción-valor (Estimar el valor de una acción y luego elegir una acción en base a esto) se ven motivados a explorar más. No importando cuales sean las acciones tomadas después, la recompensa siempre será más pequeña, entonces «decepcionado», el agente optará por cambiar de acción, probando todas las opciones disponibles varias veces antes de converger. El resultado es una mayor exploración

A este método que promueve la exploración se le denomina: Valor iniciales optimistas. Esta es una técnica muy útil para problemas estacionarios, pero para no estacionarios, se torna inútil ya que el estimado inicial únicamente aplica para la primera instancia del entorno. Cuando el entorno cambie (Por su carácter dinámico), el valor inicial ya no aplica.

#### Unbiased Constant Step Size Trick

Previamente se explicó que el método de «sample-average» no es susceptible al valor inicial, pero no es muy útil en entornos no-estacionarios. Una forma de obtener «lo mejor de dos mundos» es utilizar un step size igual a

$$\beta_n \doteq \alpha / \bar{o}_n \tag{10}$$

Donde  $\alpha$  es una constante y  $o_n$  es un valor acumulativo que inicia en 0 y se actualiza de la siguiente manera.

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha (1 - \bar{o}_{n-1}) \tag{11}$$

Este tipo de «step-size» es inmune al efecto del sesgo inicial, además de causar que el estimado del valor de la acción se torne en un «recency weighted-average» como en el método «sample-average».

#### **Selección de Acción de Límite Superior**

Recordar que en el método de « $\epsilon$ -greedy» existe la probabilidad de que el agente pruebe opciones «no avariciosas» (non greedy), no obstante, no existe preferencia sobre la selección realizada. Claramente sería mejor si se eligieran las acciones «no avariciosas» según su potencial de ser óptima o una buena opción. Para lograr esto podemos elegir una acción tomando en cuenta que tan cercano está un estimado  $Q_n$  a un valor máximo y cual es la incertidumbre del propio estimado

$$A_t \doteq \operatorname{argmax}_a \left[ \underbrace{Q_t(a)}_{\text{Explotación}} + c \underbrace{\sqrt{\frac{\ln t}{N_t(a)}}}_{\text{Exploración}} \right] \quad (12)$$

Donde:

$t$  : Tiempo actual

$N_t(a)$  : Número de veces que una acción  $a$  ha sido seleccionada antes del tiempo  $t$ <sup>5</sup>.

$c > 0$  : Parámetro que controla el grado de exploración.

A este tipo de selección de acción se le denomina «upper confidence bound» o UCB. La idea del UCB consiste en calcular los intervalos de incertidumbre de cada uno de los estimados. Estos intervalos nos dicen: «Calculo que más o menos el valor real de una acción está entre este límite inferior y este límite superior». Lo que hacemos es que nos portamos optimistas y decimos: «Bueno, si el valor puede estar entre estos intervalos y queremos la mayor recompensa posible entonces tomemos la acción con el límite superior más alto». Es optimista porque suponemos que en el mejor caso posible, el valor real de la acción se encontrará exactamente en el límite superior, maximizando la recompensa.

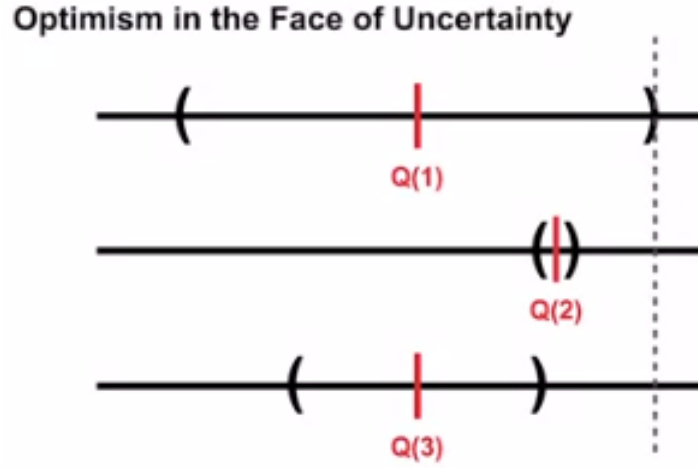


Figura 4: Tres estimados diferentes para el valor de una acción. La línea punteada representa el valor a tomar dado que se trata del límite superior de la incertidumbre

La raíz de la expresión representa la incertidumbre o varianza del estimado del valor de la acción  $a$ . Al maximizar esta sección, entonces se obtiene una especie de «límite superior» para el posible valor real de la acción  $a$ . Algunos puntos importantes a mencionar sobre este método son los siguientes:

- Cada vez que se tome una acción  $a$  el denominador aumenta, por lo que la incertidumbre baja.

<sup>5</sup>Si  $N_t(a) = 0$ , entonces  $a$  es considerada como una acción «maximizadora».

- Si se toma una acción distinta de  $a$  entonces  $t$  incrementa mientras  $N_t(a)$  permanece igual, por lo que el numerador crece en conjunto con la incertidumbre.
- Se usa un logaritmo en el numerador para que el crecimiento del numerador se haga cada vez más pequeño. Debido a esto, todas las acciones se seleccionarán, pero aquellas con estimados bajos o que se han elegido muy seguido se elegirán con cada vez menos frecuencia.

Este método es útil, pero para entornos no-estacionarios y espacios de estados muy grandes, su uso se torna impráctico.

### 6.3.2. Procesos de Decisión de Markov (MDP's)

#### Interfaz Agente-Ambiente

En una tarea a solucionar al «aprendedor» y «tomador de decisiones» se le llama «agente». Todo con lo que interactúa el agente se le denomina «medio ambiente»<sup>6</sup>. Ambos elementos interactúan de forma continua

- **El agente** tomando decisiones y **el medio ambiente** respondiendo a las mismas presentando nuevas situaciones al agente.
- **El medio ambiente** crea recompensas (Valores numéricos) que el agente busca maximizar.

