Algoritmo Autónomo de Control de Trayectorias en Escenarios con Obstáculos para Mini Drones Utilizando Técnicas de Aprendizaje por Refuerzo.

Brayan Eduardo Baron Pacheco

Código:11481729710

Universidad Antonio Nariño

Programa Ingeniería Mecatrónica

Facultad de Ingeniería Mecánica, Electrónica y Biomédica

Bogotá, Colombia

2022

Algoritmo Autónomo de Control de Trayectorias en Escenarios con Obstáculos para Mini Drones Utilizando Técnicas de Aprendizaje por Refuerzo.

Brayan Eduardo Baron Pacheco

Proyecto de grado presentado como requisito parcial para optar al título de:

Ingeniero Mecatrónico

Director (a):

Ph.D., MSc, Ingeniero Christian Camilo Erazo Ordoñez

Línea de Investigación:

Sistemas Dinámicos y de Control.

Grupo de Investigación:

Grupo de Investigación en Bioinstrumentación y Control (GIBIO).

Universidad Antonio Nariño

Programa Ingeniería Mecatrónica

Facultad de Ingeniería Mecánica, Electrónica y Biomédica

Bogotá, Colombia

2022

NOTA DE ACEPTACIÓN

El trabajo de grado titulado

_____,

Cumple con los requisitos para optar

Al título de _____.

_____

Firma del Tutor

_____

Firma Jurado

_____

Firma Jurado

Fecha:_____.

Nuestra mente siempre lo olvida, pero el futuro no existe y eso es algo increíble y sanador.

Resumen

En el presente proyecto se desarrolla y simula un algoritmo de seguimiento donde su función principal es el guiado autónomo de mini drones, este algoritmo se diseñó con el enfoque de aprendizaje por refuerzo (RL) por medio de redes neuronales. Todo el diseño fue realizado con la herramienta computacional Matlab, la cual cuenta con diversos toolboxes y herramientas que ayudan a él modelamiento tanto del dron como el de la inteligencia artificial, como se enseña en todo el capítulo 3, donde allí se describen a profundidad la serie de pasos necesarios para el diseño del algoritmo. En este software se utilizó el modelo de Simulink ya existente llamado Quadcopter Project, enfocado en la serie de mini drones Parrot Mambo.

Una vez que se diseñó y elaboró la inteligencia artificial se procedió a entrenarla como se muestra en la sección 3.4, para que finalmente tras varias horas de entrenamiento se tuviera consolidada una I.A capaz de guiar a el mini drone por dos trayectorias distintas, sin colisionar con los límites del entorno. Por último, en la sección 4 se presentan los resultados obtenidos para cada uno de los modelos de pista propuestos, teniendo un error del 0.72 % en el seguimiento de la primera trayectoria y 5.4% de error en la segunda.

Palabras clave: Mini Drone, Redes Neuronales, Control Autónomo, Aprendizaje por Refuerzo.

Abstract

This project develops and simulates a tracking algorithm where its main function is the autonomous guidance of mini drones, this algorithm was designed with the approach of reinforcement learning (RL) through neural networks. The entire design was performed with the MATLAB computational tool, which has several toolboxes and tools that help the modeling of both the drone and the artificial intelligence, as taught throughout Chapter 3, where the series of steps necessary for the design of the algorithm are described in depth. In this software we used the existing Simulink model called Quadcopter Project, focused on the Parrot Mambo mini drone series.

Once the artificial intelligence was designed and elaborated, we proceeded to train it as shown in section 3.4, so that finally after several hours of training we had consolidated an AI capable of guiding the mini drone through two different trajectories, without colliding with the limits of the environment. Finally, section 4 presents the results obtained for each of the proposed track models, having an error of 0.72% in the tracking of the first trajectory and 5.4% error in the second one.

Keywords: Mini Drone, Neural Networks, Autonomous Control, Reinforcement Learning.

Contenido

Pág.

Lista de figuras

Pág.

Lista de tablas

Pág.

1. Introducción

En el desarrollo y aplicación de las tecnologías actuales se está haciendo cada vez más relevante la implementación de vehículos aéreos no tripulados como los drones, los cuales se les puede dar uso en aplicaciones como identificación de fallas en estructuras [1], la búsqueda y rescate [2], agricultura de precisión [3], mapeo de terrenos [4], entre otros. Donde en la actualidad está sobresaliendo la aplicación de tecnologías de inteligencia artificial en estos dispositivos.

Específicamente cuando se requiere diseñar un sistema de control para mini drones (el cual es un sistema subactuado), se debe tener en cuenta parámetros como el espacio de acción [5], donde existen diferentes tipos de controladores que se enfocan en que el mini drone realice un seguimiento lo más fielmente posible a una trayectoria previamente definida, a partir de un de control automático, que puede ser inteligente o convencional haciendo uso de controladores PID o de un controlador de lógica difusa [6]. Por otro lado, existen modelos matemáticos un tanto complejos para la controlabilidad de drones, los cuales son basados en Histogramas de Gradiente Orientados (HOG) y en Filtros de Partículas, junto a la implementación de algoritmos de Árboles de Exploración Rápida (RTT), con la particularidad de que el dron controlado puede llegar a presentar cierto tipo de inestabilidad frente a perturbaciones externas del entorno, algo que es común si se utilizan controladores de este tipo o también como los PID [7].

En algunas aplicaciones de búsqueda y rescate se recurre a algoritmos de detección en base a los SSD (Single Shot Detector), considerados como uno de los más rápidos en poder realizar la detección de elementos en una imagen, con la ayuda de una técnica que se encarga de entrenar redes convoluciones pre entrenadas, llamada transferencia de aprendizaje [8]. Allí es importante resaltar la importancia de la base de datos que se va utilizar para poder realizar el proceso de reconocimiento de objetos, ya que en algunos

casos es fundamental contar con más de 80.000 imágenes para realizar el reconocimiento de tan solo cuatro elementos [9].

Por lo general en las implementaciones de un controlador de vuelo autónomo para drones se suelen utilizar tarjetas de desarrollo como Arduino o similares, las cuales son encargadas de realizar el cálculo del modelo matemático que se haya implementado, en donde se puede presentar el caso de que al momento de utilizar un controlador tenga cierta complejidad de cálculo, estas tarjetas de desarrollo se verán limitadas por el tiempo de procesamiento derivando a problemas como la creación de ruido a las señales de control [10]. Sin embargo, se pueden aplicar distintos enfoques para el control de un dron, ya que con la ayuda de redes neuronales basándose en el método Deep Learning, se pueden crear sistemas de reconocimiento del entorno, los cuales proporciona una mejora a la hora especificar todos los parámetros que debe tener en cuenta el control de movimiento del dron [11].

1.1 Descripción del Problema

Cuando se diseñan controladores para drones enfocados en el seguimiento autónomo de trayectorias, puede llegar a ser algo complejo si se utiliza un modelo típico de control, ya que se debe tener en cuenta una gran serie de variables de entrada para ser procesadas y que con estas entradas la salida de control resultante sea un movimiento deseado por parte del dron.

Por ejemplo, cómo se evidenció anteriormente la mayoría de los sistemas de control hacían uso de un controlador PID, este controlador para sistemas en entornos estáticos es adecuado donde su fácil aplicación es resaltable, pero si se tiene un ambiente o entorno de acción con datos dinámicos este tipo de controlador se queda limitado al momento de cumplir con su objetivo, ya que el controlador PID no es adaptativo dejando problemas de poca eficacia en el seguimiento de trayectorias [12]. Al momento de evaluar el comportamiento del error en un proceso dinámico, se tiene que en algunos casos el error acumulado aumenta nueve veces si el proceso se retrasa un 10 % y si el proceso sigue variando el controlador PID ya no sería viable para la aplicación de control [13]. Si se compara el uso de controladores PID y los controladores basados en el aprendizaje por refuerzo, se evidencia que no existe una diferencia muy relevante en cuanto a rendimiento en sistemas sin perturbaciones, teniendo unos valores de RMSE muy semejantes en diferentes escenarios estáticos [14] [15].

Sin embargo, el uso de redes neuronales en un sistema de control presenta una ventaja frente a las perturbaciones de un espacio de acción continuo, además con el presente proyecto se pretende elaborar una alternativa distinta al diseño del modelamiento matemático y de control tradicional, consiguiendo que una inteligencia artificial por medio del aprendizaje por refuerzo, sea la que se encargue de guiar autónomamente el mini drone a través de una trayectoria.

1.2 Objetivos del Trabajo

1.2.1 Objetivo General

Implementar un algoritmo autónomo de control de trayectorias en escenarios con obstáculos para mini drones utilizando técnicas de aprendizaje por refuerzo.

1.2.2 Objetivos Específicos

- Establecer un ambiente o entorno de acción el cual le permita a el agente interactuar, del cual serán recolectadas un conjunto de imágenes de diferentes trayectorias.
- Seleccionar una política de seguimiento basada en una red neuronal, la cual se encargue mediante observaciones del ambiente a aprender qué acciones son adecuadas, con la ayuda de una función de recompensa que incentive a el agente a cumplir con el objetivo.
- Entrenar iterativamente el agente para que, con la ayuda de algoritmos de aprendizaje por refuerzo, aprenda la una política de seguimiento a medida que interactúa con el entorno, de modo que, dado cualquier estado siempre tomará una acción adecuada.
- Realizar la validación de los resultados del mini dron en el software Matlab, a partir del modelo de simulación Quadcopter Project del entorno de Simulink, teniendo en cuenta diferentes trayectorias en diferentes entornos de acción.

1.3 Metodología

A continuación, se presentan las paso y métodos para la elaboración de proyecto, donde la mayoría de ellos serán elaborados en el software Matlab, teniendo así las siguientes etapas:

1. Creación de un ambiente virtual y obtención de datos en imágenes:

Inicialmente es necesario tener un conjunto de datos provenientes de la imagen a tiempo real de las trayectorias, para esto se debe diseñar un entorno con ciertas características como la segmentación del recorrido que debe seguir el mini dron, además esto ayudará a delimitar las zonas por las cuales el mini dron no tendrá que volar.

A partir de la toma de estas imágenes es esencial realizar varios métodos de procesamiento digital como la segmentación de áreas, para que una función propia de Matlab se encargue de identificar secciones de las trayectorias que se encuentran en las imágenes de la cámara, y así con este procedimiento se pueda identificar a partir de un sistema de sensores virtuales la posición en la que se encuentra el mini drone. Está claro que para esta aplicación en específico habrá que modificar el modelo en Simulink Minidrone Parrot Competition, con el fin de que sea posible la implementación de la primera etapa del proyecto y las posteriores.

2. Escoger una política de seguimiento basado en una red neuronal:

Se debe seleccionar la política de seguimiento óptima que relacione las observaciones que se tomen del entorno y las acciones que se van a efectuar en los controladores del mini dron, para la elaboración de este proyecto se representará esta función control en una red neuronal, la cual debe tener ciertos parámetros adecuados como el número de capas ocultas o el número de nodos en cada capa, además de elegir una estructura interna apropiada para la red neuronal.

Para que esta red neuronal aprenda de forma autónoma y optimice esta acción de control, se dará uso a él algoritmo de aprendizaje por refuerzo, método de optimización utilizado para encontrar la política óptima de seguimiento a lo largo del tiempo de aprendizaje.

3. Sistema de recompensas para el algoritmo de aprendizaje:

Para que el agente realice las acciones correctas para el guiado autónomo del mini dron, es apropiado crear una función de recompensa para que el algoritmo de aprendizaje

comprenda, cuando la política de seguimiento mejora, de modo que cada vez que el entorno cambie de estado este genere una nueva recompensa por la acción tomada y con esta nueva información el agente determina si la acción fue buena y debe repetirse o si fue mala y debe evitarse. Los parámetros a tener en cuenta en esta función de recompensas será la posición actual del mini drone y el tiempo de vuelo con que realice cada una de las trayectorias, además de tener una penalización si colisiona con algún obstáculo o no cumple su objetivo principal, el sistema de recompensas funciona a partir de un valor numérico, donde decrece si las acciones tomadas por el agente no son las correctas y aumenta si la acciones realizadas son las adecuadas para el seguimiento autónomo de trayectorias por parte del mini drone.

4. **Entrenamiento del agente, la política de seguimiento y el algoritmo de aprendizaje:**

En esta etapa el agente aprenderá de forma autónoma la mejor secuencia de acciones que generan un resultado óptimo. Con la implementación de la función de recompensa, se crea un ciclo de observación, acción y recompensa continúa hasta que se complete el aprendizaje. Además, el algoritmo de aprendizaje por refuerzo será el encargado de hallar y modificar los parámetros adecuados de la política de seguimiento del agente.

5. **Validación del Algoritmo de Seguimiento de Trayectorias:**

Con todo el conjunto que comprende el agente ya concluido se procederá a implementarlo en el entorno de simulación de Matlab, precisamente en el modelo de Simulink Quadcopter Project, para finalmente evaluar su comportamiento en distintos entornos en los que varían las trayectorias y sus parámetros de diseño.

2. Marco teórico

2.1 Cuadricóptero

El cuadricóptero también llamado drone o cuadrirrotor es una aeronave que vuela sin tripulación, este cuenta con cuatro rotores en cada una de sus aletas externas las cuales las utiliza para controlar su movimiento y estabilidad. También se les puede categorizar como UAV (Vehículo Aéreo no Tripulado) o como RPAS (Remotely Piloted Aircraft System). [18]

Figura 1 Mini Drone Parrot Mambo

Los rotores están compuestos por un motor y una hélice que se ubica en la extremidad de un brazo, estos son controlados por un módulo electrónico que controla el giro y velocidad de cada uno. Se tiene una estructura en forma de cruz de los brazos, donde también contiene la placa principal, la batería, los sensores y por lo general un transmisor inalámbrico para permitir su control remoto.

El funcionamiento del cuadricóptero puede llegar a ser algo complejo ya que, a partir de su estructura junto a componentes mecánicos y electrónicos, se debe tener una estabilidad que consiga equilibrar el empuje de todos los rotores, en otras palabras, que roten a la misma velocidad.

Se tiene que, por cada lado del dron, la pareja de rotores gira en sentido contrario para mantenerlo estable, si se requiere que el dron se eleve o descienda su altitud, la velocidad con la que giran los rotores tendrá que variar uniformemente y de la misma manera variara el torque aplicado al cuerpo según sea el caso. En cambio, para inclinar o virar la plataforma se debe aumentar y disminuir en igual medida la velocidad de los rotores que produzcan un torque a favor y en contra respectivamente. La combinación de estas acciones produce movimientos compuestos capaces de realizar desplazamientos en todo el espacio. [6]

2.2 Inteligencia Artificial

La inteligencia artificial o IA se puede decir que es la habilidad de los ordenadores para hacer actividades que normalmente requieren inteligencia humana. Específicamente la IA utiliza el potencial de los equipos de software para usar algoritmos, identificar y aprender datos, además utiliza lo aprendido en la toma de decisiones tal y como lo haría un ser humano. Sin embargo, a diferencia de las personas, los dispositivos basados en IA no necesitan descansar y pueden analizar grandes volúmenes de información a la vez. Asimismo, la proporción de errores es significativamente menor en las máquinas que realizan las mismas tareas que sus contrapartes humanas. La idea de que los ordenadores o los programas informáticos puedan tanto aprender como tomar decisiones es particularmente importante y algo sobre lo que se debe de ser consciente, ya que sus procesos están creciendo exponencialmente con el tiempo. Debido a estas dos capacidades, los sistemas de inteligencia artificial pueden realizar ahora muchas de las tareas que antes estaban reservadas sólo a los humanos.

La IA puede ser categorizado en cualquier número de maneras, pero aquí hay dos ejemplos:

- El primero clasifica los sistemas de inteligencia artificial como IA débil o IA fuerte. La IA débil, también conocida como IA estrecha, es un sistema de IA que está diseñado y entrenado para una tarea en particular. Los asistentes personales virtuales, como Siri de Apple, son una forma débil de IA.
- La IA fuerte, también conocida como inteligencia artificial general, es un sistema de IA con habilidades cognitivas humanas generalizadas, de modo que cuando se le presenta una tarea desconocida, tiene suficiente inteligencia para encontrar una

solución. La prueba de Turing, desarrollada por el matemático Alan Turing en 1950, es un método utilizado para determinar si una computadora puede realmente pensar como un humano, aunque el método es polémico.

Uno de los grandes beneficios de la inteligencia artificial es que se puede aplicar en una gran mayoría de situaciones, las siguientes son algunas aplicaciones que se han implementado y están en crecimiento actualmente:

- Detección y clasificación de objetos: Puede verse en la industria de vehículos autónomos, aunque también tiene potencial para muchos otros campos.
- Reconocimiento de imágenes estáticas, clasificación y etiquetado: Estas herramientas son útiles para una amplia gama de industrias.
- Procesamiento eficiente y escalable de datos de pacientes: Esto ayudará a que la atención médica sea más efectiva y eficiente.
- Mantenimiento predictivo: otra herramienta ampliamente aplicable en diferentes sectores industriales
- Protección contra amenazas de seguridad cibernética: Es una herramienta importante para los bancos y los sistemas que envían y reciben pagos en línea.

En ciertos casos de aplicación la IA permitirá que los robots y máquinas sean capaces de realizar tareas que los humanos consideran difíciles, aburridas o peligrosas, lo que repercutirá a su vez en que el ser humano pueda realizar aquello que antes creía imposible. [17]

2.3 Aprendizaje Automático

El aprendizaje automático (en inglés, machine learning) es uno de los enfoques principales de la inteligencia artificial. En pocas palabras, se trata un aspecto de la informática en el que los ordenadores o las máquinas tienen la capacidad de aprender sin estar programados para ello. El aprendizaje automático usa algoritmos para aprender de los patrones de datos. Por ejemplo, los filtros de spam de correo electrónico utilizan este tipo de aprendizaje con el fin de detectar qué mensajes son correos basura y separarlos de aquellos que no lo son. Éste es un sencillo ejemplo de cómo los algoritmos pueden usarse para aprender patrones y utilizar el conocimiento adquirido para tomar decisiones.

Existen tres subconjuntos del aprendizaje automático, los cuales son los que se explican a continuación:

1) Aprendizaje Supervisado: Los algoritmos usan datos que ya han sido etiquetados u organizados previamente para indicar cómo tendría que ser categorizada la nueva información. Con este método, se requiere la intervención humana para proporcionar retroalimentación.

2) Aprendizaje no Supervisado: Los algoritmos no usan ningún dato etiquetado u organizado previamente para indicar cómo tendría que ser categorizada la nueva información, sino que tienen que encontrar la manera de clasificarlas ellos mismos. Por tanto, este método no requiere la intervención humana.

3) Aprendizaje por Refuerzo: Los algoritmos aprenden de la experiencia. En otras palabras, se tiene que proporcionar "un refuerzo positivo" cada vez que aciertan.[17]

2.4 Aprendizaje por Refuerzo

El aprendizaje por refuerzo (RL) se basa en aprender que hacer o cómo asignar acciones a situaciones determinadas que se presentan en el entorno en el que se encuentra el aprendiz o técnicamente llamado agente, dicho agente tiene la habilidad de obtener información de su alrededor con la implementación de sensores y cámaras las cuales le ayudan a saber en qué estado se encuentra para así elegir su siguiente acción, teniendo como resultado una señal de recompensa numérica la cual indica qué tan correcta fue la acción realizada. Esta técnica de aprendizaje no le dice al agente qué acciones tomar, como en otras formas de aprendizaje automático, sino que el mismo debe descubrir qué acciones producen la mayor recompensa numérica probándolas, teniendo un modelo de prueba y error característica propia del aprendizaje por refuerzo, así se da paso a un ciclo de observación, acción y recompensa el cual se ejecuta hasta que se complete el aprendizaje [16].

En el aprendizaje reforzado hay que tener en cuenta dos aspectos importantes: la exploración y la explotación. Exploración se refiere a la elección de acciones de manera aleatoria. La explotación en cambio, se refiere a tomar decisiones (elegir acciones) en base a cuán valiosa es realizar una acción a partir de un estado dado. Dependiendo de cómo se requiera que se desarrolle el aprendizaje, se pueden variar los niveles de exploración y

explotación. Por ejemplo, se puede establecer que el agente elija el 30% del tiempo acciones de manera aleatoria para que explore por sí solo el ambiente, y que el 70% del tiempo restante elija las acciones más valiosas para cada estado en que se encuentre. La razón por la cual no siempre se elige una explotación de las acciones es porque el agente comienza a aprender de cero. Por lo que en un comienzo todas las acciones en el estado inicial tienen un valor nulo. Y más aún, el número de acciones disponibles para cada estado puede variar de estado en estado, por lo que el ambiente en general no es conocido de antemano. Es solo a través de la experiencia que las acciones empiezan a adquirir un valor, por lo que la exploración es crucial y necesaria.

En todo problema de aprendizaje por refuerzo hay: un agente, un ambiente definido por estados, acciones que el agente lleva a cabo (y que llevan al agente de un estado a otro), y recompensas o penalizaciones que el agente obtiene en camino a lograr su objetivo. En la Figura 2 se pueden observar las partes y el modelo de funcionamiento del aprendizaje por refuerzo.

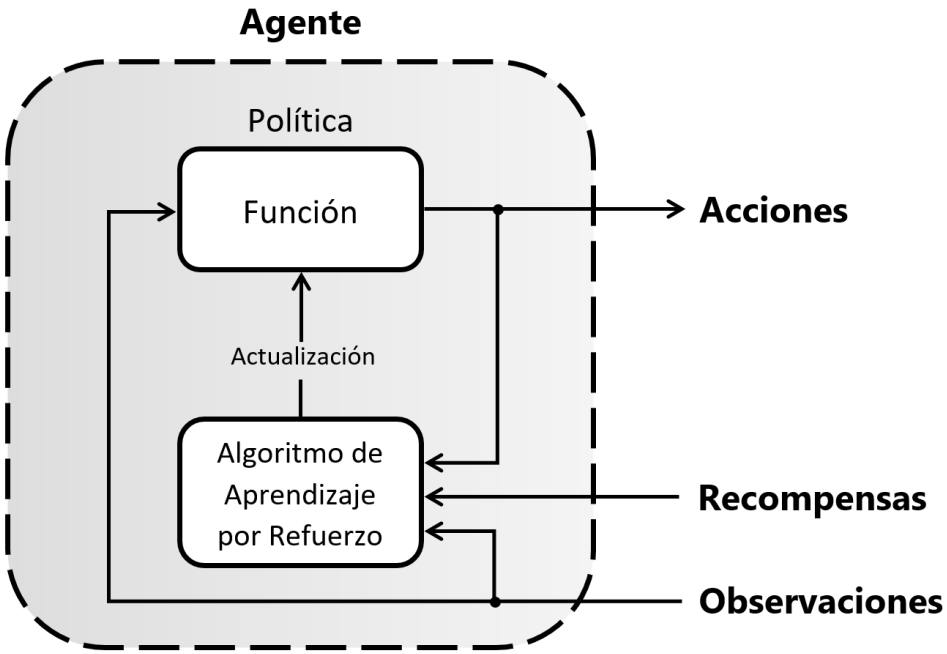


Figura 2 Esquema Aprendizaje por Refuerzo

2.4.1 Agente

Es una pieza de software la cual se encarga de explorar, interactuar y aprender del entorno o medio ambiente. Este puede observar el estado actual del medio en el que se

encuentra, a partir de este estado observado decide que acción tomar, así el entorno cambia de estado y produce una recompensa por la acción. Usando esta información el agente puede determinar si esa acción fue buena y debe repetirse, o si fue mala y debe evitarse.

2.4.2 Entorno o Ambiente

Es el lugar donde el agente aprende, allí este es capaz de enviar acciones y de recibir las observaciones y recompensas resultantes de las operaciones efectuadas, se puede decir que el ambiente es todo lo que existe menos el agente, incluyendo la dinámica del sistema.

Para configurar el entorno se debe elegir que debe existir dentro de él y si este será una simulación o una configuración física real. Un entorno simulado tendrá características en las cuales la de velocidad de aprendizaje será mayor ya que se puede hacer uso de la paralelización, también en este caso será más fácil modelar situaciones que serían difíciles de poner en práctica en la vida real y no tiene riesgos de dañar el hardware que se esté utilizando. Pero si se requiere de más fidelidad a la hora de realizar el aprendizaje, un entorno real es muy preciso, ya que nada representa el entorno de forma más completa que un entorno real, además de que si se cuenta con el no será necesario dedicar tiempo en crear y validar un modelo.