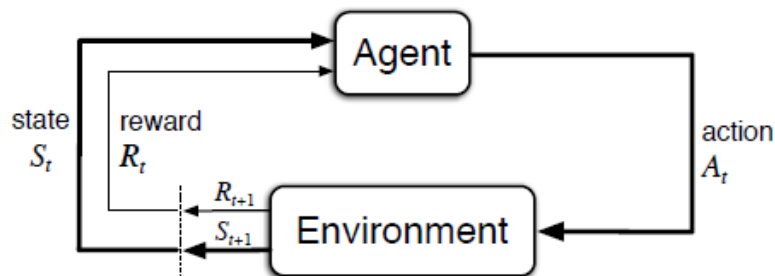


Figura 5: Interacción Agente-Ambiente en un proceso de decisión de Markov [13]

Esta interacción ocurre en pasos

- El agente interactúa luego de intervalos de tiempo discretos (1, 2, 3, 4, 5, ...)
- En cada «time step» el agente recibe información sobre el estado del ambiente:  $S_t$
- En base a este estado selecciona una acción:  $A_t$

<sup>6</sup>Algunos de los términos de Reinforcement Learning tienen un análogo en teoría de control: Agente = Controlador. Ambiente = Planta. Acción = Señal de Control.

- Un «time step» más tarde, el agente recibe una recompensa numérica  $R(t+1)$  como consecuencia de su acción.
- Vuelve a encontrar un nuevo estado.

Si este proceso se definiera como una secuencia de señales esta consistiría de una secuencia similar a la siguiente: Estado, Acción, Recompensa, Estado, ...

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

En un «Markov Decision Process» o MDP finito, los estados, acciones y recompensas tienen un número finito de elementos (Los arrays que codifican estos tienen dimensiones pre-determinadas). Además, las variables  $S_t$  y  $R_t$  tienen distribuciones probabilísticas bien definidas únicamente dependientes del estado o acción pasados. En otras palabras, la probabilidad de que aparezca un estado y recompensa específicos, únicamente depende del estado y acción previos.

Podemos decir que esta no es una restricción del MDP, pero del estado actual. El estado debe incluir información sobre todos los aspectos de la interacción «agente-ambiente» que pueden llegar a generar un cambio en el futuro. Si esto es cierto, se dice que el estado tiene la «propiedad de Márkov»

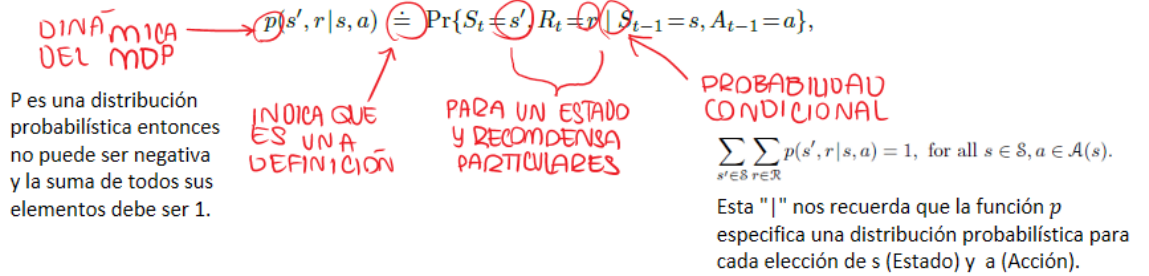


Figura 6: Dinámica de un MDP

Con esta función  $p$  (Dinámica de MDP) uno puede calcular otras cosas se deseen saber sobre el ambiente, por ejemplo:

- Probabilidades de transición entre estados

$$p(s' | s, a) \doteq \Pr\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a) \quad (13)$$

- La recompensa esperada para una pareja «estado-acción»

$$r(s, a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a) \quad (14)$$

- Las recompensas esperadas para el triplete «estado-acción-siguiente estado»

$$r(s, a, s') \doteq \mathbb{E} [R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} \quad (15)$$

Comúnmente se utiliza la notación dependiente de 4 argumentos (La primera antes de estas tres), pero estas otras pueden ser útiles ocasionalmente.

El framework de un MDP es muy flexible y aplicable a una variedad de problemas. Por ejemplo:

- Los time steps no necesariamente corresponden a intervalos uniformes de tiempo, pueden referirse a etapas de toma de decisiones y «actuación».
- Las acciones tomadas pueden de decisiones simples (Voltajes a aplicar) o de decisiones complejas (Ir o no al colegio). Pueden ser cualquier decisión que deseemos aprender a hacer.
- Los estados pueden estar definidos por medidas de bajo nivel (Medidas de sensores) o medidas más abstractas y de alto nivel (Descripciones simbólicas de objetos en un cuarto). Un estado puede consistir de cualquier conocimiento útil para tomar una decisión o tomar una acción.

La frontera entre agente y ambiente no es típicamente la misma que la frontera física de un robot o animal. En un robot por ejemplo, los motores, sensores y extremidades deberían considerarse parte del ambiente, en lugar de formar parte del agente como se esperaría. Las recompensas también son calculadas «dentro» de la frontera física de un robot o animal, pero no se consideran parte del agente como tal.

Como regla general, *cualquier cosa que no pueda ser arbitrariamente cambiada por el agente, se considera como exterior y por lo tanto, parte de su ambiente.*

No todo con lo que el agente interactúa es desconocido. Generalmente el agente está consciente de la forma en la que las recompensas son obtenidas en base a sus acciones y estado actual. Siempre consideramos el cálculo de recompensa como algo externo al agente ya que define la tarea que debe resolver el agente y por lo tanto, está fuera de su control el poder cambiarla de forma arbitraria.

- Por ejemplo: Un agente sabe todo sobre su ambiente, pero aun así tiene problemas para resolver la tarea de reinforcement learning que se le da. Eg. Alguien puede saber cómo funciona un cubo Rubik's, pero aun así le puede costar resolverlo.

La frontera «agente-ambiente» representa el límite del control absoluto del agente, no el límite de su conocimiento. En la práctica, esta frontera se establece una vez se ha elegido el juego de estados, acciones y recompensas que formarán parte del proceso de toma de decisiones que se desea resolver. El MDP framework, es una abstracción del problema

de «*aprendizaje orientado a objetivos basado en interacción*». Este problema propone que elementos como sensores, memoria y aparatos de control, además del objetivo a cumplir se pueden reducir como 3 señales:

- Acciones: Decisiones del agente
- Estados: Fundamentos utilizados para tomar decisiones
- Recompensas: Señal que define el objetivo del agente

## Metas y Recompensas

*Hipótesis de recompensa:* Todo aquello que podemos definir como objetivos o propósitos puede llegar a interpretarse como la maximización del valor esperado de la suma acumulativa de una señal escalar recibida denominada «Recompensa».

Algunos ejemplos de recompensas son:

- Robot aprendiendo a caminar: Recompensa en cada time step proporcional al desplazamiento hacia adelante del robot.
- Robot aprendiendo a escapar de un laberinto: Recompensa de -1 por cada time step que esté dentro del laberinto para que aprenda a escapar rápido.
- Robot aprendiendo a navegar un espacio: Recompensa negativa cuando hay un choque.

*La recompensa es una forma de comunicarle al agente qué es lo que se desea conseguir, no cómo se desea conseguirlo.* Por ejemplo: Jugando ajedrez, no es recomendado que al agente se le recompense por comer piezas, sino solo por ganar el juego. Si se le dan recompensas por completar «sub-objetivos», puede que el agente aprenda a comer muchas piezas pero eventualmente pierda.

## Retornos y Episodios

El objetivo de un agente es el siguiente

- Informalmente: Maximizar la recompensa acumulativa que este recibe a largo plazo.
- Formalmente: Si la secuencia de recompensas obtenida se define como  $R(t+1), R(t+2), R(t+3), \dots$ , deseamos maximizar el «retorno esperado»  $G_t$  o una función de la secuencia de recompensas.

En su forma más simple, el retorno es

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (16)$$

Donde:

$T$  : Time step final

Esta función es aleatoria, debido al comportamiento dinámico de la MDP. Debido a esto maximizamos el «retorno esperado». Esta forma de estructurar el retorno, es útil en problemas donde existe una noción natural de un «time step final». En otras palabras, es útil para interacciones «agente-ambiente» que se pueden separar naturalmente en subsecuencias o episodios. Por ejemplo, un episodio podría consistir de las partidas en un juego o intentos para recorrer un laberinto.

Cada episodio termina con un «*estado terminal*», seguido por un reset al estado inicial estándar o a una elección entre una distribución estándar de los diferentes estados iniciales.

Un episodio iniciará de la misma manera, independientemente de la forma en la que terminó el último episodio. Entonces se puede considerar que todos los episodios terminan en el mismo «estado terminal», pero con diferentes recompensas en el camino. A tareas con este tipo de episodios se les denomina «*tareas episódicas*». En estas existen:

$S$  : Estados no terminales  
 $S^+$  : Estados terminales  
 $T$  : Tiempo de finalización

Tareas que no pueden separarse en episodios definidos se denominan «*tareas continuas*». Para estas tareas es mucho más difícil definir una función de retorno ya que  $T = \infty$  luego de un largo tiempo. Para estas tareas se utilizan «descuentos» porque de lo contrario las sumatorias no serían finitas.

Descuentos: Un agente trata de seleccionar acciones a manera de maximizar la suma de las recompensas descontadas que recibe a lo largo del tiempo. En particular, elige  $A_t$  para maximizar el *retorno descontado*

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (17)$$

Donde:

$0 \leq \gamma \leq 1$  : Tasa de descuento

Los valores de recompensa recientes tienen más peso que los futuros ya que los futuros están multiplicados por potencias cada vez más altas de gamma.

Tasa de descuento: Determina el valor presente de recompensas futuras

- Una recompensa recibida  $k$  «time steps» en el futuro valdrá  $\gamma^{(k-1)}$  veces lo que valdría si se recibiera inmediatamente.
- Si  $\gamma < 1$  la suma infinita del retorno descontado tiene un valor finito, mientras la secuencia de recompensas esté acotada.
- Si  $\gamma = 0$  el agente solo se enfoca en maximizar las recompensas inmediatas. El agente se vuelve «miope».
- Cuando  $\gamma \leftarrow 1$  el agente toma más en cuenta recompensas futuras.



Un retorno puede calcularse de manera iterativa de la siguiente manera:

$$\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned} \tag{18}$$

A pesar que la sumatoria tiende al infinito, esta es finita si las recompensas son constantes y distintas de cero (Siempre y cuando  $\gamma < 1$ ). Si la recompensa es +1 el retorno sería:

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma} \tag{19}$$

## Notación Unificada para Tareas Episódicas y Continuas

Como pudimos ver hay dos tipos de tareas de aprendizaje reforzado: Episódicas y continuas. Cada una tiene una notación única, pero a veces se utilizan ambas en el mismo contexto. Para facilitar su manejo matemático se establece una notación que permita hablar de la misma manera para ambos casos.

En lugar de considerar una tarea como una larga secuencia de «time steps», ahora se considera como una serie de episodios, cada uno conformado por un número finito de «time steps».

Debido a esto, ahora la notación para  $S_t$ ,  $R_t$ ,  $A_t$  pasa a ser:  $S(t, i)$ ,  $R(t, i)$ ,  $A(t, i)$ .