2.4.3 Política de Seguimiento

Es la función que asigna observaciones de los estados percibidos del entorno, a las acciones que se pueden tomar cuando se está en estos estados, en otras palabras, define el comportamiento del agente en un momento dado, esto quiere decir que el agente se encuentra compuesto por esta política de seguimiento, además del algoritmo de aprendizaje por refuerzo, el cual es el método de optimización utilizado para encontrar la política adecuada al problema que se esté tratando. Según sea el objeto de aplicación esta política puede ser una función simple como una tabla de búsqueda, pero en aplicaciones más complejas puede llegar a ser un cálculo extenso.

2.4.4 Función de Recompensa

Es un valor numérico el cual define el objetivo del aprendizaje, este valor es recibido constantemente por el agente, el cual a partir de esta recompensa le permite comprender al algoritmo de aprendizaje cuando la toma de decisiones mejora y en última instancia, cuando se llegó al resultado que se está buscando. El objetivo principal del agente es maximizar la recompensa total que recibe a largo plazo, además esta recompensa será el parámetro principal para que el algoritmo de aprendizaje decida modificar la política de seguimiento, teniendo así que, si la acción seleccionada por la política da como resultado una recompensa baja, entonces la política puede cambiarse para seleccionar alguna otra acción en esa situación en particular en un futuro.

2.5 Redes Neuronales:

Las redes neuronales son un grupo de nodos o neuronas artificiales, que están conectados de una manera que les permite relacionar de una forma correcta una entrada y una salida determinada, siempre y cuando la combinación de nodos y conexiones de la red neuronal sea la correcta. Donde el proceso de aprendizaje consistirá en ajustar sistemáticamente los parámetros de la red para encontrar la relación óptima de entrada/salida.

2.5.1 Estructura de la Red Neuronal

Existen diversos tipos de redes neuronales cada una con una estructura interna distinta, teniendo en cuenta las características del proyecto se eligió una red neuronal completamente conectada, ya que este tipo de estructura ha funcionado de una manera adecuada en tareas de control autónomo.

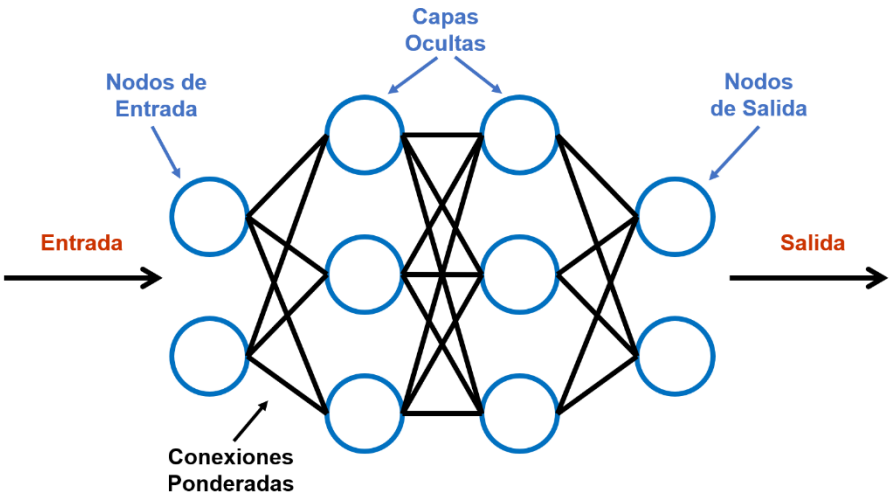


Figura 3 Red Neuronal Completamente Conectada

En la Figura 3 se puede observar la estructura de la red neuronal elegida, a la izquierda están los nodos de entrada y a la derecha están los nodos de salida, la cantidad de nodos entrada/salida dependerá de la aplicación a la cual vaya a ser usada la red. En el medio hay una serie de columnas de nodos llamadas capas ocultas. Como modo de ejemplo esta red neuronal completamente conectada tiene dos entradas, dos salidas y dos capas ocultas de tres nodos cada una, con esta estructura hay una conexión ponderada desde cada nodo de entrada a cada nodo en la siguiente capa, y luego desde esos nodos a la capa posterior y nuevamente hasta los nodos de la salida o la capa de salida.

Una sola capa completamente conectada consta de varias neuronas y el número de estas es un parámetro que no es exacto y según vaya avanzando el entrenamiento se tendrá que ir modificando este valor para encontrar un comportamiento adecuado de la red neuronal. Cada neurona es una operación matemática, la cual multiplicará la entrada (x_{in}) por una constante llamada peso (*w*), para posteriormente sumarle otra constante llamada sesgo (*b*), el resultado de esta operación (x_{out}) será la entrada de la siguiente neurona de la capa posterior.[19]

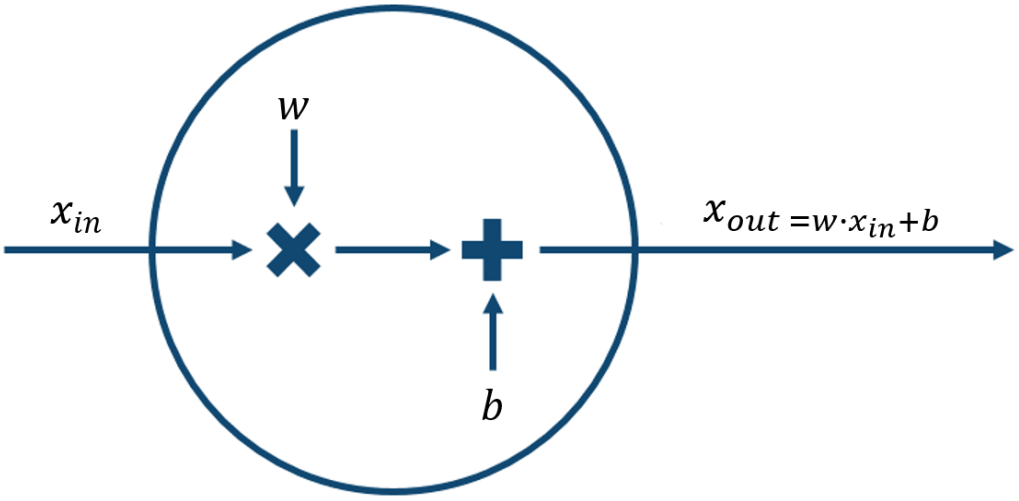


Figura 4 Operación Matemática de una Neurona

Para las redes de actores o críticos, normalmente las capas internas u ocultas son capas completamente conectadas, pero también se utilizan capas de activación, las cuales aplican una función para introducir la no linealidad. Entre las funciones de activación más comunes se tienen:

- La unidad lineal rectificada (o "ReLU"): Deja las entradas positivas sin cambios y pone las entradas negativas a cero.

- La tangente hiperbólica (tanh): Genera una curva suave de manera que cualquier entrada entre el infinito negativo y positivo se reduce a el rango [-1 1].

El modelo de la estructura de la red neuronal la cual va a representar a el actor y el crítico de un agente, será de la siguiente manera:

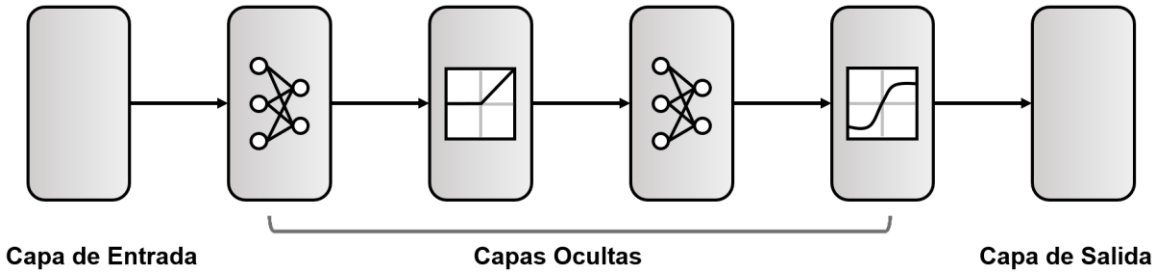


Figura 5 Capas de la Red Neuronal

2.5.2 Estructura Actor/Crítico

Hasta el momento se ha mencionado que el agente esté compuesto por un actor y un crítico y estos dos serán representados por una o varias redes neuronales. Por una parte, está el actor llamado así ya que le dice directamente al agente que acciones tomar y por otra parte se encuentra el crítico que es una segunda red neuronal que intenta estimar un valor según el estado actual del entorno y la acción que realizó el actor, como se puede ver en la Figura 6. En otras palabras, el crítico trata de estimar el valor de la recompensa futura, según las observaciones y las acciones tomadas por el actor. Este método actor/crítico está diseñado para funcionar en espacios de acción continua, porque el crítico sólo necesita observar la acción individual que realizó el actor y no necesita tratar de encontrar la mejor acción evaluando todas las posibles.

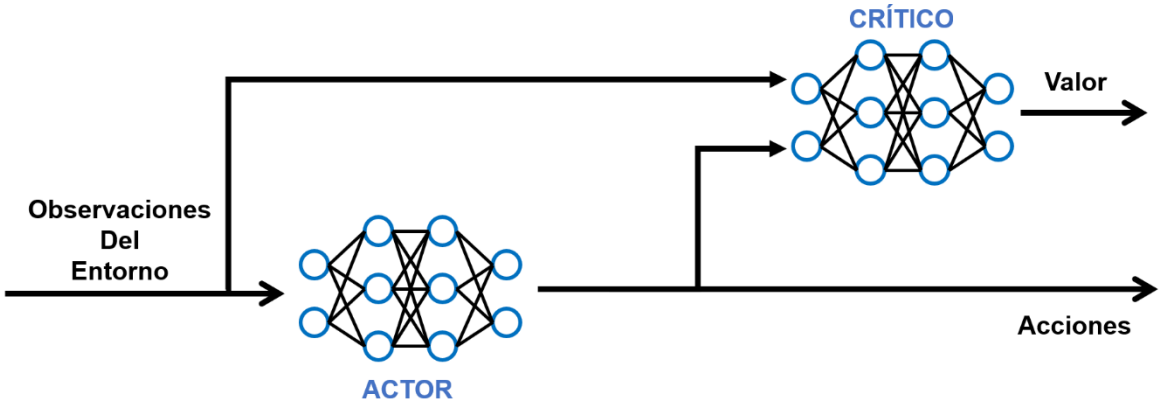


Figura 6 Estructura Actor / Crítico

El valor estimado por el crítico junto con el valor de la recompensa en el periodo del entrenamiento, serán tenidos en cuenta por el algoritmo de aprendizaje por refuerzo el cual se encargará de modificar los parámetros de las redes neuronales antes mencionadas. Específicamente como enseña la Figura 7, el crítico usa la recompensa del entorno para determinar la precisión de su predicción, donde se calculará un error el cual va a ser la diferencia entre el valor estimado del estado anterior del entorno y el valor de la recompensa del mismo. El error le da a el crítico una idea de si las acciones resultaron

mejor o peor de lo esperado por el mismo, además el crítico usa este error para actualizarse a sí mismo, para que tenga una mejor predicción la próxima vez que esté en un estado específico del entorno. El actor también se actualiza con la respuesta del crítico para que pueda ajustar sus probabilidades de volver a tomar esa acción en el futuro.

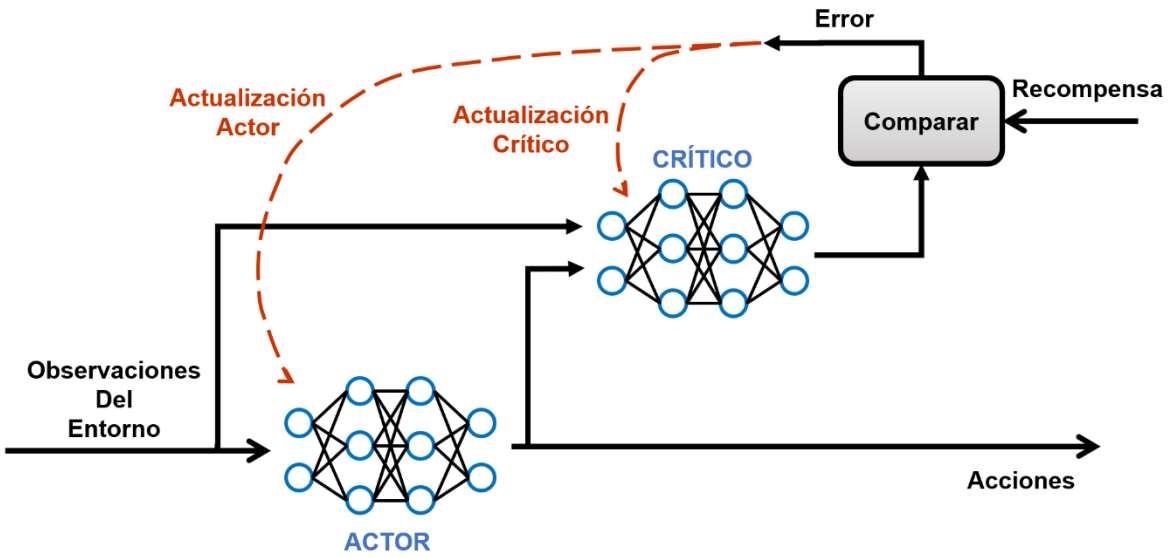


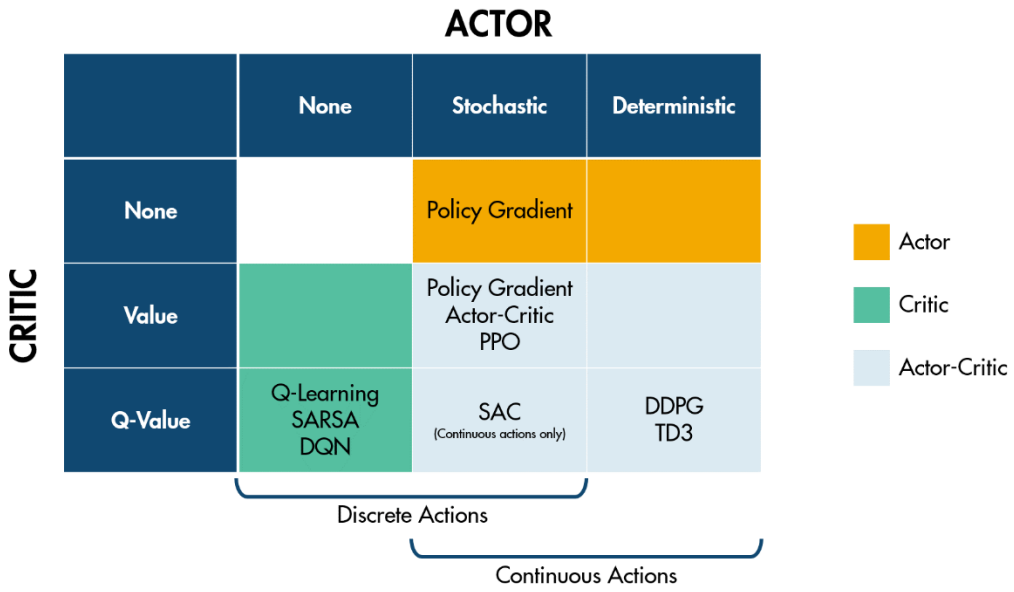
Figura 7 Actualización del Actor y el Crítico

El actor y el crítico son redes neuronales que intentan aprender el comportamiento óptimo según el espacio de acción en el que se encuentren. El actor está aprendiendo las acciones correctas a tomar utilizando la retroalimentación del crítico para saber qué acciones son buenas y malas, y el crítico está aprendiendo por decirlo de algún modo una función de valor de las recompensas recibidas, para así poder “criticar” adecuadamente la acción que realiza el actor. [19]

2.5.3 Arquitectura Agentes DDPG

Existen diversas estructuras de redes neuronales actor/crítico, donde se pueden encontrar diversas configuraciones dependiendo del tipo de actor y crítico que se vaya a utilizar. En la Tabla I se puede observar que existen dos tipos de agentes (Estocásticos y Deterministas), y a su vez también existen dos tipos de críticos (Valor y Valores-Q). A partir de estos tipos se pueden crear diferentes configuraciones de agentes, se puede observar que según el espacio de acción (Discreto o Continuo) se tiene que usar una configuración u otra.

Tabla I Configuraciones Actor / Crítico, [19]



En el caso del presente proyecto se elegirá una configuración de agente DDPG (Deep Deterministic Policy Gradient), la cual es una configuración Actor Determinista y Crítico de Valores-Q, ya que como se ha mencionado anteriormente el control autónomo del mini drone es un espacio de acción continuo.

Para conocer más a profundidad este tipo de agente se debe saber que los actores deterministas proporcionan un mapeo directo de las observaciones a los valores de acción. Aunque el actor es determinista, el agente generalmente agrega ruido a las acciones para promover la exploración, o sea que cuando todo el conjunto de la inteligencia artificial este en el periodo de entrenamiento, las acciones que sean realizadas tendrán un poco de aleatoriedad para que así el agente puede conocer cuál es su espacio de acción. Este parámetro de exploración va de la mano con el de explotación, los cuales son dos características a tener en cuenta cuando se realice el entrenamiento, más adelante en el apartado del entrenamiento se profundizará en el tema.

El diagrama de ilustrado en la Figura 8 enseña cómo el agente DDPG funciona, primero el agente observa el estado del medio ambiente que pasara a el actor, además estas observaciones también pasan por el crítico junto a la acción determinada por el actor. A partir de estas dos entradas el crítico estima un valor del estado actual, hay que tener en cuenta que hasta el momento la acción no se ha efectuado en el entorno, pero el crítico ya hizo una estimación de lo que espera a futuro según el estado y la acción determinada. Posteriormente se genera la acción en el entorno haciendo que este cambie, a la vez que se genera una recompensa, y ahora el agente recibirá las actualizaciones del nuevo estado y también la recompensa generada. El actor usa el nuevo estado para determinar una nueva acción. Esta acción y el nuevo estado se pasan al crítico, que estima un nuevo valor. Es decir, el crítico estima cuánta recompensa recibirá el agente en el futuro por esta situación, la diferencia de esto con la recompensa observada da una estimación

actualizada del error que necesita el algoritmo de entrenamiento como se había mencionado anteriormente.

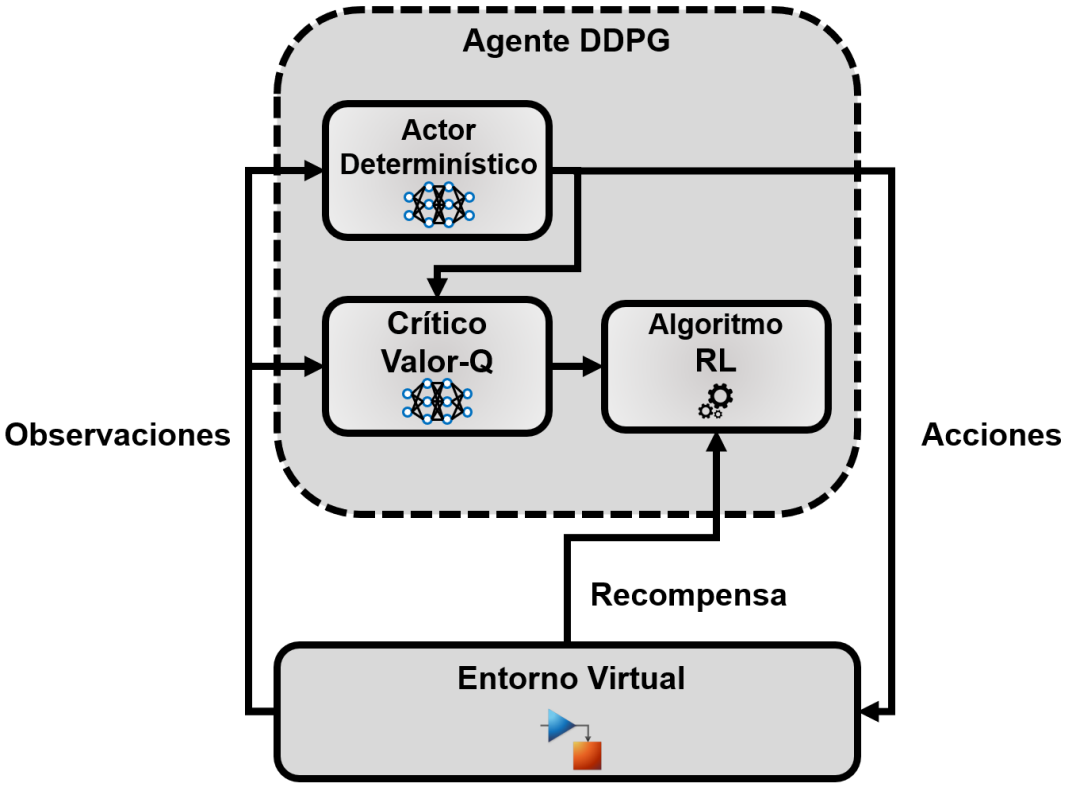


Figura 8 Diagrama Agente DDPG

El algoritmo de entrenamiento utiliza la estimación original y la estimación actualizada para actualizar tanto al actor como al crítico. El crítico se actualiza para producir estimaciones más cercanas a las logradas con las recompensas reales. El actor se actualiza para producir acciones que conducirán a estados con valores más altos de recompensa.

Recapitulando se tendrían los modelos de las redes neuronales tanto para el actor como el crítico en la Figura 9, los cuales muestran cómo debe ser la conexión entre las distintas capas, como las de entrada, las ocultas y las de salida. En el caso del actor determinista utilizado en acciones continuas, la capa final de la red neuronal debe tener una neurona para cada acción, en ocasiones se limitan las acciones del agente para que el periodo de entrenamiento sea más corto y el actor no utilice más tiempo probando acciones que no van a ser óptimas durante el entrenamiento. En este caso son útiles las capas de activación como la de tangente hiperbólica (tanh) o una capa de escala para mantener los valores de salida dentro de unos rangos dados.

Para agentes que utilicen críticos de Valores-Q como es el caso, una vez dadas las observaciones y acciones, el crítico devuelve el valor de realizar una acción determinada en un estado específico, este valor numérico es guardado en una tabla, la cual registra todos los posibles estados y todas las posibles acciones que se pueden realizar en cada estado del entorno. A partir de estos datos se crea la Q-Table o la Tabla de Calidad, en la

que se guardan los Q-Values o Valores de Calidad, los cuales determinan que tan óptimo es realizar una acción dependiendo de un estado determinado. Por lo general cuando el entorno cambia y se crea un nuevo estado el agente elige la acción con el Q-Value más alto de la Q-Table, pero de vez en cuando tomará una acción aleatoria en su lugar.

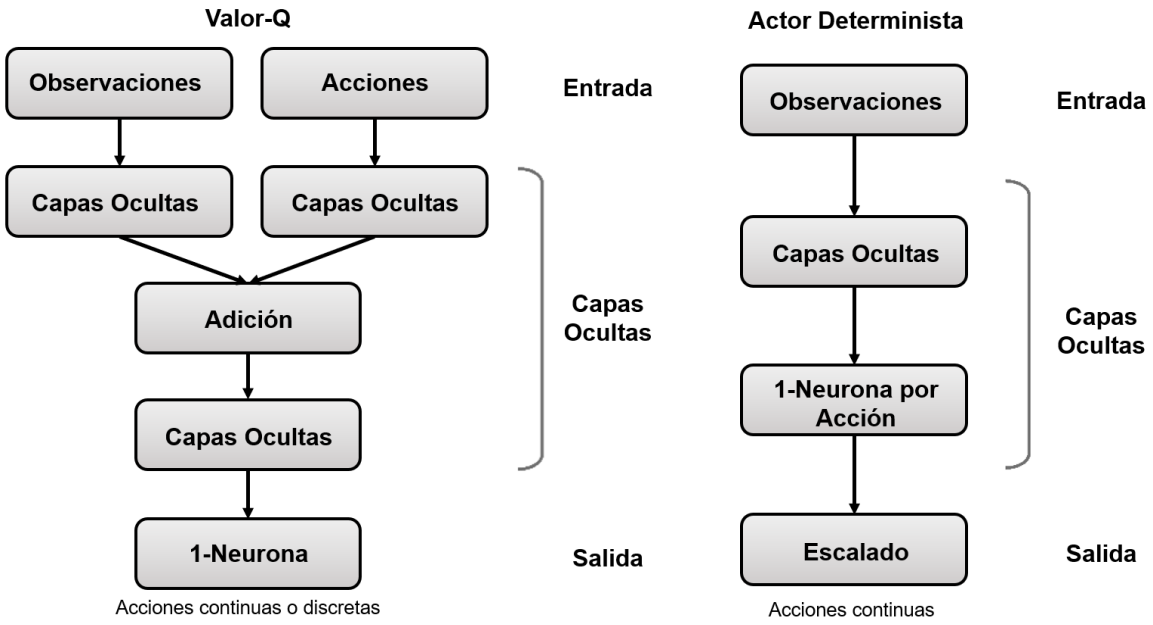


Figura 9 Arquitectura Redes Neuronales

En la Figura 9 se observa una red para representar un crítico Q-Value el cual tiene dos secuencias independientes de capas, una para las observaciones y otra para las acciones, que posteriormente se fusionan. La última capa es una capa completamente conectada con una sola neurona que representa el Q-Value. [19]

3. Algoritmo de Control de Trayectorias

En el presente capítulo se describe cada una de las etapas necesarias para diseñar y elaborar un algoritmo autónomo de control de trayectorias para mini drones utilizando el enfoque de aprendizaje por refuerzo.