Donde:

- $i$  : Número de episodio
- $t$  : Número de time step del episodio actual. Siempre inicia de 0 cada vez que se pasa a un nuevo episodio

A pesar de esta notación, la mayor parte del tiempo se habla de un único episodio en particular o de cosas que son aplicables a todos los episodios. Debido a esto, se tiende a obviar la «i» que hace referencia al número de episodio y se regresa a la notación previa.

Ahora hay otro problema: El retorno  $G_t$  se define de manera diferente para tareas episódicas y continuas.

- En tareas episódicas, el retorno es una suma finita.
- En tareas continuas es una suma infinita.

Para unificar ambas expresiones, se considera que, cuando el agente llega al estado terminal, este entra a un «estado de absorción» que solo transiciona a sí mismo y genera recompensas de 0.

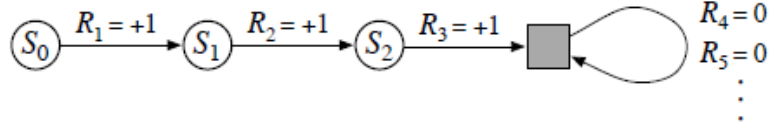


Figura 7: Sumar las recompensas de los primeros 3 estados es lo mismo que sumar la serie infinita, ya que luego se pasa al estado de absorción (Cuadro gris).

Esto incluso puede aplicarse en conjunto con «descuentos». Entonces, obviando la «i» de número de episodio y tomando en cuenta la posibilidad que  $\gamma = 1$  si la suma continúa definida, el retorno se define como:

$$G_t \doteq \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (20)$$

Donde:

- $t$  : Time step actual
- $k$  : Siguiente time step

## Policies y Funciones de Valor

Casi todos los algoritmos en aprendizaje reforzado involucran la estimación de una «*función de valor*»: Función que toma los estados (O las parejas estado-acción) y estima «que tan bueno» es para el agente estar en un estado específico según sus posibles estados futuros (O que tan bueno es realizar una acción específica en un estado dado).

La medida de «que tan bueno» viene de la cantidad de recompensas futuras que puede llegar a esperar el agente (O el retorno esperado  $G_t$ ). Las recompensas dependen de dos factores: Las acciones que tomará el agente y el estado en el que se encuentra.

La función que establece la forma de actuar del agente, se llama «*policy*». En otras palabras, una policy consiste de la forma en la que un agente selecciona una acción. Existen dos tipos:

- Determinística: Mapea o asigna una acción a cada estado. El agente puede seleccionar la misma acción en dos estados diferentes. También se pueden obviar acciones completamente.
- Estocástica:

---

## Entorno de Simulación (Toolbox) para Entrenamiento y Realización de Pruebas de Algoritmos de Enjambre

---

Para facilitar el proceso de diseño de estos controladores automático, se implementó una serie de herramientas diseñadas específicamente para agilizar el proceso de realización de pruebas, validación de los datos, etc. A esta se le denominó PSO Toolbox. En su versión actual (v1.0), esta aún no tiene la estructura de una Toolbox tradicional de Matlab.

Para correrla, abrir el archivo PSO\_Toolbox.mlx Para asegurar compatibilidad utilizar Matlab 2018b o posteriores. Dado que se trata de un livescript, es recomendable utilizar versiones más recientes, ya que el rendimiento del script mejora sustancialmente. La versión más reciente de Matlab en la que se pudo probar el archivo fue Matlab 2020a.

Casi todas las líneas en el script están documentadas, sin embargo, a continuación se presenta una breve guía del Toolbox.

### 7.1. Parámetros Generales

El script ofrece múltiples parámetros para que el usuario ajuste la situación según lo desea.

- NoParticulas: Cantidad de robots a simular
- CostFunc: Función de costo a minimizar por las partículas. Puede seleccionarse por medio de un dropdown menu o escribirse manualmente. Se ofrecen 10 funciones de costo distintas: 9 funciones «Benchmark» y 1 función dependiente de los obstáculos colocados en el escenario denominada «APF». Para más información sobre las opciones disponibles escribir en consola: `help CostFunction`.

- DimsMesa: Dimensiones de la mesa sobre la que se moverán los robots
- EndTime: Tiempo que durará la simulación
- EnablePucks: Booleano. Habilita o deshabilita la animación de los robots.

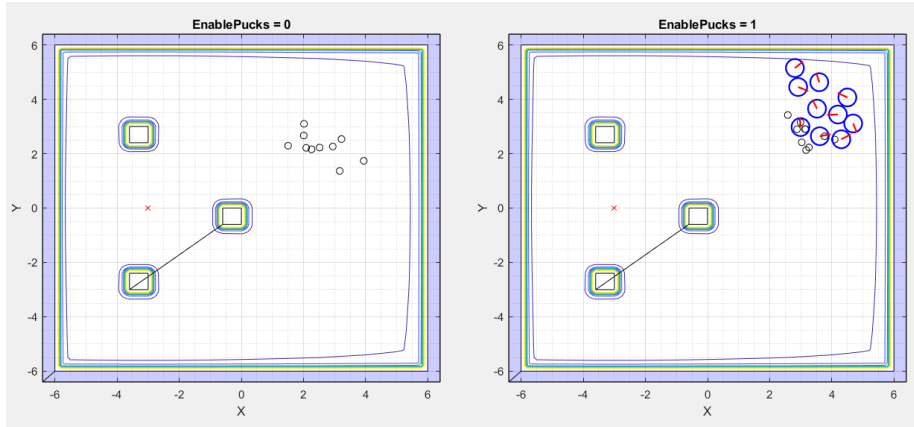


Figura 8: Efectos de alterar el parámetro *EnablePucks*.

- RadioLlantasPuck: Radio de las ruedas que emplea el robot diferencial
- RadioCuerpoPuck: Distancia del centro del robot a sus ruedas.
- PuckVelMax: Velocidad máxima del puck en función de sus actuadores
- ModoVisualización: 2D, 3D o None. El 3D se recomienda para observar más fácilmente la forma de la función de costo. El 2D es más útil para observar el movimiento de los marcadores y/o robots.