Para la elaboración este algoritmo como se mencionó en la metodología se deben seguir una serie de pasos y así poder concretar una inteligencia artificial adecuada, comenzando por el establecimiento de un entorno virtual en el que cada una de las partes implicadas del modelo (agente, función de recompensa, algoritmo de aprendizaje por refuerzo, etc.) estarán realizando cada una de sus funciones específicas, además de que todo el modelo de simulación del mini drone va estar allí implementado. Así con este entorno virtual se logrará visualizar el comportamiento exacto del mini drone, a la vez que se podrá presenciar la evolución de cada uno de los elementos que componen el aprendizaje por refuerzo en el periodo de entrenamiento.

Posteriormente de haber concretado por decirlo de algún modo el espacio de trabajo virtual, se debe diseñar la pieza de software encargada de controlar y guiar al mini drone, que en otras palabras sería el agente. Es necesario conocer todas las arquitecturas de redes neuronales y los diferentes tipos de agentes que se pueden utilizar en aplicaciones de control autónomo, para poder elegir de manera adecuada cada uno de los parámetros necesarios para la creación de la política de seguimiento que compone el agente.

Una vez que se cuente con el agente terminado y listo para poder interactuar en el entorno virtual, se debe utilizar una función de recompensa que ayudará a el agente a comprender qué acciones son correctas según las observaciones del estado en el que se encuentre. Para esto hay que tener en cuenta que las funciones de recompensa son elaboradas a partir del tipo de trayectoria que se tiene que realizar y que para una sola trayectoria puede haber diferentes funciones de recompensa que sean capaces de cumplir con el objetivo, además de que la función de recompensa es el resultado de una serie de modificaciones en un proceso de prueba y error, el cual se hace presente en la etapa de final de entrenamiento.

El algoritmo de control autónomo estará compuesto por este conjunto de etapas y pasos, el cual posteriormente irá mejorando por medio de las sesiones de entrenamiento junto con el algoritmo de aprendizaje por refuerzo. Cada una de las etapas antes mencionadas serán tratadas a profundidad en las siguientes secciones, donde se describe la serie de pasos necesarios para realizar el guiado autónomo de dos trayectorias propuestas.

3.1 Entorno Virtual de Acción

Para crear el entorno o espacio de acción como se ha mencionado anteriormente se trabajará en un entorno de simulación, en el programa Matlab y Simulink.

3.1.1 Parrot Minidrone Toolbox

Este Toolbox permite diseñar y construir algoritmos de control de vuelo para mini drones Parrot. Donde se dispone con una gran variedad de sensores integrados, como lo son el acelerómetro, giroscopio, sensores ultrasónicos y de presión de aire, además de las imágenes obtenidas de la cámara orientada hacia abajo con la que cuenta el mini drone.

Inicialmente se debe contar el *"Simulink Support Package for Parrot Minidrones"* el cual es un paquete de soporte que contiene todo lo necesario para establecer el modelo del mini drone junto a su entorno de acción en Matlab, además para fines del proyecto se tendrán que utilizar Toolboxes de aprendizaje por refuerzo, entre otros. Una vez instalados todos los paquetes necesarios se procede a abrir el modelo con la siguiente función: *"parrotMinidroneCompetitionStart"*, cuando se ejecute se tendrá el modelo de la figura 10 en el entorno de Simulink.

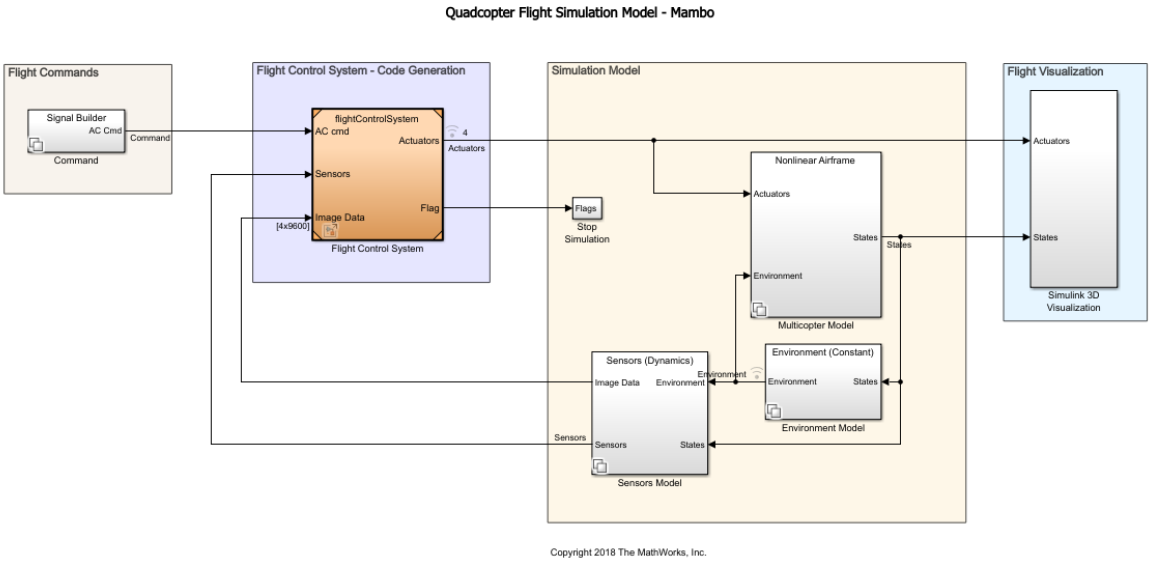


Figura 10 Modelo Simulink parrotMinidroneCompetition

Allí se evidencian diferentes bloques o subsistemas los cuales se encargan de controlar cada una de las partes del mini drone, como el bloque *"Sensor Model"* el cual se encarga del procesamiento de los diferentes sensores ya mencionados anteriormente o el bloque *"Simulink 3D Visualization"* el cual proporciona un entorno de simulación, en donde se encontrará el mini drone de una manera virtual como se apreciará en secciones posteriores.

3.1.2 Procesamiento de Imágenes de la Cámara

Además de todos los bloques mencionados anteriormente también se tiene el subsistema *"Flight Control System"* allí se encontrarán dos bloques que lo componen: El primero se encarga de procesar todas las imágenes de la pista que recibe el mini dron provenientes de la cámara que tiene incorporada, estas imágenes serán útiles ya que con un procesamiento de imágenes se podrá conocer si el dron está siguiendo la trayectoria de una manera adecuada. Como se puede ver en la Figura 11 se encuentra el "*Image Processing System*" el cual se encargará de esta tarea, inicialmente los datos de salida de la cámara que en términos generales son las imágenes, están en el espacio de color YUV, pero con un bloque conversor se pasaran las imágenes a la composición de color RGB.

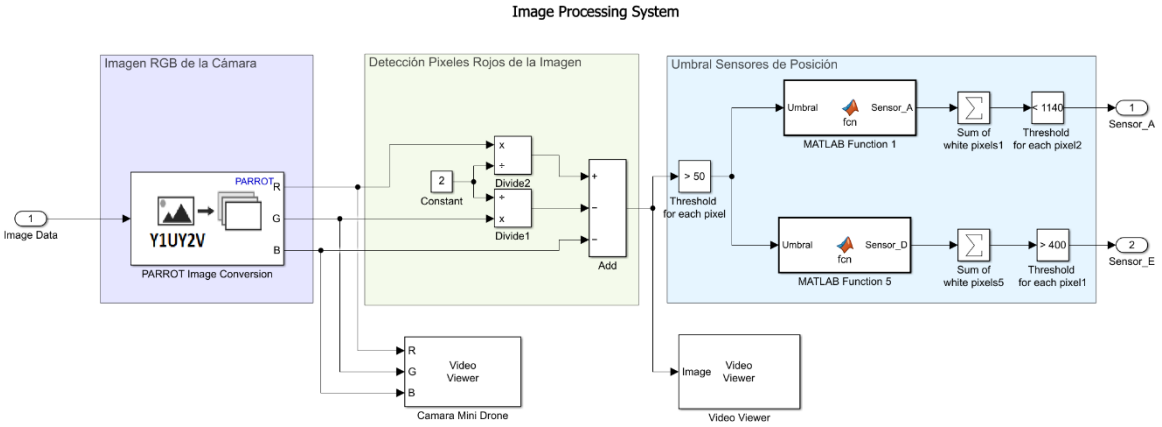


Figura 11 Image Processing System

Con esta conversión será más sencillo encontrar los píxeles de color rojo que se encuentran en la pista, los cuales son los que indican la trayectoria que debe seguir el mini drone (Figura 5), con la herramienta Video Viewer se puede ver lo que esta observado la cámara y como esta tarea de procesamiento de imágenes, se encarga de detectar solo los píxeles de la imagen que son necesarios, de la siguiente manera:

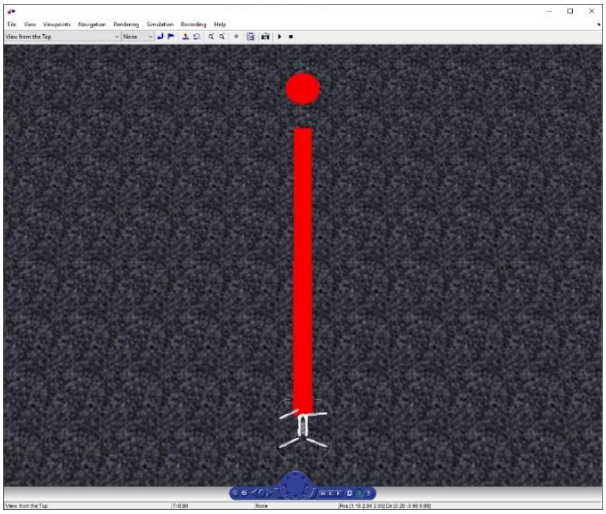


Figura 12 Entorno de Simulación

En la Figura 13 están las dos visualizaciones proveen los Video Viewer's del bloque Image Processing System de la Figura 11, en la izquierda se tienen las imágenes captadas por la cámara del mini dron y en la derecha se observa como el proceso de detección seleccionó solo los píxeles rojos de la trayectoria, transformándolos a un conjunto de píxeles blancos en una matriz de píxeles de 120x160.

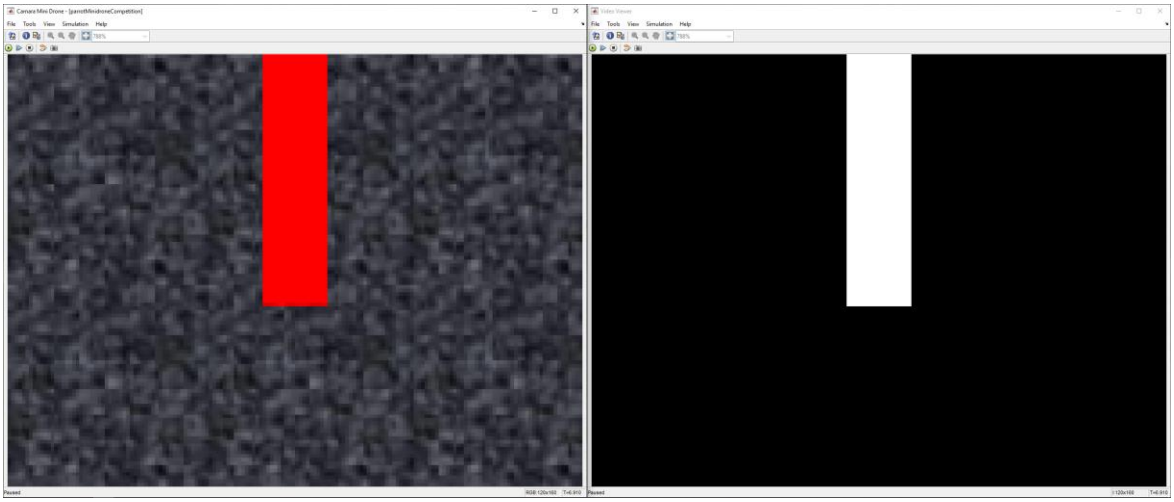


Figura 13 Detección de Píxeles Rojos

Siguiendo con el Image Processing System va a ser necesario ubicar dos sensores virtuales que estén detectando la cantidad de píxeles en una zona determinada de la imagen de la cámara, para esto se cuenta con la Function 1 y Function 5 las cuales se encargan de recortar la imagen de la cámara en un área definida como se muestra a continuación:

```
function Sensor_A = fcn(Umbral)
Umbral=Umbral(1:120,61:99);
Sensor_A = Umbral;
```

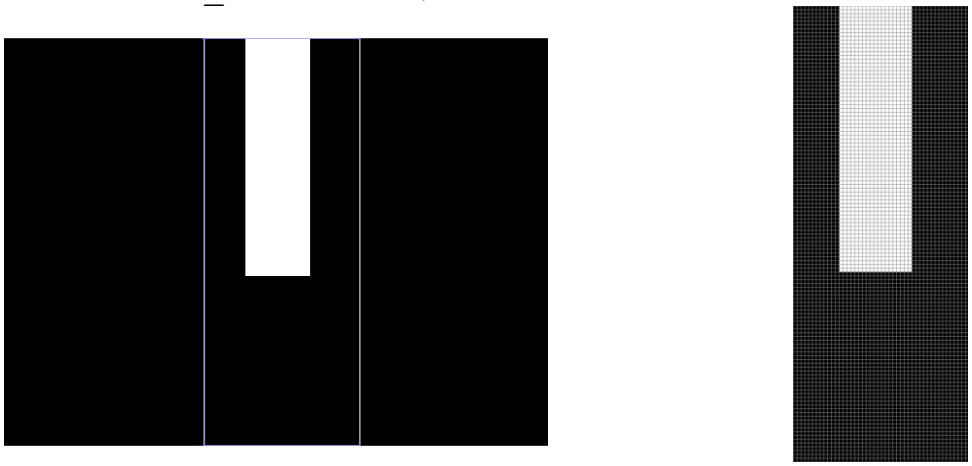


Figura 14 Selección Área Sensor Virtual A

Con este proceso serán seleccionados solo los píxeles de la cámara que se encuentran en la sección central de la imagen, en la Figura 14 se observa que sólo los píxeles que están dentro del recuadro son los que serán seleccionados, para determinar esta área central se utilizó la Function 1 que entregará la sección recortada de 120x38 píxeles ubicada en la mitad de la cámara, que es lo que se observa en la imagen de la derecha de la Figura 14. Posteriormente se procede a realizar la suma de los píxeles blancos de la zona delimitada y con unas funciones que comparan el resultado de la suma con un valor estipulado, prenden o apagan los sensores virtuales según sea el caso. Estos sensores virtuales elaborados a partir del proceso antes mencionado, serán de ayuda para establecer una función de recompensa que será utilizada en el periodo de entrenamiento del agente, allí se debe tener en cuenta que según la ubicación de la zona seleccionada de la cámara para cada uno de los diferentes sensores, se podrá saber si el mini drone está siguiendo la línea roja de la trayectoria o por el contrario si la suma de píxeles blancos es menor a un valor establecido en el Sensor A significara que el mini dron salió de la trayectoria, estos serán algunos de los diferentes estados que se tendrán que evaluar en la sección de entrenamiento y recompensa. A continuación, se presenta un diagrama de flujo explicando el todo el sistema de procesamiento de imágenes:

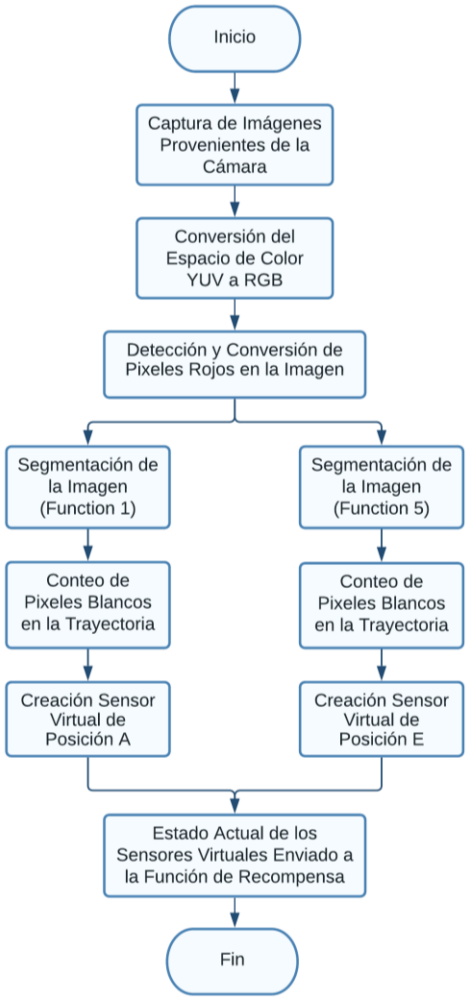


Figura 15 Diagrama de Flujo Procesamiento de Imágenes

3.1.3 Sistema de Control

Como se mencionó anteriormente el subsistema *"Flight Control System"* tiene dos bloques que lo componen, el primero (Image Processing System) ya fue descrito en la sección anterior, el segundo bloque es el Control System que en su interior cuenta con diversos sistemas, como el State Estimator que entregara las variables de estado como la posición X,Y,Z del mini dron cuando se esté ejecutando la simulación, esto será útil para elaborar la función de recompensa además de que estas variables serán los estados observados por el agente, como se puede ver en el esquema de Aprendizaje por Refuerzo de la Figura 2 y el otro sistema de control importante que se va estar utilizando es el Path Planning de la Figura 16, allí se encuentra presente en primer lugar las salidas del estimador de estados, también se encuentra el sistema de aprendizaje por refuerzo compuesto por los bloques "controller" que es donde se encuentra la inteligencia artificial o el agente encargado de realizar las acciones y también se encuentra el bloque Reward Function, el cual es una función que evaluará las observaciones y los sensores virtuales antes mencionados para que así calcule un valor de recompensa, el cual será entregado al agente. Este proceso de recompensa será desarrollado más ampliamente en secciones posteriores.

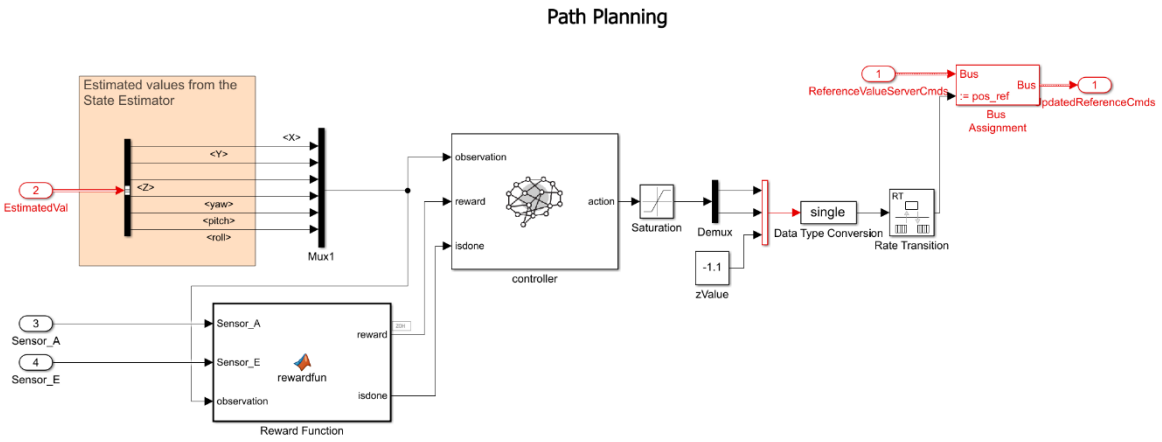


Figura 16 Path Planning

El bloque "controller" que es donde se encuentra el agente será el encargado de controlar la posición del mini dron en los ejes X, Y sin embargo la posición z que es la altura del mini dron será constante durante toda la simulación, por esto se puede observar que en la Figura 16 el valor de Z es una constante establecida a una altura de *1.1 m*. Estos valores de posición X, Y, Z sirven como referencia para la salida del sistema de Path Planning, que serán ingresados a los sistemas que se encargan de calcular las dinámicas de los motores más específicamente las fuerzas y torques que debe generar cada motor del mini dron para que este se dirija a la posición de referencia establecida por el sistema de control.

El método de control por medio del aprendizaje por refuerzo se puede comparar con el modelo tradicional de control, como muestra el siguiente diagrama de bloques:

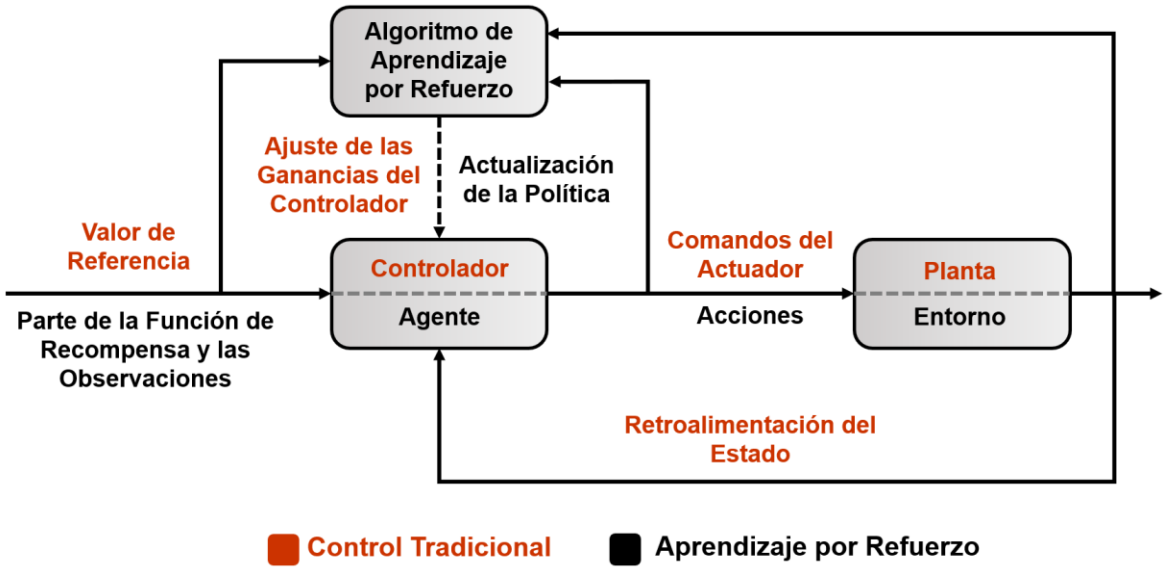


Figura 17 Comparativa Control Tradicional y Aprendizaje por Refuerzo

El objetivo del aprendizaje por refuerzo es similar al problema de control, solo que es un enfoque diferente y utiliza términos distintos para representar los mismos conceptos. Con ambos métodos se está tratando de averiguar cómo diseñar la política del agente (o el controlador) que analiza el estado observado del entorno (o la planta), para así tener las mejores acciones (o los comandos del actuador). La señal de retroalimentación del estado son las observaciones del entorno y la señal de referencia está integrada tanto por la función de recompensa como las observaciones del entorno.

Teniendo así que cada uno de los bloques del sistema de control Path Planning se pueden relacionar con el modelo típico de control como se enseña a continuación:

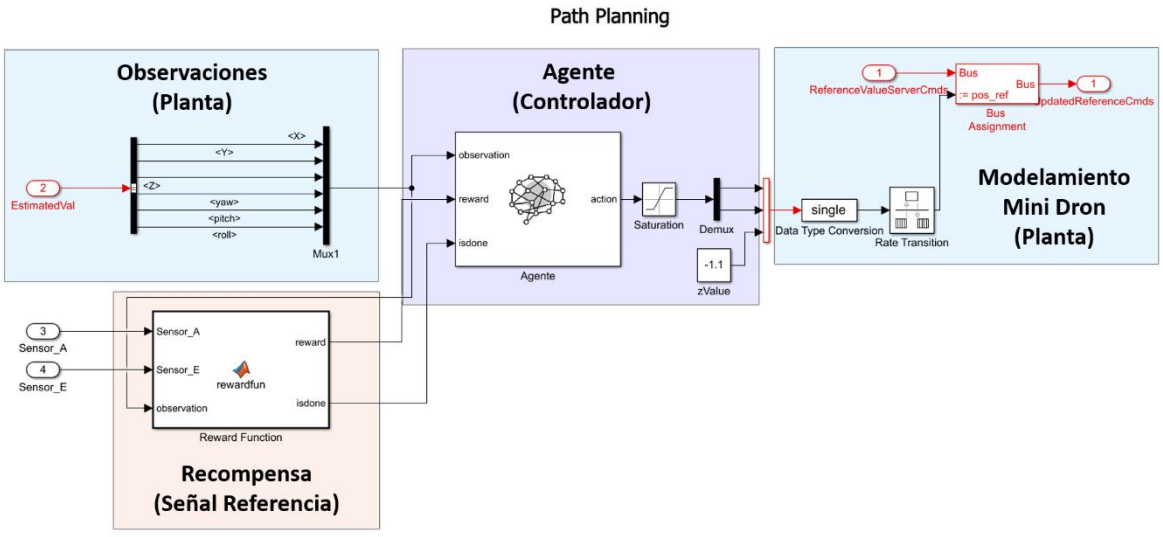


Figura 18 Modelo de Control Path Planning

3.1.4 Modificación de la Trayectoria

Una función que es de gran utilidad para establecer trayectorias y configurar las coordenadas de inicio del mini dron en el entorno virtual es la función *Track Builder,* la cual permite crear las trayectorias de la pista con un conjunto de datos de los dos ejes que en este caso serian: North y East (en metros), como se muestra en la Figura 19.

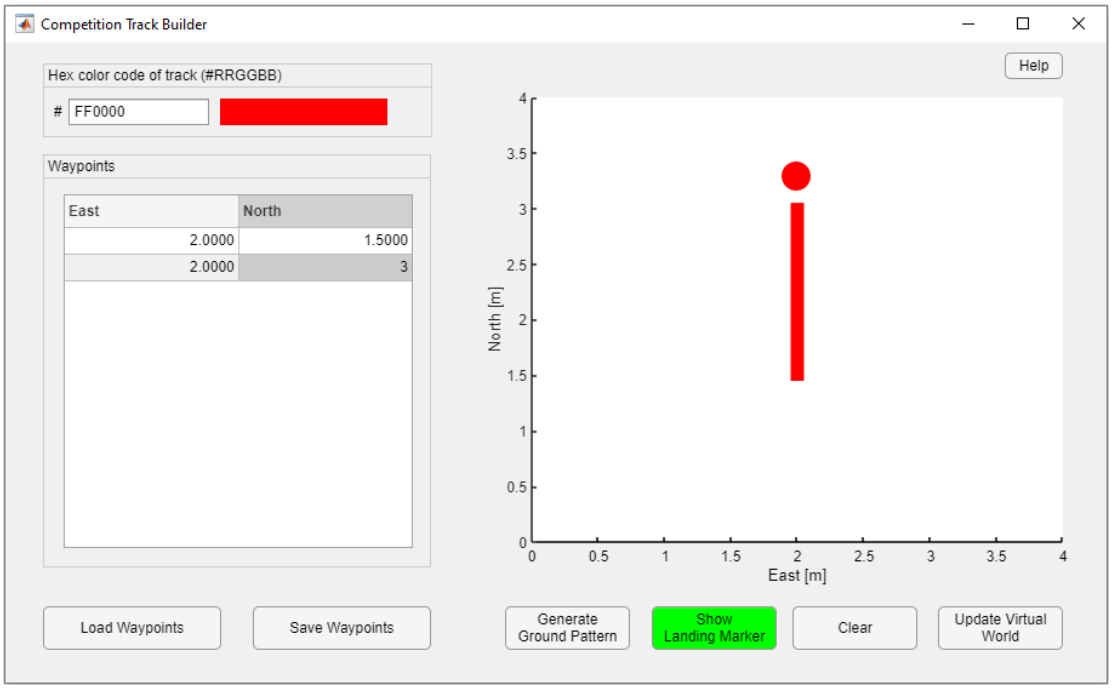


Figura 19 Modificación de la Trayectoria con Track Builder

Cabe aclarar que cuando se modifica la trayectoria, las coordenadas iniciales con las que contara el mini drone serán el inicio de la trayectoria de color rojo, para este caso se tiene que 2 m en el eje East y 1.5 m en el eje North serían los puntos iniciales y las coordenadas de la meta son el final de la trayectoria roja que están indicadas con un círculo rojo, o sea 2 m en East y 3 m en North. Este sistema de referencia sólo será necesario tenerlo en cuenta cuando se modifique la trayectoria de la pista, ya que los valores de la posición del mini drone cuando se esté corriendo la simulación serán seteados a 0 en la posición inicial de la trayectoria establecida, esto quiere decir que si se toma como ejemplo la Figura 19 en donde los puntos iniciales eran 2 m en East y 1.5 m en North, para toda la simulación por parte del modelo de Simulink siempre se tendrá que los puntos iniciales serán 0 en X y 0 en Y.

Así con estas coordenadas de posición se podrá saber gracias a el estimador de estados el punto en que se encuentre el mini drone, en toda la trayectoria que se haya modificado y establecido, hay que tener muy en cuenta que los valores de posición X, Y están por decirlo de algún modo invertidos, ya que la coordenada X indicará la posición en el eje vertical o North (teniendo en cuenta la herramienta Track Builder) y la coordenada Y indicara la posición en el eje horizontal o East.

Para este proyecto serán utilizadas dos trayectorias distintas en las que se entrenará el agente y se comprobará que cumpla con el objetivo de llegar a la meta, además se comparará temas como el tiempo de entrenamiento, las especificaciones y parámetros necesarios para que se logre el objetivo en cada una de las trayectorias estipuladas. La primera trayectoria es la que se muestra en la Figura 19, además de dos posibles pistas que son las que se muestran a continuación:

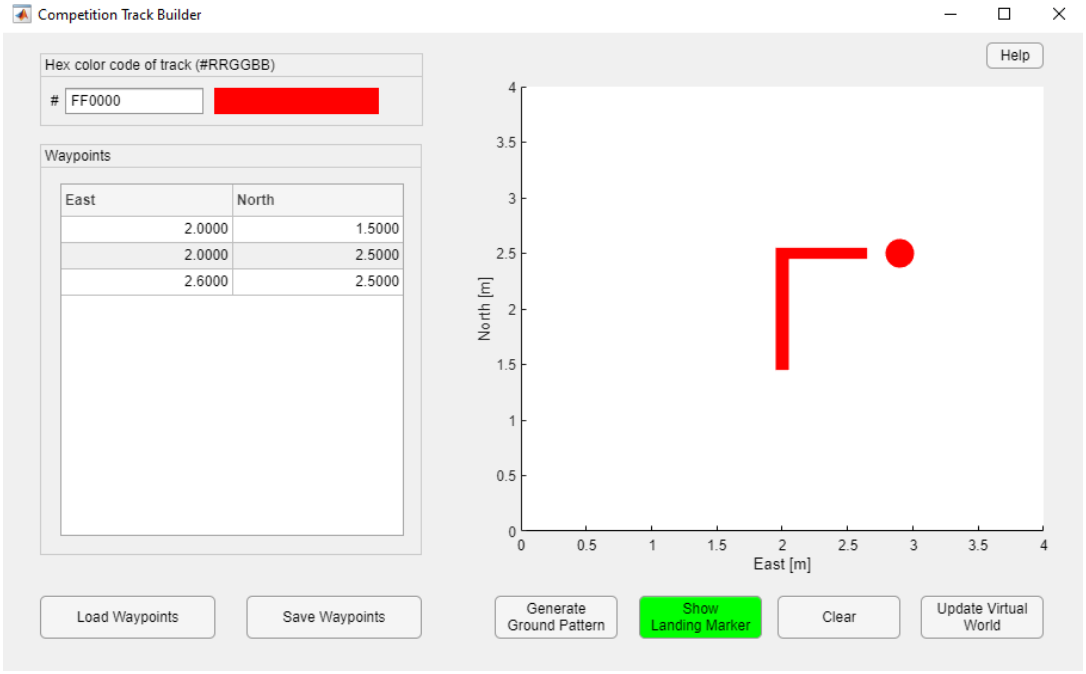


Figura 20 Modelo de Pista N°2

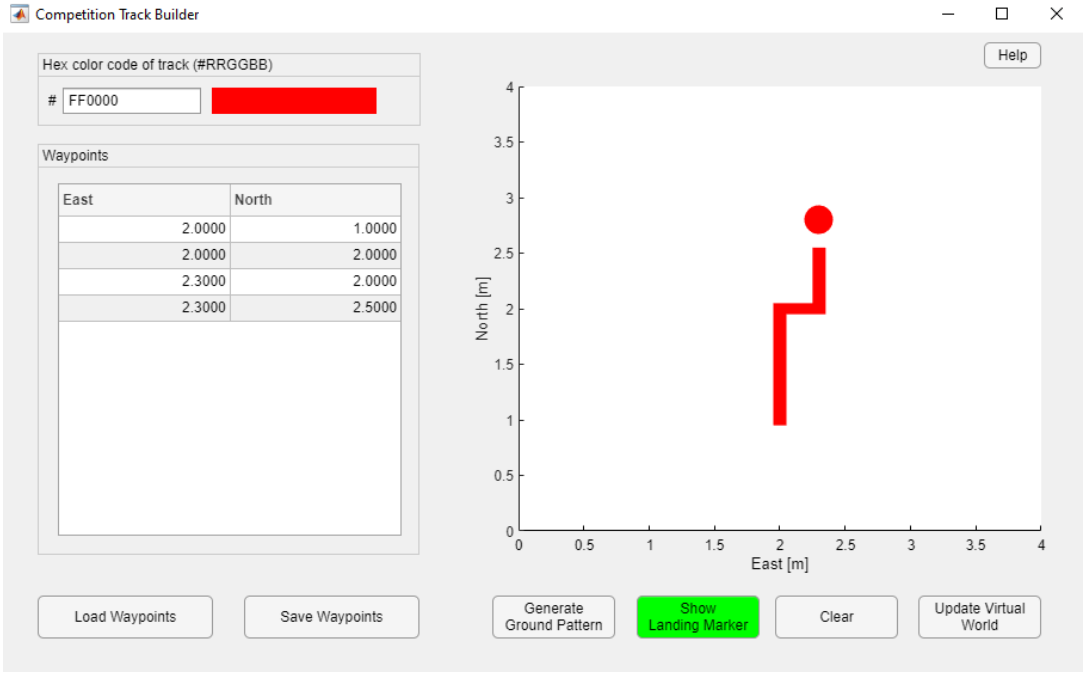


Figura 21 Modelo de Pista N°3

3.2 Política de Seguimiento Basada en una Red Neuronal

Como se mencionó anteriormente el comportamiento del agente va estar determinado por la política de seguimiento. Esta va estar compuesta por un actor y un crítico los cuales serán representados por una red neuronal cada uno, como el entorno de acción será continuo es necesario utilizar una estructura de redes neuronales para estas dos estructuras, aunque como se puede ver en la Figura 22 estas representaciones también pueden ser una serie de tablas llamadas Q-Tables diseñadas para entorno discretos, pero para fines del proyecto no es el caso.

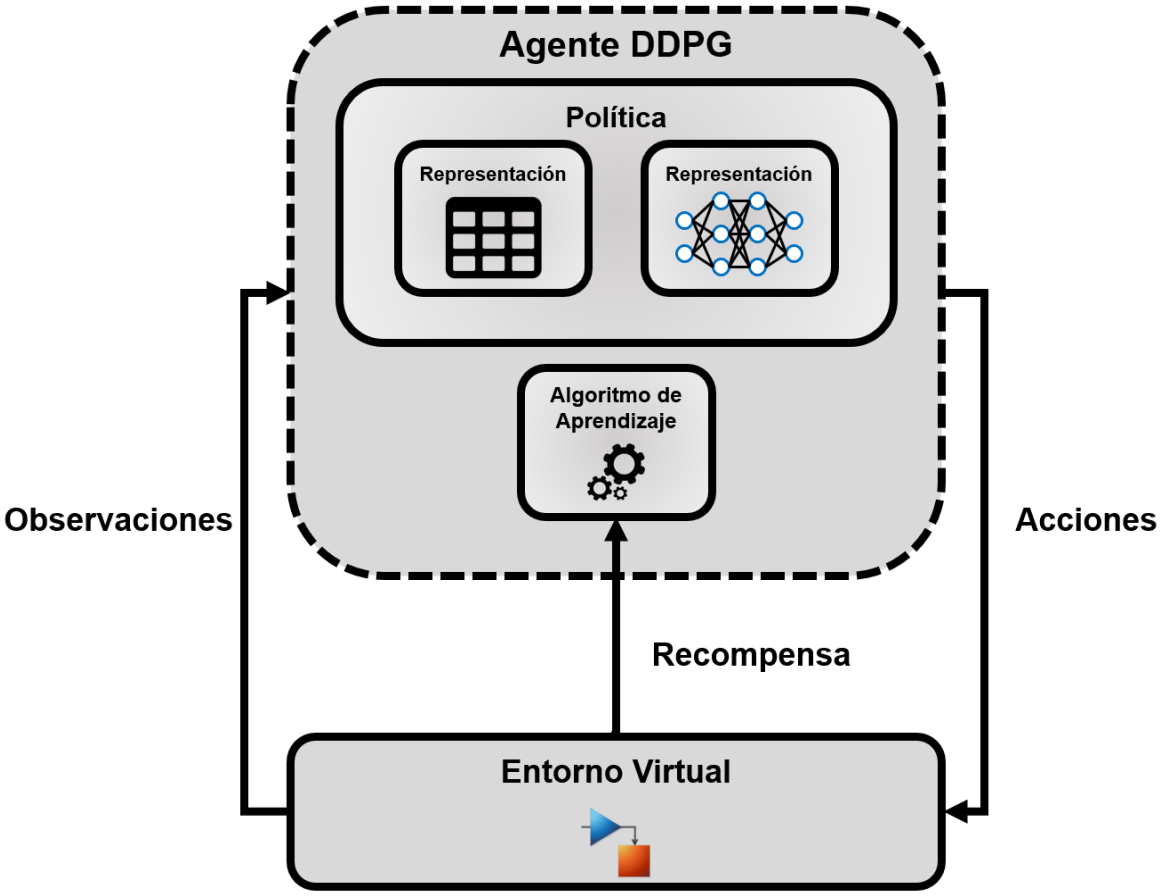


Figura 22 Representación del Agente

3.2.1 Diseño de la Red Neuronal

Una vez conocidas algunas de las estructuras y arquitecturas más comunes en la elaboración de agentes conformados por un actor y un crítico, se dispondrá a utilizar el software Matlab para realizar el diseño de la red neuronal del agente.

3.2.2 Métodos Alternos

Existen varias maneras de realizar el modelo de una red neuronal en Matlab, una de ellas es haciendo uso del Toolbox Deep Network Designer, el cual es un entorno gráfico de diseño que permite visualmente la creación de la red neuronal desde un método gráfico. Allí es posible construir las conexiones entre capas de una manera sencilla y finalmente cuando se haya terminado elaborar la estructura de la red, se puede exportar el resultado al espacio de trabajo de Matlab por medio de una variable o por un código de varias líneas el cual describe la red diseñada, el cual se encuentra guardado en un Script (Figura 20).

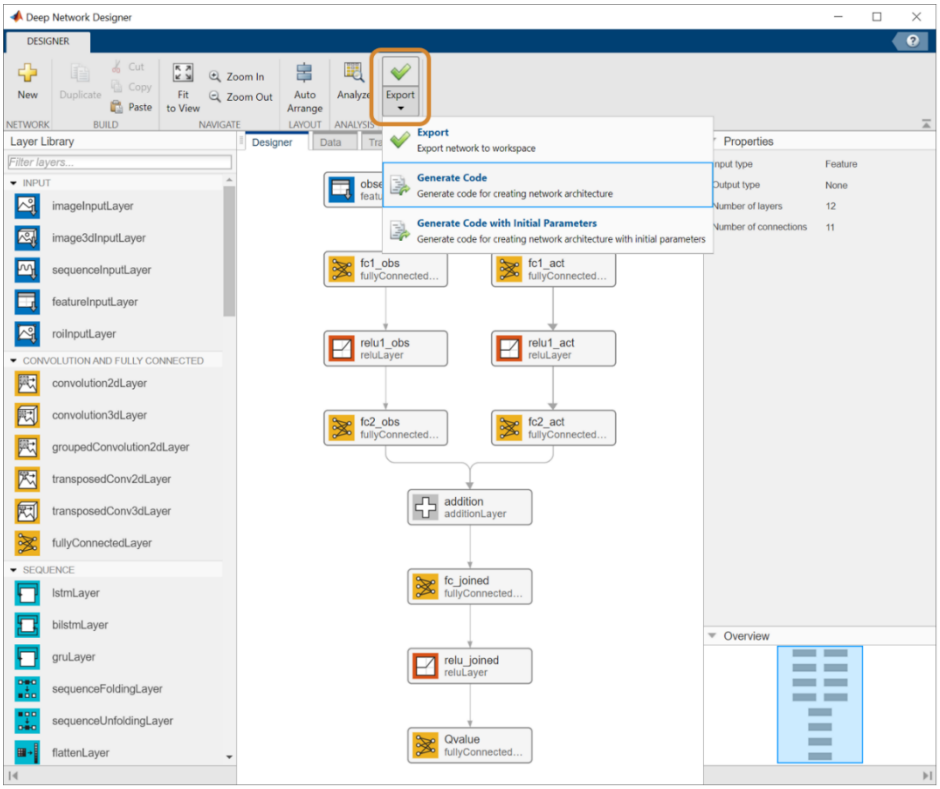


Figura 23 Deep Network Design

Otro método en la elaboración de redes neuronales para agentes y que es el más sencillo que ofrece Matlab es el de crear agentes predeterminados, esto consiste en que, a partir de una función Matlab creará un agente de la arquitectura especificada utilizando una estructura de redes neuronales predeterminada. Por ejemplo, en el caso del presente proyecto será utilizado un agente DDPG, entonces para hacer uso de esta función se tendría que utilizar el siguiente comando:

```
agent=rlDDPGAgent(obsInfo,actInfo)
```

Esto quiere decir que se puede llamar a cualquiera de las funciones establecidas para los distintos tipos de agentes actor/crítico, con las especificaciones de observación (obsInfo) y acción (actInfo) como entrada, estas dos entradas simplemente contienen el número de observaciones y acciones que se tendrían en el modelo. Así finalmente se obtendría en

una variable (*agent*) una arquitectura predeterminada de un agente DDPG. Para crear agentes predeterminados de diferentes modelos, se presenta la Tabla II en la que se enseña los tipos de agentes, el campo de acción y la función que sería necesaria para poderlos utilizar en Matlab.

Si se desea conocer la arquitectura de las redes predeterminadas o si se requiere modificar alguna opción o propiedad de estas redes neuronales, se pueden usar las funciones getActor y getCritic, para obtener el actor y el crítico de un agente. También se puede utilizar getModel para obtener la red neuronal completa del agente.

Tabla II Funciones de Representación de los Distinto Tipos de Agentes

Agent	Actions	Type	Representation(s)
Q-Learning `rlQAgent`	Discrete	Critic	`rlQValueRepresentation`
SARSA `rlSARSAAgent`	Discrete	Critic	`rlQValueRepresentation`
Deep Q-Network (DQN) `rlDQNAgent`	Discrete	Critic	`rlQValueRepresentation`
Policy Gradient `rlPGAgent`	Discrete or continuous	Actor or Actor-Critic	`rlStochasticActorRepresentation` (actor) `rlValueRepresentation` (critic)
Actor-Critic `rlACAgent`	Discrete or continuous	Actor-Critic	`rlStochasticActorRepresentation` (actor) `rlValueRepresentation` (critic)
Deep Deterministic Policy Gradient (DDPG) `rlDDPGAgent`	Continuous	Actor-Critic	`rlDeterministicActorRepresentation` (actor) `rlQValueRepresentation` (critic)
Proximal Policy Optimization (PPO) `rlPPOAgent`	Discrete or continuous	Actor-Critic	`rlStochasticActorRepresentation` (actor) `rlValueRepresentation` (critic)
Twin-Delayed Deep Deterministic Policy Gradient (TD3) `rlTD3Agent`	Continuous	Actor-Critic	`rlDeterministicActorRepresentation` (actor) `rlQValueRepresentation` (critic)
Soft Actor-Critic (SAC) `rlSACAgent`	Continuous	Actor-Critic	`rlStochasticActorRepresentation` (actor) `rlQValueRepresentation` (critic)

3.2.3 Método Utilizado

Para el desarrollo del proyecto se dispondrá a elaborar y construir la estructura de la red neuronal con una secuencia de líneas de código dispuestas en un script de Matlab, de esta manera se elaborará completamente la red desde cero de una forma manual, teniendo así un control total sobre los componentes del agente, ya que se podrá especificar los detalles de cada una de las representaciones, tanto del actor como del crítico.

Primero se debe tener en cuenta que tanto el agente, el entorno, las observaciones y las acciones son todas representadas en Matlab como variables, así que para especificar cómo se definen las observaciones y las acciones se debe utilizar la función:

rlNumericSpec

Esta función es utilizada cuando las acciones y las observaciones son valores numéricos que no forman de un conjunto finito de estados observables y posibles acciones, ya que

los estados del entorno y la combinación de las acciones que se pueden realizar en el modelo del mini dron serán de cierto modo infinitas, si el objeto de estudio fuera en un espacio de acción discreto en el que las observaciones y acciones fueran finitas se podría usar la función *lFiniteSetSpec*. Posteriormente se procede a definir la variable del entorno a partir del modelo de Simulink Minidrone Parrot Competition, con la siguiente función:

rlSimulinkEnv

Las entradas de esta función son el nombre del modelo de Simulink y las variables que definen las especificaciones de observación y acción, vistas previamente. Con estas dos funciones se pueden crear las primeras variables del entorno de la siguiente manera:

```
obsInfo = rlNumericSpec([6 1]);
actInfo = rlNumericSpec([2 1],"UpperLimit",2,"LowerLimit",0);
env =
rlSimulinkEnv("parrotMinidroneCompetition","flightControlSystem/Co
ntrol System/Path Planning/controller",obsInfo,actInfo);
```

En el modelo de Simulink (Figura 8) se tienen seis observaciones de cada uno de los estados posibles, esto se debe a que un mini dron posee seis grados de libertad (tres de traslación y tres de rotación), por esto se puede apreciar que, en la primera línea del código anterior, se declara la variable *obsInfo* con la función correspondiente `rlNumericSpec` y como entrada se tiene el vector de dimensiones de los posibles estados observados, es decir `[6 1]`.

Del mismo son declaradas las acciones con la variable *actInfo,* allí se aprecia que las acciones son dos, esto se debe a que el agente solo va estar encargado de controlar el dron en dos direcciones de traslación (*X* y *Y*), la altura del mini dron será una constante la cual lo mantendrá a una misma altura en toda la simulación, en este caso la altura predeterminada será de 1.1 *m.* Las tres variables de rotación que son yaw, pitch y roll tendrán un valor de cero, ya que para fines del proyecto no es necesario la utilización de estas variables, además de que al reducir el espacio de acción a dos dimensiones, será mucho más factible la utilización de inteligencia artificial para el control autónomo del mini drone y los tiempos de entrenamiento serán mucho menores a comparación de tener un diseño con más acciones, de otro modo el planteamiento de controlar un mini dron con tantas posibles acciones sería un reto de una complicación elevada.

Cuando se declaran las acciones en la variable *actInfo* en el código anterior, se limitaron las acciones posibles en un rango de `[0 2]`, con la ayuda de los comandos `UpperLimit` y `LowerLimit` que determinan el rango superior e inferior de las acciones, esto significa que las acciones que ejecuta el agente son las posiciones en X y Y del mini dron y estas solo tendrán un rango de 0 m a 2 m, ya que como están diseñadas las pistas (Figura 9, 10 y 11) el mini drone no necesitará moverse más allá de estos límites, además como solo se tienen dos posibles acciones el vector de dimensiones sería `[2 1]`. El valor numérico de las posiciones del mini dron que van a ser determinadas por el agente, en otras palabras,

las acciones, son valores de referencia de la posición que necesitará los sistemas de control del modelo de Simulink Minidrone Parrot Competition, estos sistemas como el *"Multicopter Model"* son los que se encargan de calcular los torques y fuerzas que necesita cada uno de los motores del mini dron y así éste puede realizar las trayectorias para las cuales va a ser entrenado.