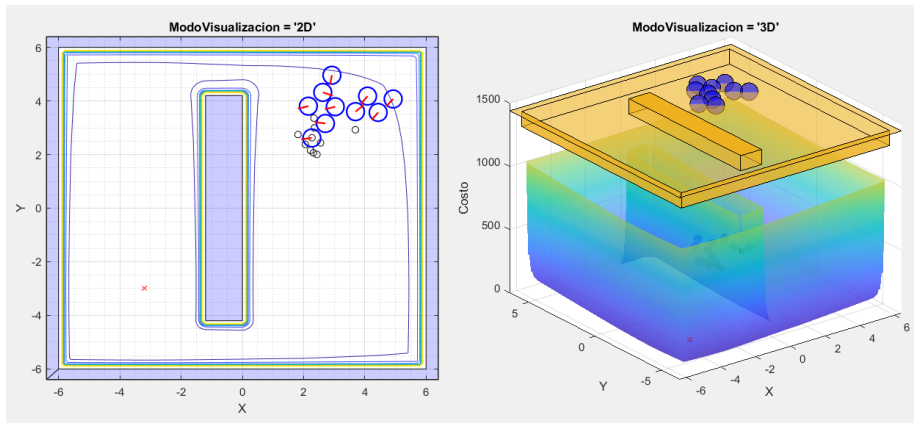


Figura 9: Efectos de alterar el parámetro *ModoVisualizacion*.

## 7.2. Obstáculos

Basado en la implementación de «Artificial Potential Fields» de Juan Pablo Cahueque. Si se seleccionó «APF» como función de costo, esta funcionalidad permite que el usuario diseñe los obstáculos a posicionar en la mesa de trabajo. Se ofrecen 5 opciones diferentes.

- Polígono: El usuario puede dibujar el polígono que desee. Para colocar un vértice hacer click. Para finalizar, cerrar el polígono regresando al primer vértice colocado y haciendo click en este punto.

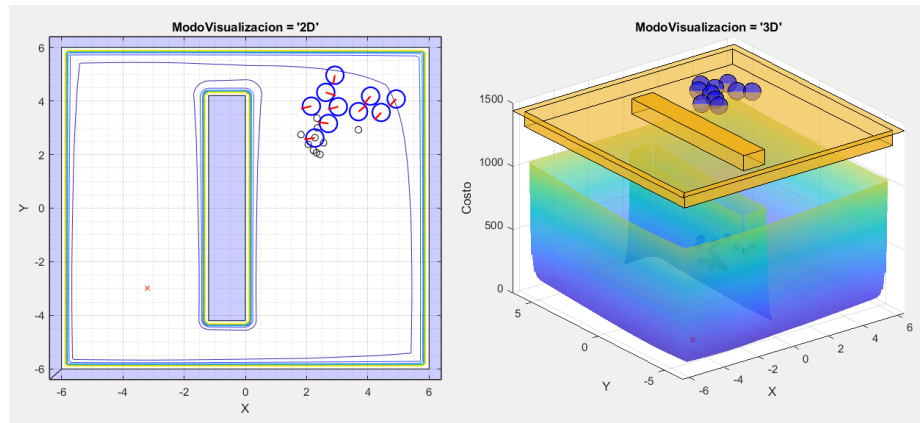


Figura 10: Creación de obstáculo poligonal.

- Cilindro: Coloca un cilindro de radio 2 en el centro de la mesa de trabajo (El radio puede cambiarse manualmente).
- Caso A: Réplica del escenario A utilizado en la tesis de Juan Pablo.

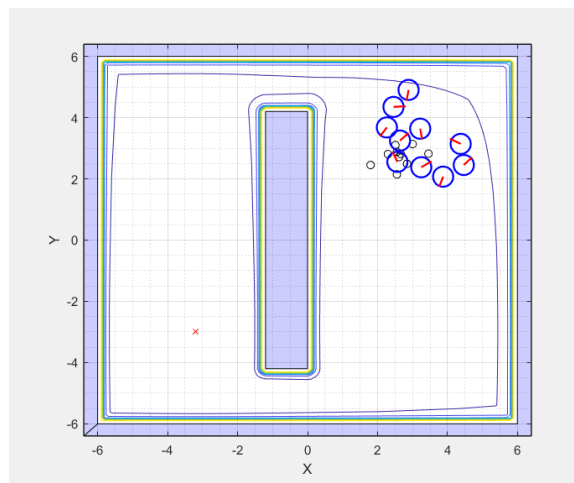


Figura 11: Caso A en tesis de Juan Pablo Cahueque

- Caso B: Réplica del escenario B utilizado en la tesis de Juan Pablo.
- Caso C: Réplica del escenario C utilizado en la tesis de Juan Pablo. Debido a que consiste de múltiples obstáculos, aún existe un error en su renderización. De aquí viene el artefacto de la línea negra observada entre obstáculos y su falta de coloración.