Y por último es utilizada la función `rlSimulinkEnv` que crea el entorno en el cual va estar funcionando todo el conjunto de la inteligencia artificial, hay que tener en cuenta que al momento de utilizar la función se debe especificar precisamente la ruta en la que se encuentra el agente en el modelo de Simulink, así que si el agente que se va utilizar se encuentra entre algunos subsistemas del modelo se debe colocar precisamente donde se va a encontrar. Además, se debe colocar los parámetros `obsInfo` y `actInfo` que son las especificaciones de entorno para las observaciones y las acciones.

3.2.3.1 Elaboración del Agente Determinístico

Una vez declaradas las variables del entorno, las observaciones y las acciones, se procede a diseñar la estructura de la red neuronal del agente. Primero se comenzará con la construcción de las capas del actor determinista que va a componer el agente, como se observa en la Figura 9, este tipo de actores en la primera capa que es la capa de entrada se van a recibir las observaciones, como se había mencionado antes en total van a ser seis observaciones, así que se necesitará que esta primera capa tenga seis neuronas de entrada, posteriormente se encontraran las capas ocultas en las que se encuentran las redes neuronales completamente conectadas además de las de activación y por último la capa de salida que tendrá la misma cantidad de neuronas como de acciones que va a realizar el agente, en este caso 2. Para realizar esta estructura de capas en Matlab simplemente se tiene que indexar cada capa como si fuera cada una un elemento de un array, como se puede observar de la siguiente manera:

```
actnet=[featureInputLayer(6,"Name","obs"),
fullyConnectedLayer(100,"Name","fc1"),
reluLayer("Name","relu1"),
fullyConnectedLayer(100,"Name","fc2"),
reluLayer("Name","relu2"),
fullyConnectedLayer(100,"Name","fc3"),
reluLayer("Name","relu3"),
fullyConnectedLayer(2,"Name","act"),
tanhLayer("Name","scact")]
```

Primero se debe nombrar la red de capas que representará el actor determinista, en este caso será llamada `actnet`, después se debe comenzar con una capa de entrada la cual va a recibir las observaciones del entorno, como se mencionó antes se cuenta con seis observaciones disponibles por esto es que al momento de utilizar la función

`featureInputLayer` para crear la capa de entrada, es especificado que debe tener seis neuronas correspondientes a cada observación. A continuación, se observa que cada una de las capas se le asigna un nombre, para que al momento de ir construyendo una red de capas sea algo más organizado y sencillo de comprender, a esta capa de entrada se le asignó el nombre `obs` con la ayuda de la opción `"Name"`.

Posteriormente se añade una capa completamente conectada con la función `fullyConnectedLayer,` se especifica que debe de tener cien neuronas, este es un valor que es elegido como inicial y si en etapas posteriores como la de entrenamiento y/o validación del agente, no se llega a un comportamiento deseado por parte de éste, uno de los parámetros que se pueden modificar es el de la cantidad de neuronas, aumentando esta característica que se tiene solo en las capas completamente conectadas de las capas ocultas o intermedias del agente. Conectada a esta segunda capa se tiene una capa de activación para evitar la linealidad, más específicamente se tiene la capa de la función de unidad lineal rectificada (ReLU), la cual pone a cero cualquier valor negativo a cero y deja los valores positivos sin modificar, la función utilizada es `reluLayer`, esta no requiere ninguna especificación en la entrada, pero de igual manera se le puede asignar un nombre.

Luego se siguen añadiendo más capas, al igual que el número de neuronas el número de capas es un parámetro que puede cambiar según el caso de aplicación, para este proyecto se eligió una configuración de nueve capas en total junto con las capas de entrada y salida. Hay que tener en cuenta que la capa de salida debe tener dos neuronas, que corresponden al número de acciones que puede realizar el agente, que en este caso sería el movimiento en X y Y. Finalmente se tienen la siguiente configuración:

```
actnet =

  9×1 Layer array with layers:

     1   'obs'     Feature Input     6 features
     2   'fc1'     Fully Connected   100 fully connected layer
     3   'relu1'   ReLU              ReLU
     4   'fc2'     Fully Connected   100 fully connected layer
     5   'relu2'   ReLU              ReLU
     6   'fc3'     Fully Connected   100 fully connected layer
     7   'relu3'   ReLU              ReLU
     8   'act'     Fully Connected   2 fully connected layer
     9   'scact'   Tanh              Hyperbolic tangent
```

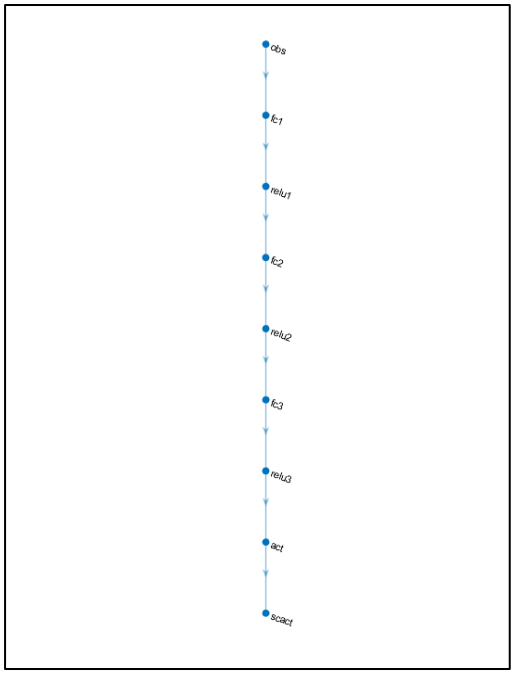


Figura 24 Configuración de Capas del Actor Determinista

Como se puede ver en la Figura 24, el actor cumple con cada una de las partes que componen un actor determinista como se muestra en la arquitectura de una red neuronal de la Figura 9. Siguiendo con el mismo concepto ahora se realizará la construcción del crítico del agente.

Para finalizar con la elaboración del actor, se debe crear la variable de Matlab la cual almacene esta estructura de capas que representarán al actor determinista, además que se tiene que especificar la capa de la red que debe ser conecta con las observaciones y de igual manera con la capa de las acciones, esto se realiza de la siguiente manera:

```
actor=rlDeterministicActorRepresentation(actnet,obsInfo,actInfo,"O
bservation","obs","Action","scact")
```

La variable que va a almacenar el agente será `actor`, con la función `"rlDeterministicActorRepresentation"` primero se indica el nombre de la red neuronal que representará a el agente `(actnet)`, después se especifica cuáles son las variables en las que se van a encontrar las observaciones y acciones (`obsInfo` y `actInfo`), y por último se define específicamente que capa de la red es la que debe estar conectada a las observaciones y las acciones. Se puede observar en el código anterior que para realizar esta conexión es necesario utilizar el nombre que se le asignó a cada capa en el momento de crear la estructura de la red. Por esto el segmento del código anterior `"Observation","obs"` se refiere a que las observaciones que se van a encontrar en la variable `obsInfo`, serán conectadas a la capa llamada `"obs"` de la red neuronal `"actnet"` y del mismo modo pasará con el segmento de código `"Action","scact"` el cual se encargará de conectar las acciones de la variable

correspondiente, a la capa del actor correspondiente, que en este caso es llamada `"scact"`. Con esto se habrá terminado la elaboración del agente determinístico.

3.2.3.2 Elaboración del Crítico de Valores-Q

Como el tipo de agente que se está empleando en este proyecto es un agente DDPG (Deep Deterministic Policy Gradient), el crítico debe ser un crítico de valores Q, en la Figura 9 se observa cómo es la estructura que debe seguir un crítico de este tipo. Como la arquitectura de la red toma dos entradas (observaciones del ambiente y acciones del actor), esto requiere una red con dos rutas de capas que se fusionan posteriormente, para producir una única salida que vendría siendo el Valor-Q (Figura 22).

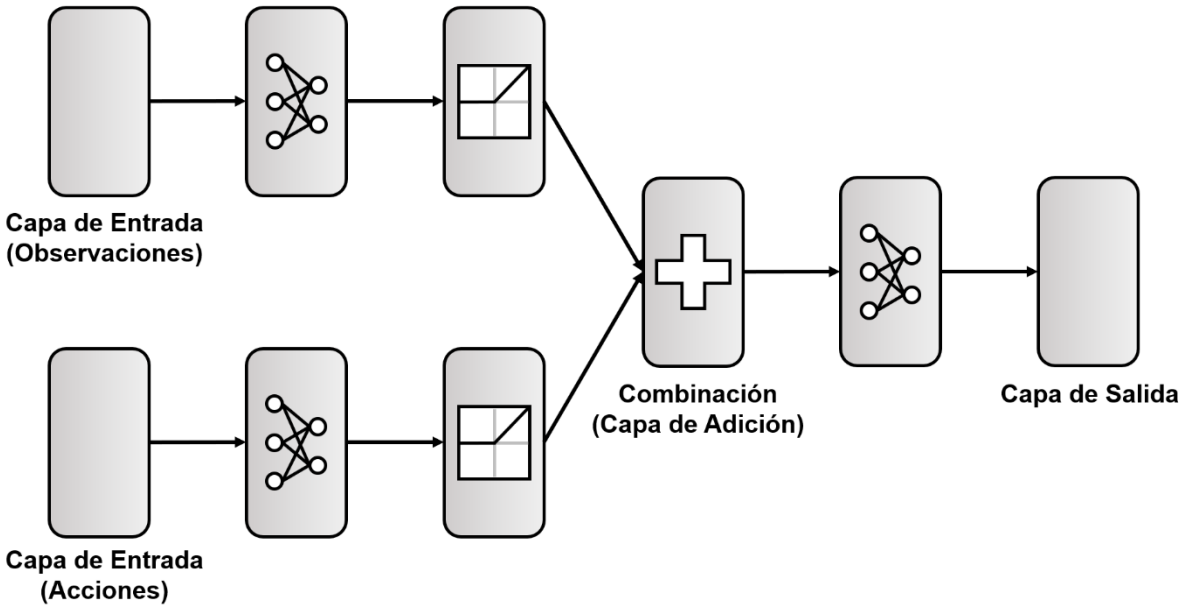


Figura 25 Arquitectura Crítico de Valores-Q

Una red con múltiples rutas conectadas se conoce como red de gráfico acíclico dirigido (DAG).

Para crear una red con múltiples rutas en Matlab, primero se debe crear cada ruta por separado, al igual que la forma anterior se debe crear un array de capas, para que después se pueda conectar todas las rutas en una sola red. Primero se comenzará a construir la ruta de las observaciones, de la siguiente manera:

```
obsPath=[featureInputLayer(6,"Name","obs"),
fullyConnectedLayer(100,"Name","fc1"),
reluLayer("Name","relu1"),
fullyConnectedLayer(100,"Name","fc2")]
```

Esta primera ruta será llamada `obsPath`, la cual tendrá una capa de entrada de seis neuronas correspondiente a los seis posibles estados observables, luego tendrá una

combinación de capas completamente conectadas de cien neuronas, junto a una capa de activación ReLU, como muestra la siguiente estructura:

```
obsPath =

  4×1 Layer array with layers:

     1   'obs'     Feature Input     6 features
     2   'fc1'     Fully Connected   100 fully connected layer
     3   'relu1'   ReLU              ReLU
     4   'fc2'     Fully Connected   100 fully connected layer
```

Ahora se procede a crear la segunda ruta que corresponde a las acciones, de la siguiente manera:

```
actPath=[featureInputLayer(2,"Name","act"),
fullyConnectedLayer(100,"Name","fcact")]
```

Algo importante a tener en cuenta es que los nombres de las capas son únicos y deben ser diferentes, además no deben confundirse o cambiarse con otros, esto con el fin de distinguir entre dos capas diferentes y al momento de realizar la conexión resulte de una manera adecuada. El nombre designado para la ruta de las acciones del crítico es `actPath`, la capa de entrada posee dos neuronas que pertenecen a las dos acciones que realizará el agente, luego tendrá una sola capa completamente conectada adicional de cien neuronas, teniendo así que:

```
actPath =

  2×1 Layer array with layers:

     1   'act'     Feature Input     2 features
     2   'fcact'   Fully Connected   100 fully connected layer
```

Cabe aclarar que tanto la ruta de las observaciones como la de las acciones son un conjunto de capas que no son muy extensas, apenas poseen una estructura de pocas capas, ya forman de un conjunto de una red neuronal mayor (Crítico de Valores-Q), hay que tener presente que además de estas dos rutas hace falta la ruta la cual conecta los dos “caminos” y como resultado entrega el Valor-Q. Por esto no tienen que ser tan extensas las dos primeras rutas, de lo contrario resultaría una estructura de una red neuronal con unas dimensiones algo extensas, donde se tendría que tener contemplado que el tiempo en el periodo de entrenamiento es mayor y de igual manera los requerimientos computacionales serían más exigentes. Cuando se comienza por primera vez el diseño y construcción de las distintas estructuras actor – crítico que constituyen el agente, es recomendable empezar con una red neuronal simple con una estructura de pocas capas y neuronas, e incluso otra opción sería comenzar con una red predeterminada como se describió en el numeral *4.3.1 Métodos Alternos*. Pero si el agente diseñado no parece ser capaz de aprender y realizar el objetivo propuesto, lo ideal es comenzar a añadir capas ocultas y aumentar el número de neuronas de cada capa.

Ahora se procede a crear la última capa que conectará la ruta de las observaciones y la ruta de las acciones, como se enseña en el siguiente código:

```
joinedpath=[additionLayer(2,"Name","add"),
reluLayer("Name","relu2"),
fullyConnectedLayer(100,"Name","fc3"),
reluLayer("Name","relu3"),
fullyConnectedLayer(1,"Name","value")]
```

El nombre designado para esta última ruta es `joinedpath`, donde la primera capa o capa de entrada es la función `additionLayer`, la cual se encarga de fusionar o sumar las salidas de las dos capas que la van a preceder, las salidas de las capas que van a ser agregadas deben de tener las mismas dimensiones. Por esto la ruta de las observaciones como la de las acciones finalizan con una capa completamente conectada de cien neuronas. Como se puede apreciar a continuación:

```
joinedpath =

  5×1 Layer array with layers:

     1   'add'     Addition          Element-wise addition of 2
                                     inputs
     2   'relu2'   ReLU              ReLU
     3   'fc3'     Fully Connected   100 fully connected layer
     4   'relu3'   ReLU              ReLU
     5   'value'   Fully Connected   1 fully connected layer
```

Para especificar el número de rutas que van a ser conectadas a la `additionLayer`, se debe colocar en los parámetros el número *n* de rutas a ser conectadas, como en este caso solo se tienen dos rutas se colocará dos en las opciones, junto con el nombre de la capa que será `add`, los nombres de las distintas capas serán de gran utilidad para realizar la conexión final de las tres rutas que se acaban de elaborar.

Además en el código anterior se puede ver que esta ruta de adición, tiene un conjunto de capas de activación ReLU y también posee capas completamente conectadas, pero lo más importante es que la última capa de esta tercera ruta sea una capa completamente conectada de una sola neurona, como muestra la última línea del código, esto se debe a que como esta capa es la última capa de la red neuronal que representara a él crítico, debe de tener como salida un solo valor que en otras palabras vendría siendo el valor de calidad o Valor-Q.

En resumen, se tiene que las rutas realizadas hasta el momento y que van a componer el Crítico de Valores-Q son:

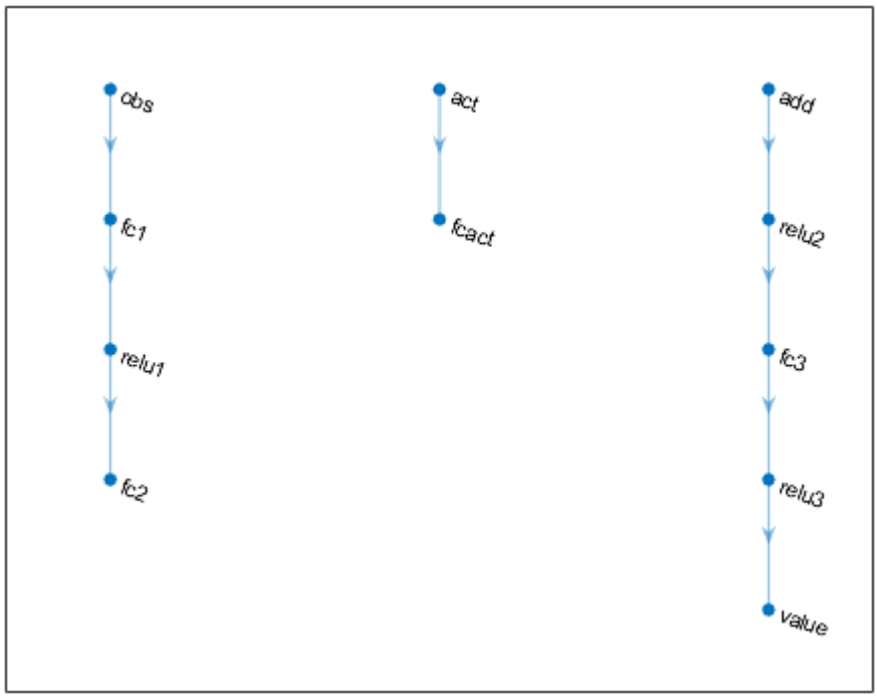


Figura 26 Rutas del Crítico de Valores-Q

Esta forma gráfica que enseña la Figura 26, la cual permite visualizar las distintas estructuras de redes neuronales que se van elaborando ya sea para el actor o el crítico, se realiza con la ejecución del siguiente código:

```
net=layerGraph(obsPath)
net=addLayers(net,actPath)
net=addLayers(net,joinedpath)
plot(net)
```

La función `layerGraph`, permite crear una red gráfica a partir de los arrays de capas creados anteriormente, además se puede utilizar la función `addLayers`, para agregar otro array de capas a una red de gráficos existente, de esta manera se van añadiendo las estructuras creadas al mismo gráfico original y por último se utiliza la función `plot` para graficar la red de capas que se encuentra almacenada en la variable `net`, dando como resultado la Figura 26. De la misma manera se elaboró la red gráfica de las capas del actor en la Figura 24.

Hasta el momento se han construido las tres rutas, pero estas aún no están conectadas entre sí. Para realizar la conexión entre las capas correspondientes se utilizará la función:

```
net=connectLayers(net,"from","to")
```

Donde “from” y “to” son los nombres de las capas que se van a unir, por lo general las capas de adición que son las que conectan o fusionan dos rutas distintas, tienen varias entradas, como en este caso son dos entradas se tiene que especificar la conexión

utilizando el orden "layername/portname", aquí el nombre de la capa es la que se especificó anteriormente en la elaboración de la ruta y el nombre del puerto para una capa de adición por defecto se denominan “in1”, “in2”, etc. Como se puede apreciar en la siguiente imagen:

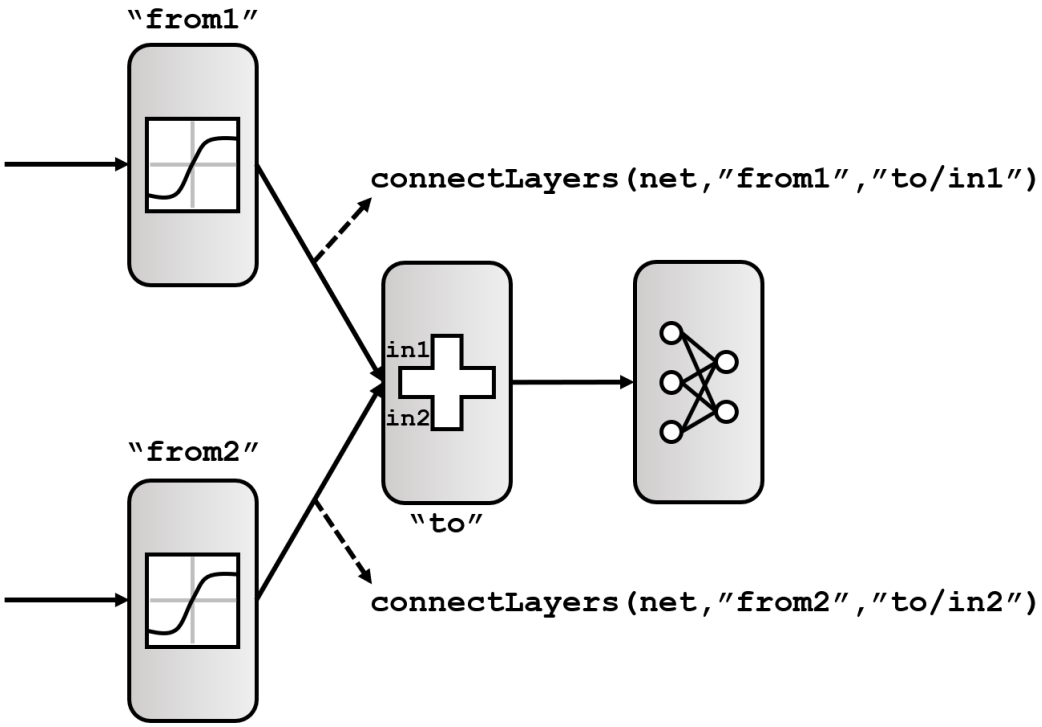


Figura 27 Conexión Capas de Adición

De este modo se procede a realizar la unión de las tres rutas, con el segmento de código que se tiene a continuación:

```
qvalnet=connectLayers(net,"fc2","add/in1")
qvalnet=connectLayers(net,"fcact","add/in2")
plot(qvalnet)
```

Teniendo en cuenta la Figura 26 se realiza la conexión de una manera sencilla y rápida, primero la capa final de la ruta de las observaciones tiene el nombre de `“fc2”`, así que se le asigna a esta el primer puerto de entrada de la capa de adición `“add/in1”`. Luego la capa final de la ruta de las acciones es llamada `“fcact”`, la cual se le asigna el segundo puerto de entrada de la capa de adición `“add/in2”`. El nombre de la variable que almacenará toda la estructura de las capas que representarán a el crítico será `qvalnet`.

Finalmente, con la función `plot` se podrá ver la red gráfica de la estructura de capas que representarán el Crítico de Valores-Q, que tendrá el agente de la inteligencia artificial que se encargará del control autónomo del mini drone. Como se puede ver en la Figura 28, el crítico cumple con cada una de las partes que componen un Crítico de Valores-Q , como se muestra en la arquitectura de una red neuronal para un agente DDPG de la Figura 9.

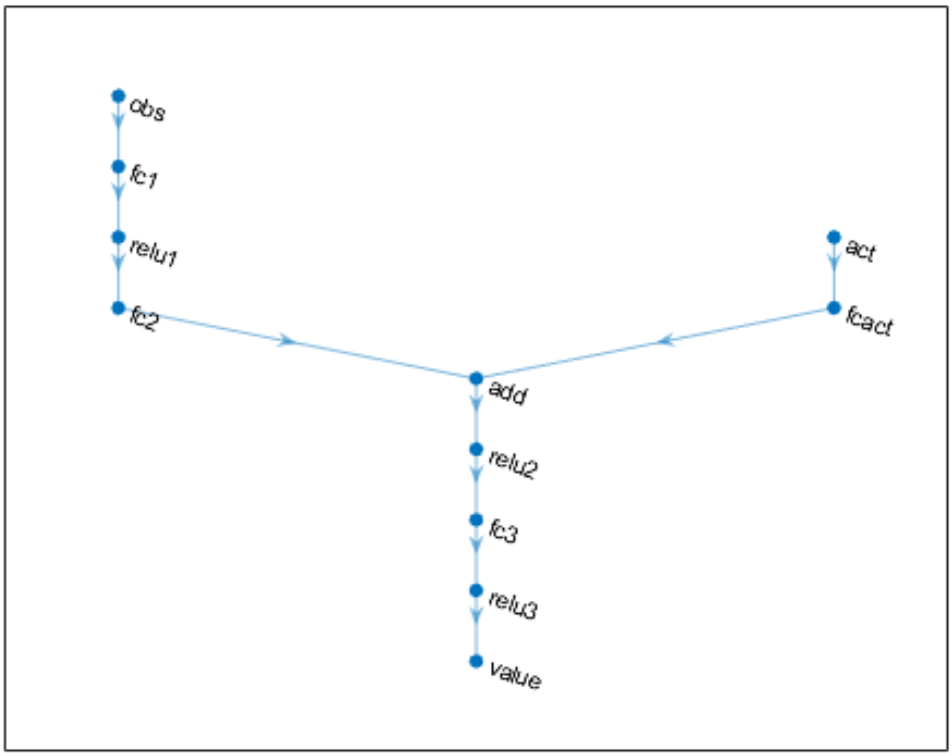


Figura 28 Configuración de Capas del Crítico de Valores-Q

Para finalizar con la elaboración del crítico, se debe crear la variable de Matlab la cual almacene esta estructura de capas que representarán a él Crítico de Valores-Q, además que se tiene que especificar la capa de la red que debe ser conectada a las observaciones y de igual manera con la capa de las acciones, esto se realiza de la siguiente manera:

```
critic=rlQValueRepresentation(qvalnet,obsInfo,actInfo,"Observation
","obs","Action","act")
```

La variable que va a almacenar el crítico será `critic`, con la función `"rlQValueRepresentation"` se puede crear un crítico a partir de una red gráfica, como la que se visualiza en la Figura 28. Primero se indica el nombre de la red neuronal que representará a el crítico `(qvalnet)`, después se especifica cuáles son las variables de entorno en las que se van a encontrar las observaciones y acciones (`obsInfo` y `actInfo`), y por último se define específicamente que capa de la red es la que debe estar conectada a las observaciones y las acciones. Se puede ver en el código anterior que para realizar esta conexión es necesario utilizar el nombre que se le asignó a cada capa en el momento de crear la estructura de la red. Por esto el segmento del código anterior `"Observation","obs"` se refiere a que las observaciones que se van a encontrar en la variable `obsInfo`, serán conectadas a la capa llamada `"obs"` de la red neuronal "`qvalnet`" y del mismo modo pasará con el segmento de código `"Action","act"` el cual se encargará de conectar las acciones de la variable correspondiente, a la capa del

crítico correspondiente, que en este caso es llamada `"act"`. Con esto se habrá completado toda la elaboración del Crítico de Valores-Q.

En conclusión, para la elaboración de un sistema autónomo de control para mini drones, se debe tener una pieza de software que sea la encargada de tomar las acciones adecuadas según las observaciones que se tengan del entorno, a esto en aprendizaje por refuerzo se le llama agente. El agente internamente está constituido por dos secciones, una llamada actor encargado de ejecutar las acciones y por otra parte el crítico que se encarga de estimar que tan acertada fue la acción ejecutada por el actor. Y así con la ayuda del algoritmo de entrenamiento se concretará una inteligencia artificial capaz de realizar el objetivo para la cual fue diseñada.

Teniendo en cuenta esto, las dos secciones que componen a él agente ya fueron diseñadas y elaboradas anteriormente, por parte del actor determinista se ecuentra la variable `actor` y por parte del crítico de Valores-Q esta la variable `crítico`, las cuales representan a cada sección por medio de una estructura de capas. Ahora simplemente queda crear la variable de Matlab en la que se va a encontrar el actor de la inteligencia artificial del sistema de control.

Para esto se puede usar la función `rlDDPGAgent`, la cual creará un agente DDPG (Deep Deterministic Policy Gradient o Gradiente de Política Determinista Profunda), a partir del actor determinista y el crítico de Valores-Q ya creados, entonces se tiene que:

```
agent=rlDDPGAgent(actor,critic)
```

Con este proceso ya estaría finalizado la creación del agente, un aspecto a tener en cuenta es que cuando se crea un actor o un crítico para la elaboración de un agente, los valores de los pesos y sesgo de las diferentes redes neuronales que componen estas secciones, son inicialmente aleatorios y como es de esperar el agente no tendrá un comportamiento adecuado. Para esto se debe realizar el entrenamiento, periodo en el cual estos parámetros internos serán ajustados por el algoritmo de entrenamiento, esto será profundizado en la sección 3.4.

Un factor que es muy importante y hay que tener en cuenta en el momento de la elaboración de un agente, son sus opciones internas. Estas son una serie de parámetros que definen el comportamiento del agente, entre estas opciones están: la configuración de los diferentes modelos de ruido que se encuentran presentes a la hora de realizar una acción por parte del actor, también se encuentra como una opción la sintonización del tiempo de muestreo del agente, el cual es el intervalo de tiempo que tendrá este para realizar una acción. Existen otras opciones que son más específicas y según sea el caso de aplicación será necesario realizar alguna modificación.

Por lo general cuando se crea un agente desde cero todas estas opciones vienen por defecto en valor estipulado por Matlab, en una primera instancia sólo será necesario cambiar dos parámetros, los cuales son:

- ResetExperienceBufferBeforeTraining: Esta opción es utilizada para borrar el búfer de experiencia antes de iniciar algún entrenamiento y viene por defecto en “true”. Lo ideal es que, si se va a realizar varios entrenamientos con el mismo agente, éste siempre tenga guardado su búfer de experiencia, que es donde se almacenan todos los datos y parámetros que ha ido obteniendo de entrenamientos pasados, por esto debe ser cambiado a “false”

- SaveExperienceBufferWithAgent: Con esta opción se puede guardar los datos del búfer de experiencia en el momento que se guarda un agente después del entrenamiento, por defecto está establecida en “false”. Así que, si se planea capacitar aún más a un agente guardado, se puede comenzar a entrenar con el búfer de experiencia anterior como punto de partida, es algo similar a la opción anterior, solo que en esta opción se debe modificar a “true”.

Para realizar este proceso en Matlab se puede realizar de forma manual dirigiéndose a la variable en la que está guardada el agente, o también se puede hacer uso de la función `rlDDPGAgentOptions`, de la siguiente manera:

```
agentOptions=rlDDPGAgentOptions('ResetExperienceBufferBeforeTraini
ng',false,'SaveExperienceBufferWithAgent',true)
```

En este paso se creó la variable `agentOptions`, que estará guardando qué opciones internas del agente serán modificadas, en este caso fueron solo dos, pero pueden ser todas las que se necesiten y se requieran para el caso de aplicación en cuestión. Ya por último para aplicar estos cambios se procede a volver a utilizar la función `rlDDPGAgent`, como se mostró anteriormente, pero con una leve modificación, como se enseña en la siguiente línea del código:

```
agent=rlDDPGAgent(actor,critic,agentOptions)
```

Una vez está la variable creada finalizaría todo el proceso de creación de un agente, ahora se tiene que pasar a una etapa de entrenamiento, donde el agente aprenderá la mejor política de seguimiento a medida que interactúa con el entorno virtual, de modo que, siempre ejecute la acción más óptima dependiendo del estado que se encuentre. Todo el proceso de la función de recompensa y entrenamiento serán explicados más a profundidad en los siguientes capítulos.

3.3 Sistema de Recompensas para el Algoritmo de Aprendizaje

En esta sección se desarrollarán los diferentes sistemas de función de recompensa para cada una de las trayectorias propuestas en el numeral *3.4*, como cada una de las trayectorias es distinta se tiene que modificar los parámetros del entorno de simulación y a su vez las funciones de recompensa, para que en cada una de las pistas la inteligencia artificial que controla el mini dron cumpla con el objetivo de llegar a la meta, que en todas las trayectorias dicha meta está indicada con un círculo de color rojo.

3.3.1 Función de Recompensa Primer Modelo de Pista

El primer modelo de pista es el que se indica en la Figura 29, la trayectoria que tiene que seguir el mini drone es una línea recta con una longitud aproximada de 1.5 m. La función de recompensa tiene como objetivo, de algún modo guiar el mini drone hacia el punto de meta. Al tener una línea recta vertical como trayectoria, lo ideal es que el movimiento solo se produzca en el eje X, ya que si hay movimiento en el eje Y significaría que el mini drone se ha salido de la pista, y por lo tanto tendrá algún tipo de penalización en la función de recompensa. Hay que recordar que el modelo Parrot Minidrone Toolbox tiene su sistema de ejes invertido, por ende, el eje vertical es considerado como el eje X y el eje horizontal como el eje Y.

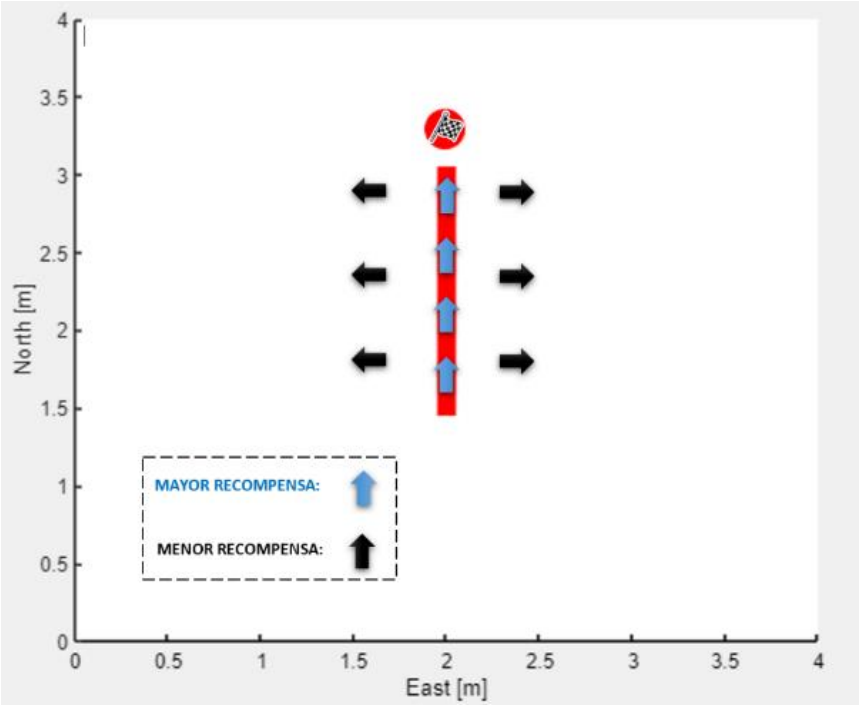


Figura 29 Recompensa Pista N°1

Cuando se tiene un agente basado en una función de valor o de recompensa, como es el caso, dicha función tomará el estado del entorno y generará un valor de recompensa, de tal modo que:

$$valor\ de\ recompensa = función\ (observaciones\ del\ estado)$$

Así que, si se quiere recompensar a el agente por realizar movimientos en el X, se debe tener una función de recompensa que refleje aquel incentivo, para esto a continuación se presenta la función que cumple con este requerimiento:

$$r = 0.5 \cdot e^{(x)} - 1$$

En esta ecuación o función de valor, *r* corresponde a el valor numérico de la recompensa y *x* es valor de la posición en el eje vertical, que tendría el mini drone en cualquier estado en el que se encuentre, para visualizar de una manera sencilla el comportamiento de esta función se tiene el siguiente gráfico:

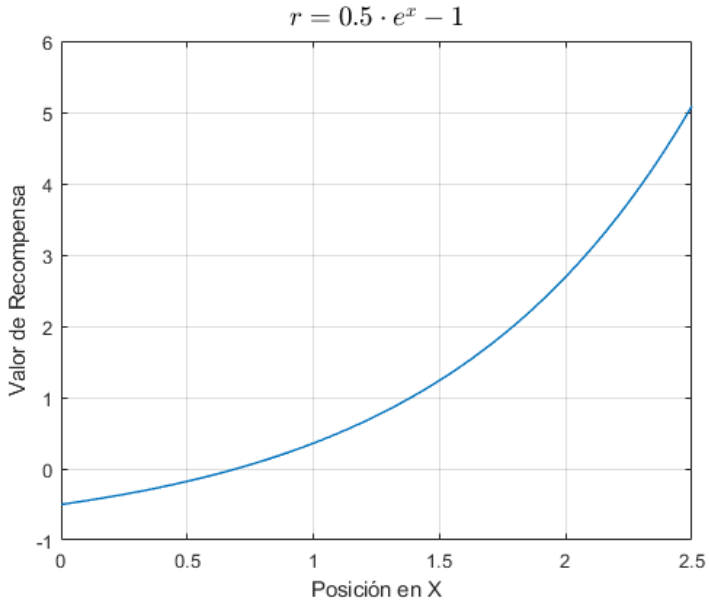


Figura 30 Recompensa Posición Eje X

Se puede observar que a medida que la posición en X aumenta el valor de la recompensa también lo hará, así que cuando el agente ejecute acciones que correspondan a mover el mini drone en el eje X positivo, tendrá una recompensa positiva. También para incentivar a que el agente no se quede en la posición inicial (0 m en X y 0 m en Y), la función de recompensa anterior penaliza al agente si este se queda en valores de posición menores a aproximadamente 0.6 m en el eje X.

Cuando se elabora una función de recompensa no existen restricciones ni reglas, esta puede tomar valores de recompensa que se encuentren en cualquier rango, por ejemplo, pueden estar en un rango de [-10 a 10] o de [-500 a 1000], ya que el algoritmo de entrenamiento siempre tratará de buscar el mayor valor de recompensa que sea posible,

de tal modo que no importara en gran medida que los valores de recompensa estén en rangos de magnitud grande o pequeña.

Hasta el momento solo se ha propuesto una función de recompensa que tiene en cuenta los valores de la posición del mini dron en el eje vertical, lo que conlleva a una situación en la que la recompensa es escasa. Esto significa que con muy pocas variables de estado o en este caso con solo una, se está tratando de incentivar al agente para que aprenda qué acciones debe realizar de una manera adecuada y eficiente. Lo que deriva en un problema de extensos periodos de entrenamiento, en los cuales la mayoría del tiempo el agente se dedicará a probar diferentes acciones y visitar muchos estados sin recibir ninguna recompensa en el transcurso del episodio de entrenamiento, y por lo tanto sin aprender algo significativo en el proceso. Es muy poco probable que el agente realice aleatoriamente la secuencia de acciones exactas para llegar a un objetivo propuesto sin haber aprendido antes algo, en un modelo como el de la pista N°1 de la Figura 12 puede que con muchas iteraciones el agente logre llegar a la meta ya que es una línea recta y de cierto modo es una tarea sencilla, al no tener que realizar una secuencia de combinaciones entre la dos acciones posibles, pero en los modelos de pista N°2 y N°3 se requiere de una secuencia de acciones más compleja que simplemente ir en una sola dirección.

Lo ideal es elaborar una función de recompensa que sea más precisa, la cual tenga en cuenta más parámetros que solo el valor de una única posición del mini drone, así con recompensas más completas que evalúen distintos parámetros, se podrá guiar a el agente por el camino correcto y de una manera más rápida. Por esta razón también se añadió en la función de recompensa una función que evaluará la posición en Y, que es la posición horizontal, de tal manera que:

$$r = \left(0.5 \cdot e^{(x)} - 1\right) + \left(e^{(-20 \cdot y^2)}\right)$$

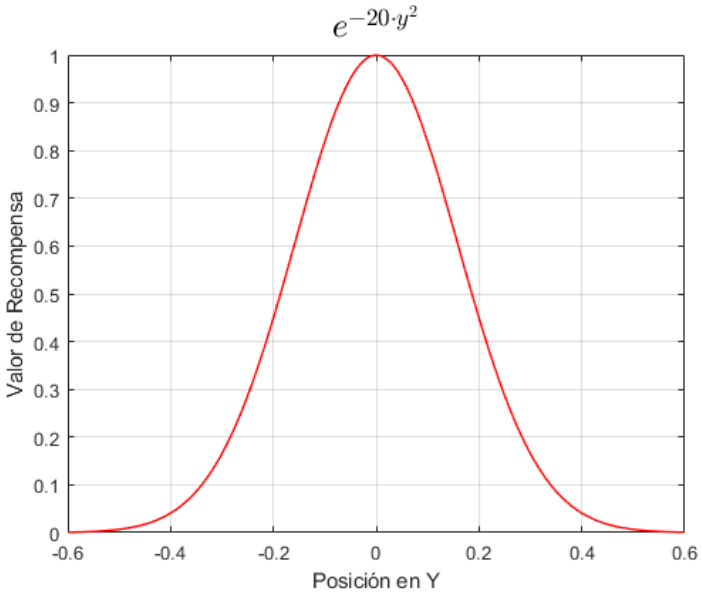


Figura 31 Recompensa Posición Eje Y

Añadiendo esta otra ecuación a la función de recompensa inicial, ya no solo se evaluará la posición en X sino también en Y. Como el objetivo del agente para este primer modelo de pista, es solo moverse en el eje vertical, se recompensará a el agente si se mantiene en la posición inicial de 0 m en el eje Y, si se gráfica solamente la última parte que se añadió a la función se obtiene el gráfico de la Figura 31, allí se observa que, si el mini drone se aleja de la posición de 0 m en el eje horizontal, dejará de recibir recompensa y entre más se aleje menos recompensa irá recibiendo, en cambio mientras más centrado se encuentre al eje vertical recibirá más recompensa.

De momento se creería que con solo estos dos parámetros en la función de recompensa ésta ya estaría completada, pero la correcta creación de una función de recompensa es posiblemente una de las tareas más difíciles del aprendizaje por refuerzo. Ya que realmente no se sabe si una función de recompensa está mal diseñada, hasta después de haber pasado mucho tiempo entrenando a el agente y este no haya podido alcanzar los resultados esperados.

Siguiendo con la elaboración de la función de recompensa, hay que tener en cuenta que el modelo de Simulink tiene un bloque específico para el agente (controller), como muestra la Figura 16, la salida de este bloque son las acciones del agente, pero a la vez este requiere tres propiedades de información del entorno: las observaciones, la recompensa y si se ha alcanzado un estado donde se deba terminar el episodio (isdone). El episodio es una simulación del agente en el entorno, donde se podrá comprobar el comportamiento de la inteligencia artificial, el episodio se puede configurar para que dure un tiempo máximo determinado y cuando se realiza el periodo de entrenamiento uno de los parámetros a tener más en cuenta, es el número de episodios que se van a realizar durante una sola sesión de entrenamiento, (se profundizará más en la Sección 3.4 de entrenamiento). En el caso del mini drone el episodio termina si este alcanza la meta o si se desvía demasiado de la trayectoria a seguir, los cuales se determinan a partir del estado observable del mini drone. Por lo tanto, el valor numérico de la recompensa depende del estado en él se encuentre y de si se alcanzó alguna de las condiciones para que se termine el episodio. Para que el modelo sepa cuándo se debe terminar el episodio, el bloque Reward Function de la Figura 16, representa una función que evalúa tanto los estados observados como los sensores virtuales de posición, que se mencionaron en la Sección 3.1.2 Procesamiento de Imágenes de la Cámara, para dar como resultado el valor de la recompensa además de si el episodio tuvo que ser finalizado.

Específicamente en este bloque que representa la función `rewardfun`, se van a determinar dos variables que representarán cuando el episodio deba de finalizar, por un lado, se crea la variable `madeit` la cual indica que el mini drone ha llegado a la meta, y por otro lado se tiene la variable `collied` que indica cuando el mini drone se ha alejado demasiado de la trayectoria. Estas dos variables son booleanes, así que cuando alguna de las dos pasa a un estado verdadero, se da a entender que el episodio debe finalizar por medio de la variable `isdone`, que es la que pasa por el bloque del agente y siempre va a estar evaluando a `collied` y `madeit`.

Para entender más a profundidad cómo se realiza lo antes mencionado, se enseña el código del bloque de función `rewardfun` en la Figura 32, allí se observa que para tener almacenada la posición en una variable se debe especificar, cuál de las seis observaciones corresponde a la posición en X y la posición en Y, como las observaciones son un elemento de dimensiones [6 1] se elige numéricamente que observación le corresponde a cada posición. Por ende, la observación número uno es la posición en X y la observación número dos es la posición en Y, esto se puede saber fácilmente viendo el modelo de Simulink de la Figura 16, allí las salidas del bloque del estimador de estados están nombradas en ese mismo orden. Posteriormente en el código está declarada la variable `madeit`, esta se activará o pasará a un estado verdadero cuando la posición en X sea mayor o igual a 1.5 m, esto significara que el mini drone alcanzó la meta, pero por otro lado, se tiene la variable `collied`, que se activará cuando la posición en X sea menor a -0.4 m o cuando la posición en Y sea menor a -0.4 m o mayor a 0.4 m, si se tiene en cuenta las dimensiones de la pista número 1, estos valores corresponderían a posiciones que ya estarían demasiado alejadas de la trayectoria y por lo tanto significaría que el episodio debe de finalizar.

```
1   function [reward,isdone] = rewardfun(Sensor_A,Sensor_E,observation)
2
3       x=observation(1);
4       y=observation(2);
5
6       madeit =(x >= 1.5);
7
8       collied=(x < -0.4) || (y < -0.4) || (y > 0.4);
9
10      reward = rewardfun (Sensor_A,Sensor_E,observation,madeit,collied);
11
12
13      isdone = collied || madeit;
```

Figura 32 Función rewardfun N°1

En la línea 13 del código anterior estaría creada la variable `isdone` que necesita el agente, la cual se activará cuando `madeit o collied` pasen a un estado verdadero. Como se enseñó anteriormente las recompensas de la posición son discretas, ya que independientemente del estado en el que se encuentre el mini drone este siempre tendrá una recompensa, no importa si este no llega a la meta el siempre tendrá un valor de recompensa de vuelta. Pero para mejorar la función de recompensa cuando los episodios terminen, la recompensa debe ser lo suficientemente grande como para compensar las posibles recompensas mínimas o máximas que podrían lograrse si el episodio no hubiera terminado. Por ejemplo, si el mini drone se queda estático en una posición de la trayectoria la cual le genere una recompensa positiva, este aprenderá que es una buena acción, pero no lo es ya que no estaría logrando el objetivo que es llegar a la meta. Para tratar de corregir este problema, se debe de dar una recompensa lo suficientemente grande cuando el mini drone llegue a la meta, lo ideal es que el valor de la recompensa sea diferenciable en gran medida cuando un episodio llega a la meta y cuando no. También se debe hacer

los mismo cuando el mini drone se desvía de la trayectoria, pero dando una penalización de gran magnitud, así el valor de la función de recompensa será lo bastante bajo para hacerle entender a él algoritmo de entrenamiento, que las acciones realizadas por el agente no son las más óptimas sí que quiere lograr el objetivo.

En el código de la Figura 32 se puede observar que en la línea 10 se encuentra la función que se encarga de realizar el cálculo numérico para obtener el valor de la recompensa, cuando se ejecuta esta línea se llama a la función que se enseña en la Figura 33, por lo general dicha función está guardada en otro Script de Matlab distinto, pero debe estar dentro de la misma carpeta que tenga el modelo de Simulink Minidrone Parrot Competition.

```
1  function r = rewardfun (Sensor_A,Sensor_E,state,madeit,collied)
2
3     x = state(1);
4     y = state(2);
5
6     r = (0.5*exp(x)-1) + exp(-20*y^2) - (1.5-x) - 0.1 * Sensor_A + 0.1*Sensor_E - 2*collied + 5*madeit;
```

Figura 33 Función de Recompensa Pista N°1

Finalmente, en la Figura 33 se puede visualizar la función de recompensa final que se utilizó para el primer modelo de pista, allí la salida es `r` que es el valor numérico de la recompensa y como entradas se tiene los sensores virtuales de posición (Sensor_A y Sensor_E), los dos estados de posición y las dos señales de finalización `madeit` y `collied`.

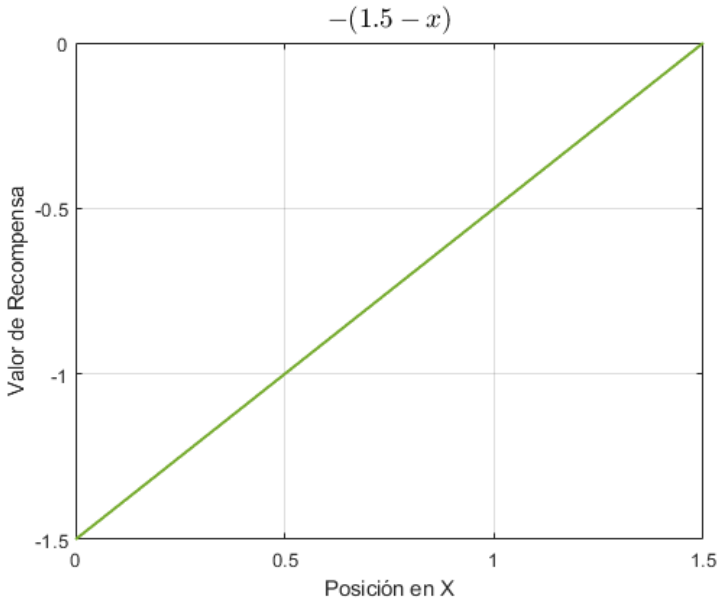


Figura 34 Penalización Posición Eje X

En los dos primeros términos de la función se encuentran las ecuaciones que se mencionaron anteriormente ($0.5 \cdot e^{(x)} - 1$) y ($e^{(-20 \cdot y^2)}$), después para mejorar la función de recompensa se añade el término $-(1.5 - x)$, con esto lo que se busca es añadir una

penalización cuando la distancia del mini drone a la meta es muy grande, de esta forma se trata de incentivar a el agente para que avance lo más rápido posible en la posición vertical, por ejemplo, en la Figura 34 se observa que si la posición en X es cercana a 1.5 m que es donde se encuentra la meta, la penalización será menor en comparación a cuando la posición en X sea cercana a 0 m, que es el punto de inicio.