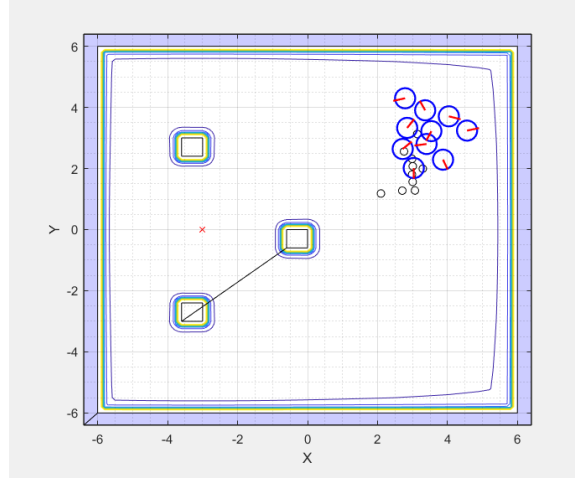


Figura 12: Caso C en tesis de Juan Pablo Cahueque

### 7.3. Tipo de Restricción

Como se mencionó previamente, el algoritmo canónico de PSO es dependiente de los parámetros  $\omega$  y  $\chi$ . Existen múltiples maneras de elegir estos parámetros, pero en el toolbox se ofrecen tres opciones:

- Inercia: Si se desea abandonar el esquema que asegura la convergencia propuesto por Clerc [10], el usuario puede obviar la ecuación y utilizar el valor de  $\omega$ , o inercia, que desee. Se ofrecen 5 tipos diferentes de inercia. Para más información escribir en consola: *help ComputeInertia*
- Inercia: Si se desea abandonar el esquema que asegura la convergencia propuesto por Clerc [10], el usuario puede obviar la ecuación y utilizar el valor de  $\chi$ , o inercia, que desee. Se ofrecen 5 tipos diferentes de inercia. Para más información escribir en consola: *help ComputeInertia*

$$\begin{aligned}
 \omega &= \chi & \chi &= \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \\
 C_1 &= \chi\phi_1 & \phi &= \phi_1 + \phi_2 \\
 C_2 &= \chi\phi_2
 \end{aligned} \tag{21}$$

- Mixto: Uso simultáneo de un tipo de inercia (Por defecto exponencialmente decreciente), en conjunto con los parámetros de restricción propuestos por Clerc. Utilizado por Aldo en su tesis.

## 7.4. Colisiones

Un entorno de simulación realista es necesario para obtener resultados útiles al momento de realizar pruebas. Debido a esto, se implementó «Collision Detection» entre los robots. Durante cada iteración, los robots revisan la distancia entre ellos (Para más información escribir en consola: `help getDistsBetweenParticles`) y si esta es menor a 2 radios de E-Puck, los robots se clasifican como «en colisión». Seguido de esto se procede resolver las colisiones, alejando a los robots el uno del otro hasta eventualmente resolver todas las colisiones existentes.

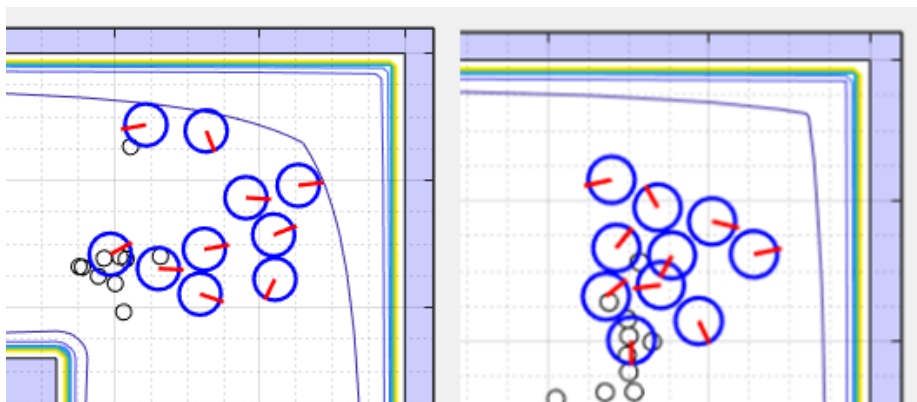


Figura 13: Con solución de colisiones (Izquierda) y sin solución de colisiones (Derecha)

Desgraciadamente, debido a que al alejar un robot del otro se pueden llegar a crear más colisiones, en algunas ocasiones el algoritmo puede no converger en una solución. Por lo tanto, **el algoritmo implementado es inestable y si no se restringe puede llegar a trabar Matlab**. Para controlar esto se le colocó un número máximo de iteraciones en las que puede llegar a producir una solución válida. Con esta «solución», el algoritmo funciona relativamente bien aunque puede producir errores frecuentemente.

Si se desea, el usuario puede acceder a la función `SolveCollisions.m` y cambiar el parámetro `IteracionesMax`. Los errores disminuyen al incrementar el número de iteraciones, pero el tiempo computacional requerido incrementa. En futuras versiones del Toolbox se desea implementar un algoritmo de «Collision Detection» mucho más robusto como «Speculative Collisions» que también incluya elementos como las paredes o los obstáculos como tal.

## 7.5. Controladores

Como es explicado por Aldo [14] en su tesis, el acoplar el movimiento de un robot diferencial directamente al desplazamiento de una partícula PSO no es recomendable. Las

partículas se desplazan de manera muy agresiva, por lo que los robots podrían quemar sus motores en el proceso de intentar seguir su paso. Entonces, los E-Pucks no siguen directamente las posiciones de las partículas PSO, sino que utilizan su dirección como una sugerencia de hacia donde ir. Debido a esta diferencia, a las partículas del algoritmo PSO se les pasa a denominar «Marcadores PSO» en el Toolbox.

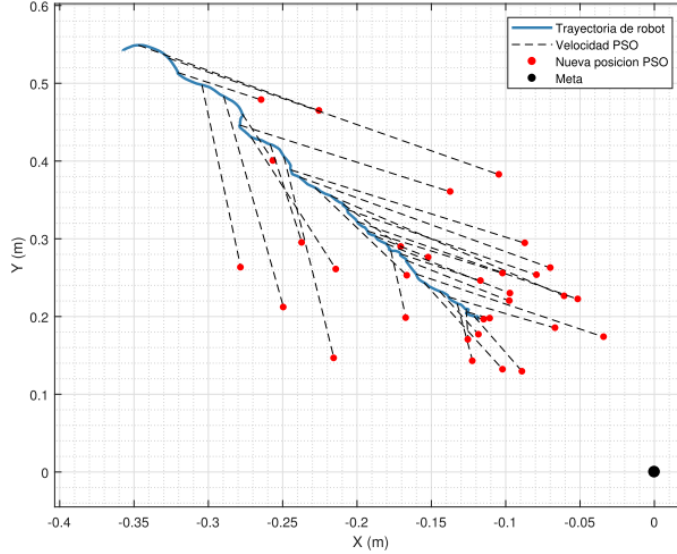


Figura 14: Marcadores PSO

Los controladores son los encargados de seguir estas sugerencias hasta llegar a la meta. En la Toolbox se ofrecen dos opciones: Un controlador LQR y un LQI (Ya que estos fueron los que obtuvieron los mejores resultados en la tesis de Aldo). La salida de estos controladores es la velocidad lineal y angular de los E-Pucks. Normalmente estas cantidades deben ser mapeadas por medio de estas ecuaciones