Posteriormente se hace uso de los sensores virtuales de posición para elaborar una función de recompensa más completa, como se mencionó en secciones anteriores los dos sensores se encargan de realizar la suma de píxeles blancos, que son los que indican la trayectoria que debe seguir el mini drone, en un área determinada de la imagen que tiene incorporada el mini drone. El Sensor_A se encargará de monitorear la sección central/vertical de la cámara, como se enseña en la Figura 14; uno de los objetivos del proyecto es que el mini drone siga la trayectoria de una forma precisa, esto significa que en el primer modelo de pista al tener que seguir la línea recta, la sección central/vertical de la cámara siempre debe de estar centrada con la línea de color rojo. Por ende, se espera que haya constantemente píxeles rojos (con el sistema de procesamiento de imagen pasaran a ser píxeles blancos) en esta área y cuando se realice la sumatoria de la cantidad de los mismos debe ser de por lo menos de 1140 píxeles, si este valor llega a ser menor significara que el mini drone se desvió de la trayectoria, ya que no se encuentran presentes los píxeles correspondientes a la línea roja. En el subsistema "*Image Processing System*" de la Figura 11, se observa que cuando la suma de la cantidad de píxeles blancos es menor a 1140 se detecta el umbral de la Function 1 y por lo tanto se activa el Sensor A, esto significa que debe de haber una penalización cuando este sensor se encienda. Como se detalla en la función de recompensa de la Figura 33, el término $(-0.1 * Sensor\_A)$ hace referencia a lo antes mencionado, si este sensor se activa se penalizará a el agente con un valor de -0.1.

Por otro lado, el Sensor_E funciona de la misma manera solo que se encarga de monitorear otra zona de la cámara, más específicamente la zona central, como se enseña en la siguiente imagen:

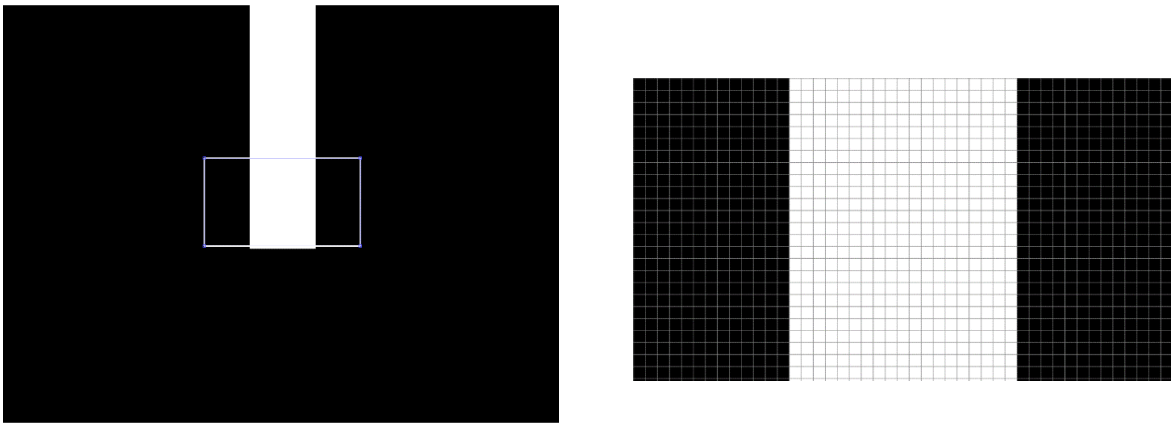


Figura 35 Selección Área Sensor Virtual E

Para delimitar esta zona se utilizó la Function 5, que es la que se muestra a continuación:

```
function Sensor_E = fcn(Umbral)
Umbral=Umbral(45:69,59:103);
Sensor_E = Umbral;
```

Como se puede observar este sensor tendrá en cuenta una pequeña área central de la imagen que esté proporcionando la cámara, de tal modo que, si dentro de esta sección se encuentra una cantidad de por lo menos 400 píxeles blancos, significará que el mini drone está centrado con la trayectoria y el Sensor_E se encenderá. Cuando esto suceda se debe premiar a el agente, ya que está realizando las acciones adecuadas para que el mini drone se mantenga alineado con la trayectoria. Por esto en la función de recompensa está el término $(+\ 0.1 * Sensor\_E)$, así cuando el sensor virtual de posición E este encendido se le dará una recompensa positiva de $+0.1$.

Por último, se le debe dar una recompensa positiva y negativa a los estados `madeit` y `collied`, los cuales son los estados que representan condiciones para que se termine el episodio. Cuando se enciende `madeit` significa que el mini drone logró llegar a la meta, por lo que se le debe recompensar positivamente, en la función de recompensa se observa que cuando esto sucede se recompensa a el agente con un $+5$. Pero cuando se activa el estado `collied` se da a entender que el mini drone se alejó de los límites propuestos y desvió de la trayectoria, por lo que se penalizara a el agente con un -2 como muestra la función de recompensa en el término $(-\ 2 * collied)$. Con esto estaría finalizada la función de recompensa para el primer modelo de pista, cabe aclarar que esta no fue la función inicial que se había planteado, como se mencionó antes la función de recompensa es una de las tareas más complicadas de elaborar en el aprendizaje por refuerzo, ya que se requiere de varios periodos de entrenamiento para saber si la función de recompensa es adecuada y precisa para alcanzar el objetivo que se pretende cumplir.

Por ejemplo, uno de los factores a tener más en cuenta es el equilibrio entre las recompensas positivas y las penalizaciones, si estas están de cierto modo desbalanceadas se tendrá el caso de que se recompensará demasiado a el agente por todas las acciones que decida, o por el contrario se tendrá el caso que penalizará cualquier acción que realice el agente a pesar de ser medianamente buena. A demás se pensaría que los valores de recompensa y penalización de los sensores virtuales de posición A y E son muy bajos, considerando los demás valores de la función de recompensa, pero teniendo en cuenta lo que se mencionó anteriormente, si se llega a aumentar estos valores la función de recompensa cambiaria y ya no sería tan efectiva. Todos estos parámetros son tenidos en cuenta cuando se ajusta la función de recompensa, que después de todo es el resultado de un proceso extenso de prueba y error.

3.3.2 Función de Recompensa Segundo Modelo de Pista

El segundo modelo de pista es el que se enseña en la Figura 36, allí se debe seguir una trayectoria en forma de “L” donde el mini drone primero debe de recorrer una sección en línea recta vertical por 1 m, y posteriormente debe desplazarse de manera horizontal

por aproximadamente 0.6 m para así llegar a la meta. En esta ocasión el agente debe aprender a cómo detenerse en una sección determinada para después cambiar de dirección, esto lleva consigo a que el agente tenga que estudiar de una forma de prueba y error las dinámicas del sistema del mini drone.

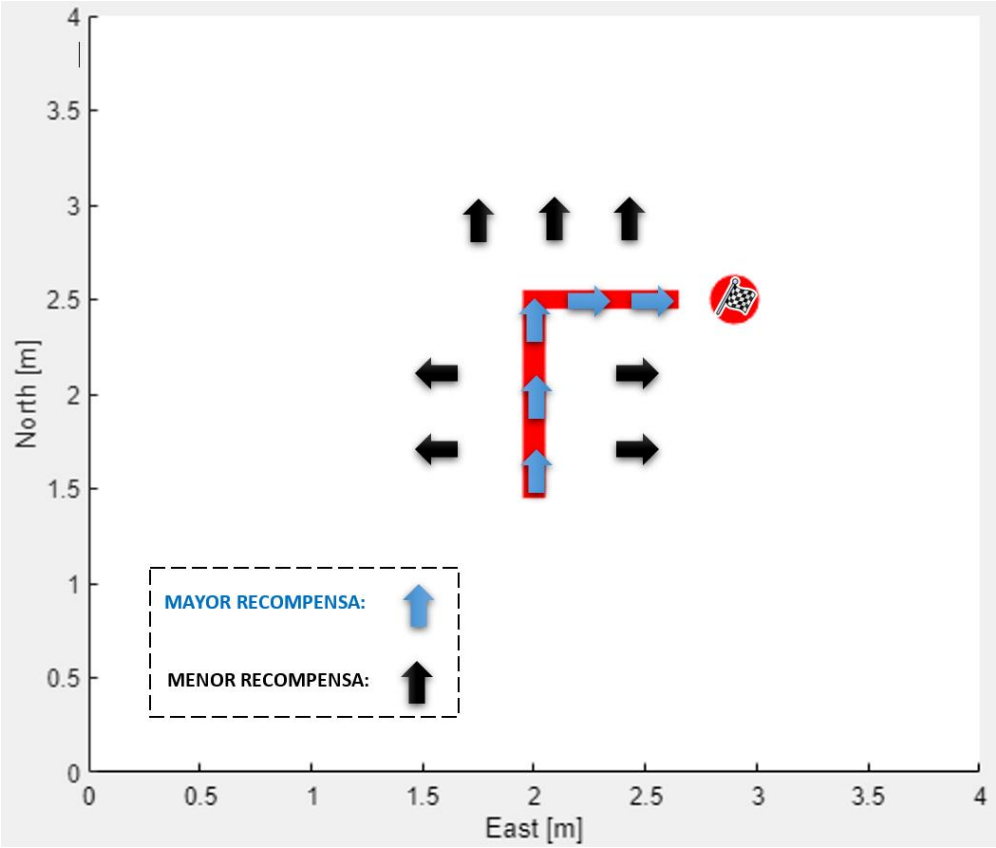


Figura 36 Recompensa Pista N°2

Primero se tendrá que elaborar la función `rewardfun` para este modelo, que es la que se indica en la Figura 37, aquí lo principal es dejar estipuladas las variables `madeit` y `collied`, las cuales son condiciones de estado para que el episodio finalice. Primero para que el estado `madeit` suceda el mini drone debe de llegar a la meta, por ende, la posición de éste debe ser de 1 m en el eje X y 0.6 m en el eje Y, si se observa la línea seis del código la variable `madeit` está concretada con estos dos valores, solo que en el posición vertical (eje X) se tiene el condicional $(x > 0.9)$, no está estipulado con 1 m simplemente por darle un poco de margen a el agente cuando este en el periodo de entrenamiento, además junto con el condicional $(y \geq 0.6)$ se especifica que por lo menos la posición del mini drone para que se produzca el estado `madeit` debe ser de por lo menos estos dos valores. Cabe aclarar que los caracteres (&&) corresponden a la función lógica AND y los caracteres (||) corresponden a la función lógica OR.

```
1  function [reward,isdone] = reward(Sensor_E,obsevaciones)
2
3      x = obsevaciones(1);
4      y = obsevaciones(2);
5
6      madeit = (x > 0.9) && (y >= 0.6);
7
8      collied = ((x <= 0.9) && (y >= 0.1)) || (x <= -0.1) || (y <= -0.04) || (x >= 1.1);
9
10     reward = rewardfun (Sensor_E,observation,madeit,collied);
11
12     isdone = collied || madeit;
```

Figura 37 Función rewardfun N°2

Posteriormente se elabora la variable de terminación `collied`, esta es un poco más extensa que la anterior ya que este modelo de pista en forma de "L" añade algo más de complejidad al diseño. Se puede observar en la línea 8 del código que la variable está compuesta de cuatro partes:

- $(x \leq 0.9)$ && $(y \geq 0.1)$: Parte inferior derecha de la pista.
- $(x \leq -0.1)$: Parte inferior de la pista.
- $(y \leq -0.04)$: Parte izquierda de la pista.
- $(x \geq 1.1)$: Parte superior de la pista.

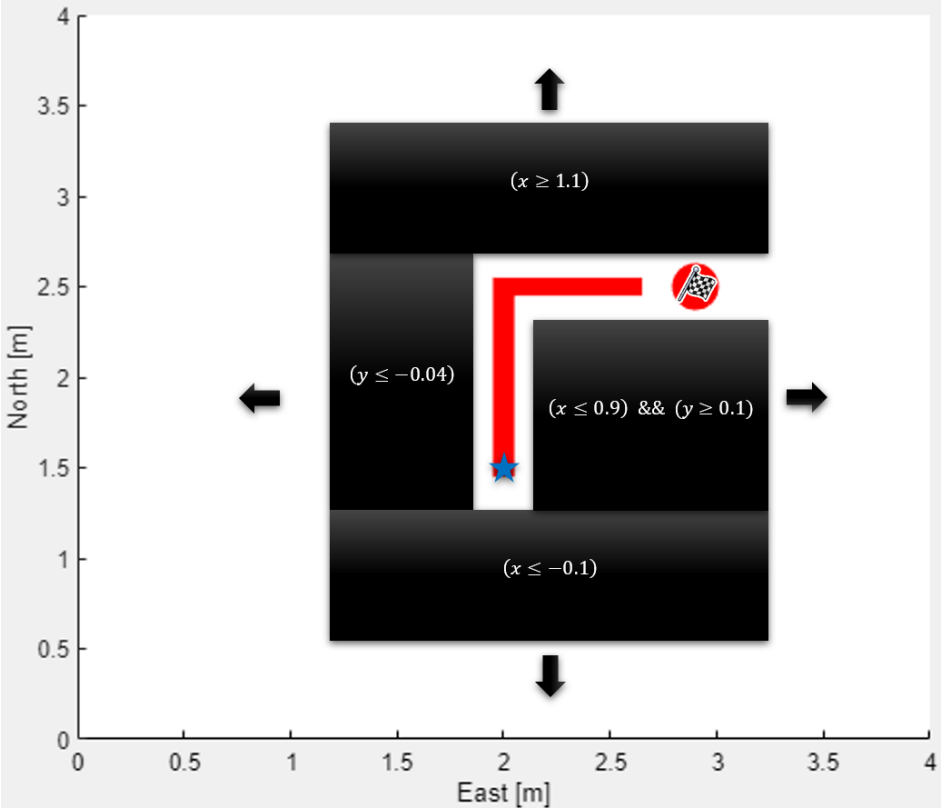


Figura 38 Límites de Posición Modelo N°2

Todas estas secciones corresponden a las posiciones en las cuales no debe estar el mini drone, en el periodo de entrenamiento cuando el agente por cualquier razón aleatoria opte por dirigirse a estos sitios para explorar el entorno, con la ayuda de la función de recompensa y la función `collied` será informado que esta no es una buena ruta a seguir para llegar a la meta. Estas secciones de forma aproximada se observan de manera gráfica en la Figura 38, cabe aclara que la posición inicial del mini drone (indicada con una estrella), es el 0,0 de la pista y que las secciones de color negro están establecidas según este marco de referencia, además como indican las flechas de color negro estas áreas se extienden hasta el infinito, solo dejando libre la sección por la cual el mini drone debe desplazarse. Finalmente, con la elaboración de las variables `madeit` y `colllied` se puede realizar la variable `isdone`, de tal modo que $(isdone = collied \,||\, madeit)$, cuando alguna de las dos variables terminales se activa, `isdone` pasará a ser verdadero indicándole a el agente que el episodio debe terminar.

En el código de la Figura 37 se puede observar que en la línea 10 se encuentra la función que se encarga de realizar el cálculo numérico para obtener el valor de la recompensa, cuando se ejecuta esta línea se llama a la función que se enseña en la Figura 39, se puede observar que uno de los cambios con respecto al modelo anterior es que el Sensor_A no es utilizado, como el área para la cual fue diseñado abarca la sección central/vertical de la cámara no sería buena idea implementarlo, ya que cuando el mini dron este desplazándose por decirlo de algún modo en la segunda parte de la pista en “L” este se moverá horizontalmente hasta la meta, pero sin rotar el mismo mini drone, ni tampoco el ángulo de la cámara, entonces el Sensor_A ya no cumpliría una buena función en este caso, ya que al estar diseñado para una línea recta vertical, no sería óptimo utilizarlo cuando se cambie el sentido de dirección y la línea recta sea horizontal. Por esta razón solo se utiliza el sensor virtual de posición E, que siempre va estar monitoreando la sección central de la imagen de la cámara y funcionará tanto para los movimientos que se efectúen en el eje vertical, como para los que se realicen en el eje horizontal.

```
1  function r = rewardfun (Sensor_E,state,madeit,collied)
2
3  x = state(1);
4  y = state(2);
5
6  r = 1.2*(exp(x)-1) + 1.2*(exp(y)-1) - 1*(1-x) - 1*(0.6-y) + 1*Sensor_E + 350*madeit - 20*collied;
```

Figura 39 Función de Recompensa Pista N°1

En la función de recompensa se tiene el primer término $(1.2 \cdot (e^{(x)} - 1))$ que corresponde a los movimientos en X, como se ilustra en la Figura 40 a medida que el valor de la posición aumenta en el eje vertical el valor de la recompensa también aumenta de manera exponencial, como la idea es que en esta dirección el mini drone avance la distancia de un 1 m, cuando sobrepase este valor la variable de terminación `collied` se activará y el episodio terminara, además de penalizar en la función de recompensa a el agente por pasar del límite, que específicamente es de 1.1 m, esto con el fin de que el agente tenga siempre algo de margen en el periodo de entrenamiento.

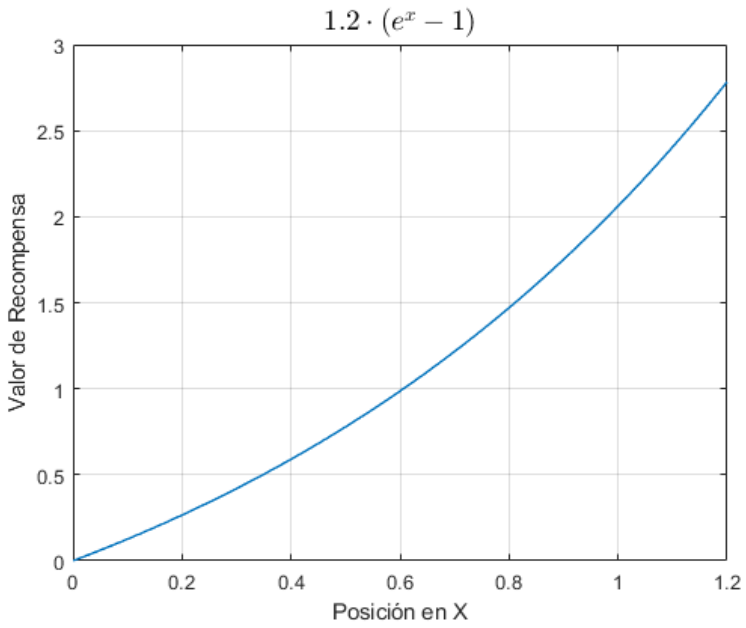


Figura 40 Recompensa Posición Eje X Modelo N°2

El siguiente término $(1.2 \cdot (e^{(y)} - 1))$ se encarga de evaluar la recompensa para los movimientos en el eje Y o eje horizontal, como se puede apreciar es la misma ecuación para los movimientos en X, así que el valor de la recompensa es el mismo de la Figura 40, siempre y cuando el valor de la posición en Y aumente el valor de la recompensa también lo hará, solo que el agente tiene que aprender cuándo moverse en el eje vertical y cuando en el eje horizontal, para lograr la mayor de cantidad de recompensa posible.

Después de añadir dos términos que suman recompensas positivas, se agregan dos términos que penalizan a el agente si la distancia de la meta con respecto al mini drone es bastante grande, para esto se tienen los términos $(-1 \cdot (1 - x))$ y $(-1 \cdot (0.6 - y))$ en la función de recompensa de este segundo modelo de pista.

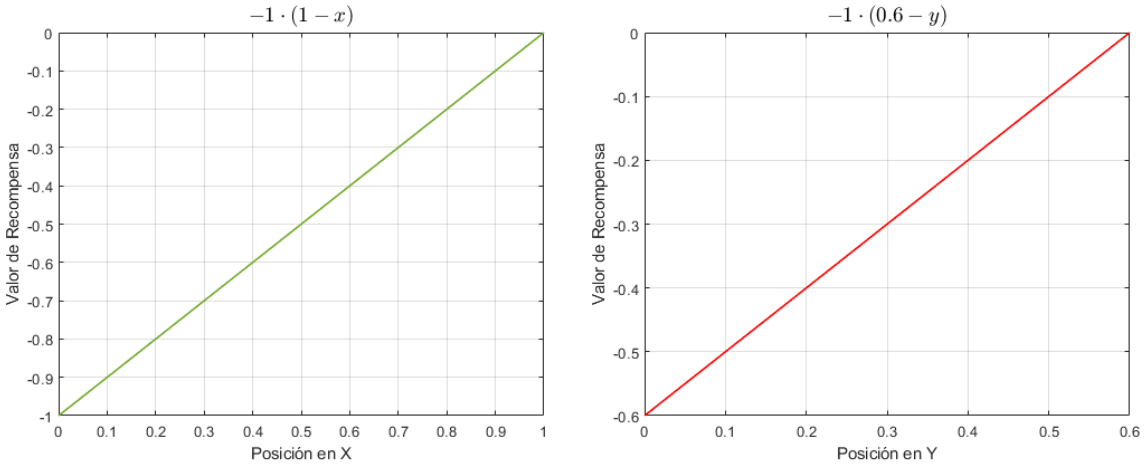


Figura 41 Recompensa de Posición en X y Y

A medida que la distancia entre la meta y el mini drone es más grande el valor de la penalización aumenta, como se puede ver en las gráficas de la Figura 41, del mismo modo el agente tiene que encontrar la manera de tener la menor penalización posible por esto debe encontrar y aprender las acciones adecuadas para cumplir el objetivo de llegar a la meta.

Posteriormente en la función de recompensa se utiliza el Sensor_E, lo que significa que, si el dron está centrado dentro de la línea de la trayectoria roja, la imagen de la cámara también lo estará y por lo tanto la sección que se encarga de monitorear el sensor virtual tendrá a la vista píxeles color blanco. Si la cantidad de píxeles es la adecuada (400) se sabrá que el mini drone está centrado en la trayectoria y tendrá que ser recompensado positivamente, por esto en la función de recompensa se encuentra el término $(+1 \cdot (Sensor\_E))$, lo que indica que cuando esté condición suceda en el valor de recompensa se sumará un $+1$.

Por último, se le debe dar una recompensa positiva y negativa a los estados `madeit` y `collied`, los cuales son los estados que representan condiciones para que se termine el episodio. Cuando se enciende `madeit` significa que el mini drone logró llegar a la meta, por lo que se le debe recompensar positivamente, en la función de recompensa se observa que cuando esto sucede se recompensa a el agente con un $+350$. Pero cuando se activa el estado `collied` se da a entender que el mini drone sobrepaso los limites propuesto, por lo que se penalizara a el agente con un -20, como muestra la función de recompensa en el término $(-20 * collied)$. Como se puede observar los valores de recompensa y penalización de estas dos variables cambiaron con respecto al modelo N°1, ya que como se mencionó antes la función de recompensa está en constante cambio hasta encontrar una que cumpla con el objetivo, y en el proceso de ajuste se definió que estos valores eran los adecuados para tener el equilibrio entre recompensas positivas y negativas, junto a la recompensa que se provee cuando las variables de terminación son activadas.

El siguiente paso es comenzar con la etapa de entrenamiento, una vez que la red neuronal que representa a el agente está concretada y las funciones de recompensa están establecidas, es momento de que la inteligencia artificial adquiera el conocimiento del entorno por medio del entrenamiento, allí tomará acciones y explorará diversos estados del ambiente, donde se le retornara un valor en función de que tan buena fue la serie de acciones que realizo y junto al algoritmo de entrenamiento se modificara la política de seguimiento del agente que la compone, estos periodos de entrenamiento se enseñan más a profundidad en la siguiente sección.

3.4 Entrenamiento del Agente y de la Política de Seguimiento

En esta sección se profundizará en todo lo que conlleva el proceso de entrenamiento de una inteligencia artificial por medio del aprendizaje por refuerzo, en esta etapa el agente recibe recompensas del entorno y estas recompensas se utilizan para actualizar la política de seguimiento, donde dicha política está basada en redes neuronales como se mencionó en secciones anteriores. El algoritmo de entrenamiento es el encargado de ajustar y modificar los parámetros del actor y del crítico, según las recompensas que vaya obteniendo el agente.