$$\dot{\varphi}_R = \frac{2v + 2\omega l}{2r}, \quad \dot{\varphi}_L = \frac{2v - 2\omega l}{2r} \quad (22)$$

a las velocidades angulares de las ruedas del robot. No obstante, en el caso de la Toolbox, las velocidades se utilizan directamente para guiar el movimiento de los robots. Para más información escribir en consola `help getControllerOutput`.

## 7.6. Criterios de Convergencia

La función `EvalCriteriosConvergencia.m` permite que el usuario diseñe sus propios criterios de convergencia y los agregue a una lista de condiciones que se evalúan durante cada iteración. La salida de la función es una lista de cuales son los criterios cumplidos, en conjunto con su nombre. Por defecto la Toolbox ofrece 4 opciones

1. «Mínimo Alcanzado»: Cierta porcentaje de partículas llega a alguno de los mínimos de la función.



2. «Mínimo Alcanzado (80 %)»: Criterio 1 cumplido y ha transcurrido el 80 % de las iteraciones máx.
3. «Posición Convergíó»: Todas las partículas se han quedado «quietas» o se han movido poco desde la última iteración
4. «Iter. Máx Alcanzadas»: Se ha alcanzado el número máximo de iteraciones.

Para más información escribir en consola `help EvalCriteriosConvergencia`

## 7.7. Análisis de Resultados

Al finalizar la simulación, el usuario puede analizar los resultados obtenidos haciendo uso de 4 gráficas distintas

### 7.7.1. Evolución del Global Best

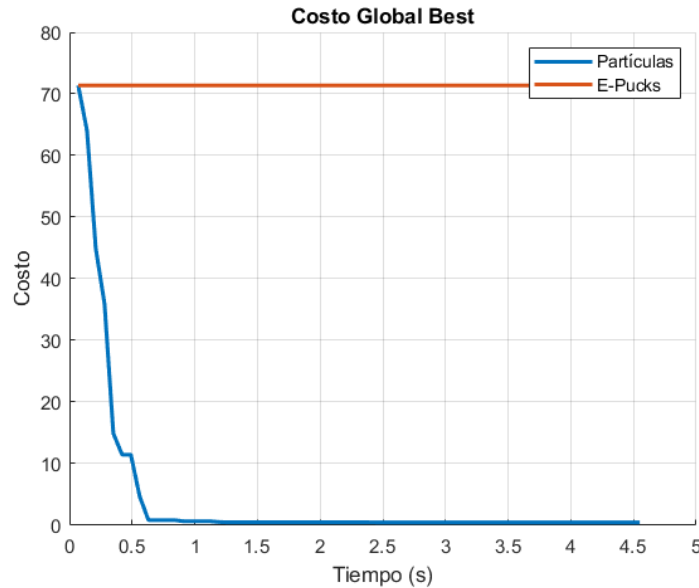


Figura 15: Evolución de la minimización hacia el global best de la función

Utilizada para determinar si los robots y las partículas efectivamente minimizan la función de costo que se eligió. Dada la naturaleza del movimiento de los robots, muy comúnmente la curva de los robots parece estar «atrasada» con respecto a la de las partículas o marcadores PSO.

### 7.7.2. Análisis de Dispersión de Partículas

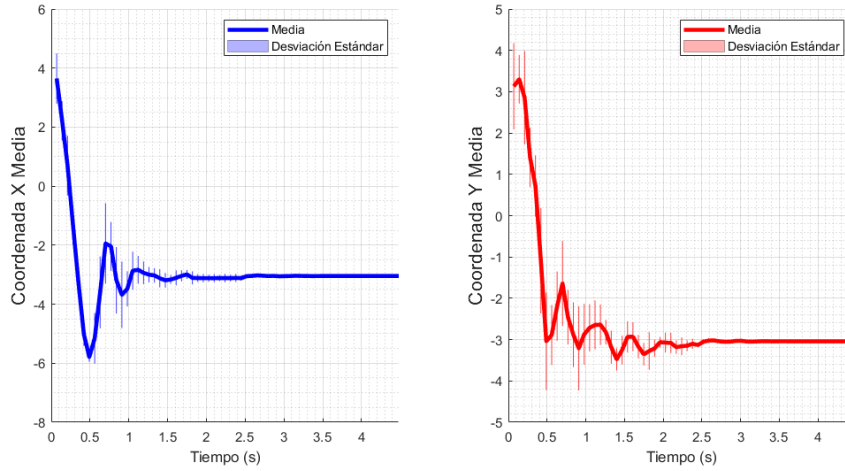


Figura 16: Dispersión de las partículas sobre el eje X y Y.

Dos cualidades importantes de las partículas del PSO es su capacidad de exploración y la precisión de su minimización. Con estas gráficas, la precisión se puede evaluar viendo la línea gruesa coloreada y la exploración utilizando las líneas correspondientes a la desviación estándar. Si las líneas gruesas se estabilizan en las coordenadas de la meta, las partículas son precisas. Si la desviación estándar es muy pronunciada, las partículas exploran minuciosamente el área de trabajo antes de converger.

En el caso presentado, por ejemplo, las partículas son precisas y convergen con rapidez, aunque exploran poco.

### 7.7.3. Velocidad de Motores

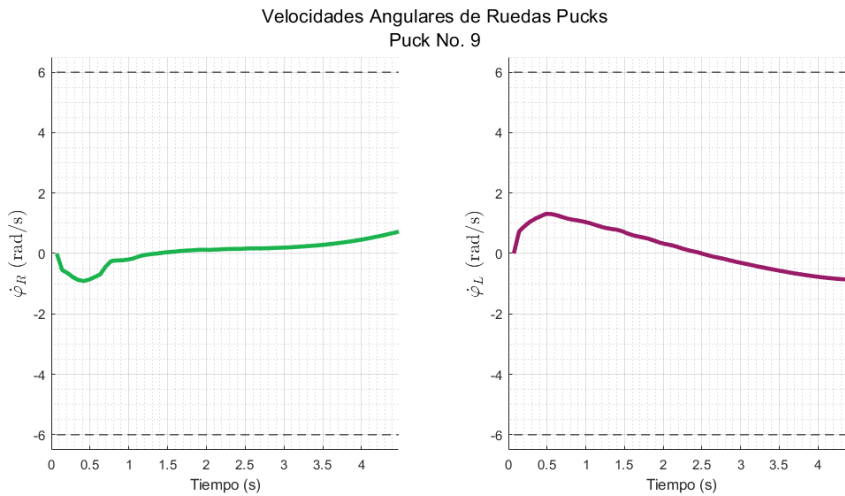


Figura 17: Velocidad angular observada en los motores del puck con los picos más altos de velocidad en dicha corrida

Utilizando la cinemática inversa de un robot diferencial se calculan las velocidades angulares de las ruedas de todos los robots (Ecuación 22).

La Toolbox obtiene las velocidades angulares medias de todas las ruedas y determina cual fue el robot con las velocidades más altas. Toma este robot como selección y grafica la evolución de las velocidades angulares de sus dos ruedas. Útil para analizar si los actuadores del robot crítico presentan saturación. Como ayuda se incluyen líneas punteadas, las cuales consisten de los límites de velocidad con los que cuenta el robot (Basado en *PuckVelMax*).

#### 7.7.4. Suavidad de Velocidades

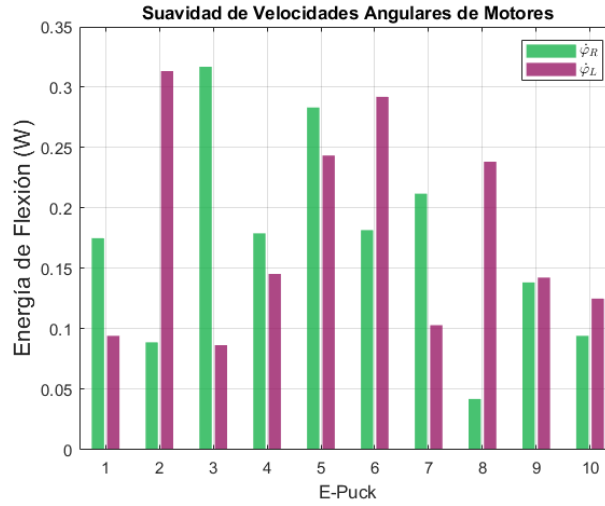


Figura 18: Energía de flexión observada en las velocidades angulares de las ruedas de cada puck.

Basado en el criterio de evaluación empleado por Aldo en su tesis. Se realiza una interpolación de los puntos que conforman la curva de velocidades angulares de las ruedas, y luego se calcula la energía de flexión de la curva. Si la energía de flexión es baja, la suavidad de operación es mucho mayor. Prueba ideal para diagnosticar cuantitativamente la suavidad de operación.

## 7.8. Grabación de Videos / Frames

Para facilitar la presentación de resultados, la Toolbox cuenta con dos opciones de exportación de gráficas: SaveVideo y SaveFrames.

- SaveFrames: Guarda cada una de las frames generadas durante el proceso de animación en el directorio raíz.
- SaveVideo: Crea un video a partir de las frames generadas durante el proceso de animación. Por defecto el framerate es de 30 y el formato es mp4.

Si se desea observar una demostración de los resultados obtenidos utilizando estas funciones, consultar el documento de ReadMe, presente en el repositorio para esta Toolbox.

Advertencia: Durante el proceso de creación de videos, la animación corre más lento.

---

## Obtención de Datos para Entrenamiento de Red Neuronal

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



---

## Controladores Basados en Redes Neuronales

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.





---

### Controlador de Hiperparámetros Basado en Reinforcement Learning

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



---

## Validación de Resultados

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



---

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

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



## CAPÍTULO 13

---

Conclusiones

---





- Implementar un modo manual en el PSO toolbox, el cual permita que el usuario controle como un robot a control remoto a los E-Pucks. Este modo puede ser utilizado luego para realizar «Inverse Reinforcement Learning», donde a partir de un ejemplo dado por los diseñadores, el algoritmo intenta deducir porque ese ejemplo dado consiste de la policy óptima a seguir.
- El objetivo de un agente con Reinforcement Learning es encontrar la policy óptima que genere la mayor recompensa a largo plazo. Existen métodos que permiten encontrar de manera estocástica una policy óptima, dotando a diferentes agentes con diferentes policies y luego utilizando algoritmos genéticos para poder seleccionar entre las mejores policies hasta eventualmente encontrar la óptima.



- [1] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klapotocz, 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.
- [14] A. A. Nadalini, L. A. Rivera y M. Z. Arenales, «Implementation of a PSO trajectory planner using kinematic controllers that ensure smooth differential robot velocities»,

## CAPÍTULO 16

---

Anexos

---

### 16.1. Cosos cosos cosos