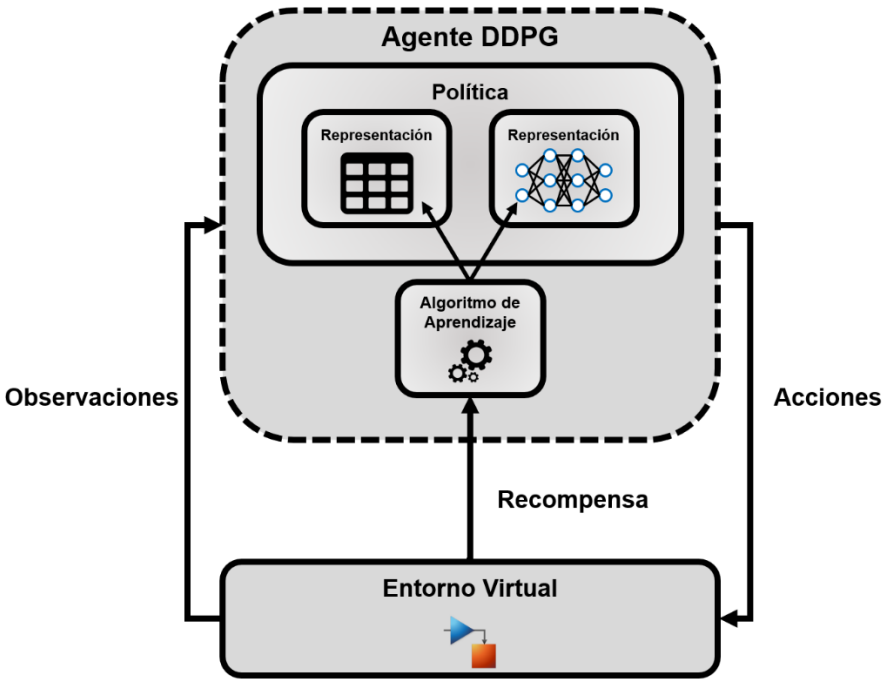


Figura 42 Diagrama Proceso de Entrenamiento

Para comenzar el proceso de entrenamiento se debe utilizar la función `train`, este comando se encargará de inicializar todo el modelo y cargar las variables necesarias como el entorno y el agente. El modo de utilizar la función es la siguiente:

```
info = train(agent,env);
```

A diferencia de muchas funciones de Matlab, la función `train` no devuelve el agente modificado como salida en otra variable, en cambio, actualiza directamente la variable del agente a medida que avanza el entrenamiento. La salida de `train` es una estructura de datos que contiene el progreso del entrenamiento, esto será guardado en la variable `info`. Los parámetros que necesita esta función son `agent` que corresponde a el agente creado en la sección 3.2, junto a la variable que define el entorno virtual `env`.

Pero como es costumbre, la mayoría de funciones del Toolbox de aprendizaje por refuerzo de Matlab, tienen la posibilidad de ajustar diversas opciones, y la función `train` es una de ellas. Por ejemplo, de forma predeterminada el entrenamiento se detiene si el número de episodios llega a ser igual a 500, pero en ciertos casos puede que este valor de episodios sean muy pocos para que un agente pueda aprender algo significativo. Para modificar este tipo de opciones se cuenta con la función `rlTrainingOptions`, el cual permite modificar una serie de parámetros que son útiles y que se deben de tener en cuenta antes de iniciar el proceso de entrenamiento. Teniendo así que con esta función y el parámetro `'MaxEpisodes'`, se podrá cambiar la cantidad de episodios para una sesión de entrenamiento, otra opción que se puede modificar es la de `'ScoreAveragingWindowLength'`, o longitud de la ventana para promediar las recompensas, cuando el agente está en el periodo de entrenamiento por lo general se suele enseñar cual es la recompensa promedio que está obteniendo el agente según los últimos 5 episodios, este valor es útil para tener una aproximación de que tan bien está aprendiendo el agente, pero si se desea saber el valor promedio de la recompensa para un mayor cantidad de episodios se debe modificar el valor por defecto con esta opción.

También existen opciones para detener el entrenamiento antes de que se complete la cantidad máxima de episodios establecidos, el método más sencillo es utilizar el botón “Stop Training” que aparece en la ventana de progreso, pero también se puede detener el entramiento cuando se alcanza un criterio de éxito, por ejemplo, se puede configurar que el entrenamiento concluya cuando la recompensa promedio alcance un valor de umbral determinado, esto también se realizaría con la función `rlTrainingOptions` y el parámetro `StopTrainingCriteria','AverageReward"`. Todas estas opciones modificables serán de utilidad para inicializar el entrenamiento en los dos modelos de pista.

3.4.1 Entrenamiento del Primer Modelo de Pista

Lo primero que se debe de realizar es ajustar las opciones de entrenamiento, en primera instancia se ajustó la cantidad de episodios que va a tener el entrenamiento a 2000 episodios, lo siguiente es especificar los pasos por episodio. Los pasos de un episodio son la cantidad de acciones que se le permitirán realizar a el agente en único episodio, en este caso fue configurado en 15 pasos, así que esto significa que para el primer modelo el agente deberá llegar a la meta con 15 acciones como máximo, teniendo en cuenta que cada acción será evaluada y efectuada cada un segundo. Si el agente no ha terminado el episodio ya sea porque no llego a la meta (`madeit`) o porque no se desvió lo demasiado de la trayectoria (`collied`) y se superan los 15 pasos establecidos, el episodio finalizara y comenzara uno nuevo, esto es por si el que el agente decide quedarse estático en un punto de la trayectoria y no genera una condición de estado para que el episodio culmine.

Las líneas de código que enseñan cómo se ajustan estos parámetros están a continuación:

```
opts=rlTrainingOptions("MaxEpisodes",2000,"MaxStepsPerEpisode",15)
opts.ScoreAveragingWindowLength=100
info= train(agent,env,opts);
```

El parámetro para establecer la cantidad de pasos por episodio es `"MaxStepsPerEpisode"`, además se puede observar que hay dos formas de configurar las opciones de entrenamiento, el primer método es como se enseña en la primera línea del código, en la que se crea la variable que va a guardar las opciones (`opts`) y la segunda manera es del modo en el que se configuró el parámetro `'ScoreAveragingWindowLength'`, allí lo que se realiza es modificar directamente la variable `opts`. Posteriormente se hace uso de la función `train`, pero debe tener dentro de ella la variable `opts` para que se tengan en cuenta las opciones modificadas.

Con todo lo anterior configurado de la manera correcta ya se puede proceder a iniciar el entrenamiento, para primer modelo de pista el progreso de entrenamiento que se tuvo, es el mostrado a continuación:

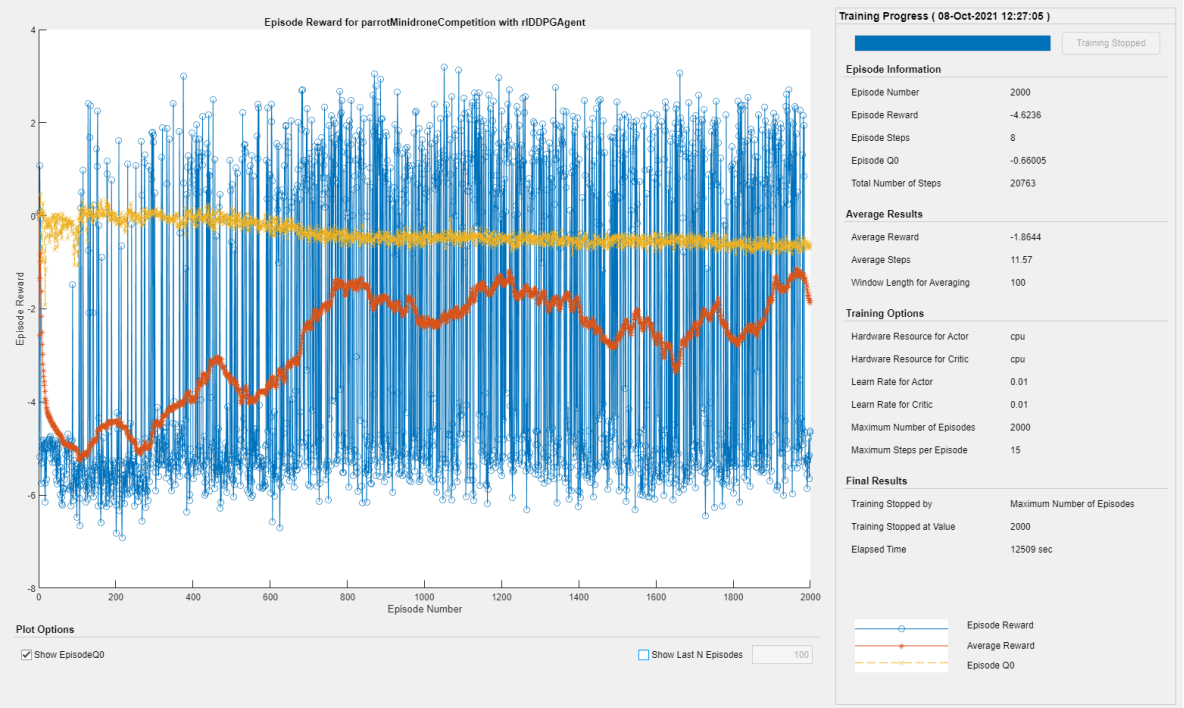


Figura 43 Progreso de Entrenamiento Modelo N°1

Cada vez que se comienza un periodo de entrenamiento el progreso se enseña de forma predeterminada en una nueva ventana, se puede observar en la Figura 43 que al lado izquierdo se ilustra una gráfica que enseña la recompensa obtenida (color azul) para cada episodio realizado. Por lo general se desea que el valor de la recompensa aumente a medida que avanza el entrenamiento, sin embargo, la aleatoriedad en la realización de acciones del agente y su interacción con el entorno dará como resultado una variación en la recompensa recibida por cada episodio simulado. Por lo tanto, el progreso del agente se determina a partir de valor promedio de las recompensas, que es la línea de color naranja que se enseña en la gráfica, según las configuraciones establecidas este valor promedio será calculado con las recompensas de los últimos cien episodios. Como el agente que se construyó en secciones anteriores contiene un crítico, en la gráfica de progreso se mostrará la predicción del crítico para el valor de recompensa da cada

episodio, que estará ilustrado por la línea de color amarillo. Se espera que la predicción del crítico comience a coincidir con la recompensa real a medida que avanza el entrenamiento, esto indicará que el crítico está aprendiendo a evaluar con precisión el valor de recompensa en un estado en particular.

En el lado derecho de la ventana de progreso del entrenamiento se tienen varios datos y parámetros establecidos, como los datos actuales del entrenamiento que son: el número de episodio, la recompensa, los pasos realizados, el valor de estimación del crítico y el número total de pasos que se han realizado hasta ese momento del entrenamiento. También se enseñan los resultados promedios de la recompensa y los pasos, además cuando se finaliza el entrenamiento se calcula cuál fue el tiempo total de entrenamiento, para este caso fueron 12.509 segundos lo que es igual a un tiempo de entrenamiento total de 3.4 horas.

3.4.2 Entrenamiento del Segundo Modelo de Pista

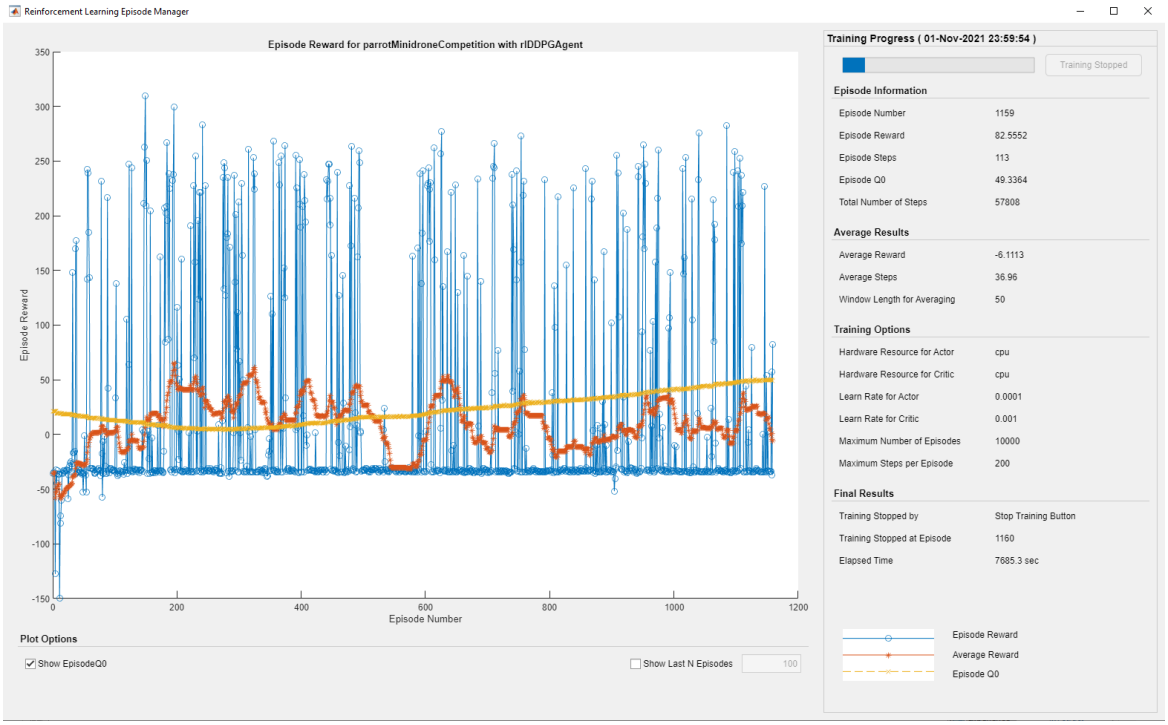
En este caso al tener un modelo de trayectoria que añade más complejidad, las opciones de entrenamiento tuvieron que ser modificadas a las que se muestran a continuación:

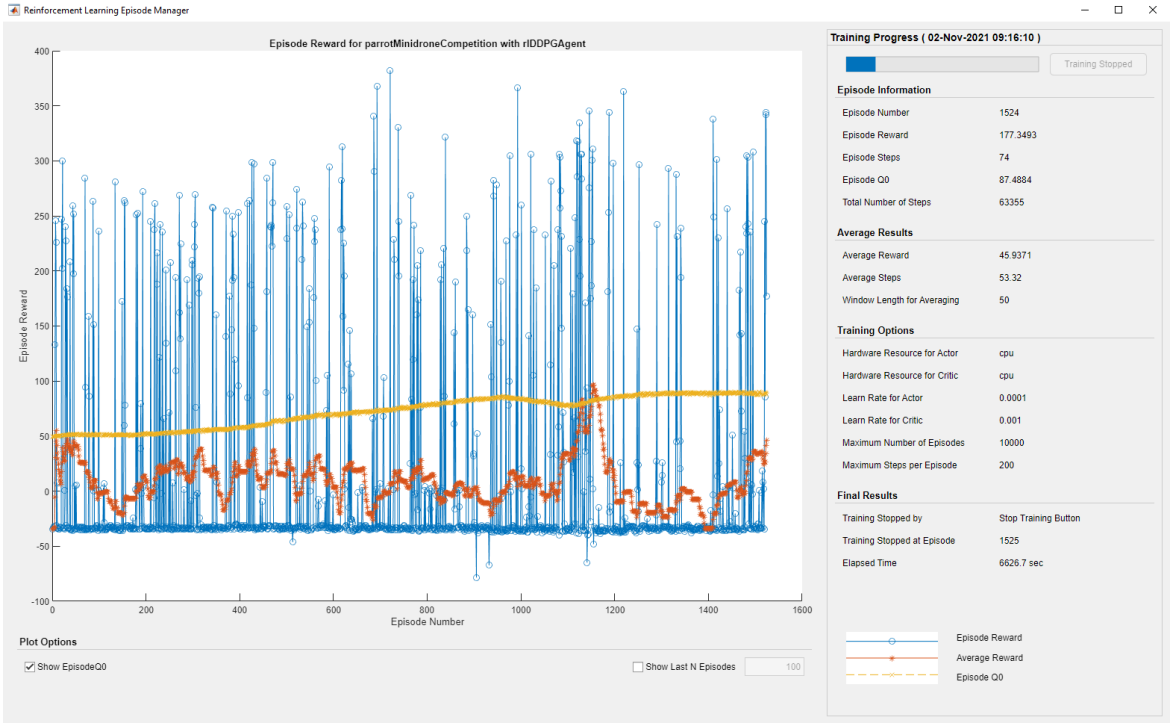
```
opts=rlTrainingOptions("MaxEpisodes",10000,"MaxStepsPerEpiso
de",100);
opts.ScoreAveragingWindowLength=50;
info = train(agent,env,opts);
save(ropts.SaveAgentDirectory + "/finalAgent.mat",'agent')
```

Como se indica en el anterior código el número máximo de episodios se estableció en 10000, además el número de máximo de pasos por episodio es de 100, estos dos valores fueron aumentados con el fin de que el agente explorara mejor el entorno de la trayectoria, teniendo más tiempo disponible por episodio o simulación. El valor promedio de la recompensa será calculado con los últimos 50 episodios del periodo de entrenamiento y además de comenzar este proceso con la función `train`, se hace uso de la última línea del código anterior para guardar siempre el agente entrenado después de que finalice la sesión de entrenamiento, esto se realiza ya que cuando se termina el periodo de entrenamiento el agente final queda en la misma variable `agent`, pero no se guarda en la carpeta del modelo, para después poder volver a utilizar el agente ya entrenado. Si no se utiliza este método, se puede guardar manualmente dando clic derecho en la variable en la que esté almacenado el agente, de lo contrario si no se guarda al momento de cerrar el programa la variable con el agente entrenado se borrara.

Para el segundo modelo de pista se tuvieron que realizar varias sesiones de entrenamiento por razones adversas, pero de igual manera en cada sesión de entrenamiento se utilizó el mismo agente y no uno creado desde cero, esto quiere decir que cuando se finalizaba un

entrenamiento el agente final de ese entrenamiento era el que se seguía entrenando en la siguiente sesión de entrenamiento, las ventanas de progreso que se tuvieron para este segundo modelo son las que se encuentran a continuación:





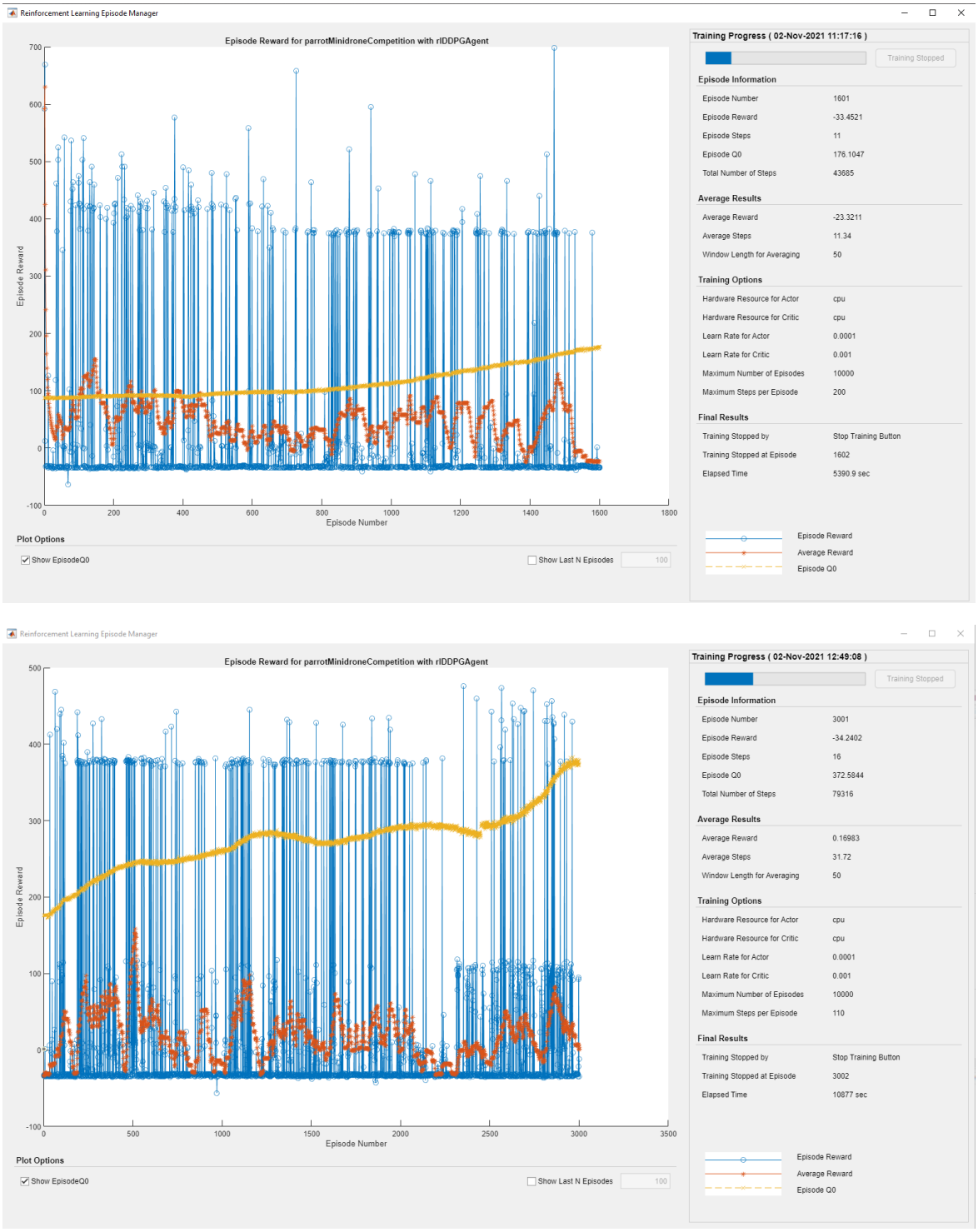


Figura 44 Sesiones de Entrenamiento Modelo N°2

Lastimosamente el entrenamiento del agente fue pausado varias veces y no se puede ver de una manera completa y continua el progreso que fue teniendo éste, de todas maneras, en las distintas gráficas de las sesiones de entrenamiento, se observa como el agente va obteniendo cada vez mayor valor de recompensa. Específicamente en la primera gráfica

el valor máximo que se tenía de recompensa era de aproximadamente 250 a 300, después en las siguientes gráficas se observa como aumenta este valor hasta llegar a un valor promedio de 400 a 500, dando a entender que el agente estaba aprendiendo de una manera adecuada. Si se suma el tiempo total de todas las sesiones de entrenamiento se obtiene un tiempo de 16.268 segundos o 4.5 horas, con un total de 7.285 episodios. Esto sin tener en cuenta todas las sesiones de entrenamiento previas, en las que se cambiaron varios parámetros y se iba ajustando la función de recompensa, para tener el modelo final funcional.

4. Validación de la Inteligencia Artificial

Finalmente queda comprobar que el agente diseñado realizó para cada modelo de pista la función de guiar a el mini drone por la trayectoria estipulada, ya de por sí en las simulaciones realizadas con el agente entrenado se evidencia que el mini drone en los dos modelos llega a la meta. Pero una forma de poder observar la trayectoria exacta que sigue el mini drone, es a través de las observaciones obtenidas a partir del modelo de Simulink, en el subsistema “Path Planning” de la Figura 16 se puede añadir un bloque llamado “To Workspace” en la salida de las observaciones del estimador de estados, este bloque guarda todos los datos de la posición en una variable que será utilizada para obtener solamente la posición en el eje X y en el eje Y, con estos dos datos se podrá graficar las trayectorias que realizó el mini drone en las dos pistas, como enseñan los dos siguientes figuras:

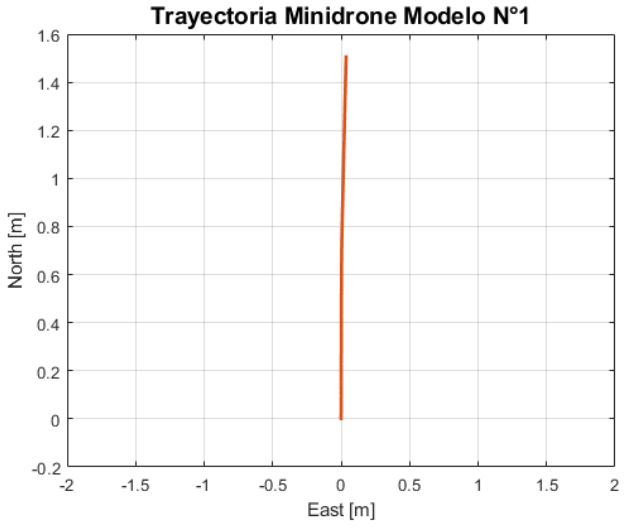


Figura 45 Trayectoria Minidrone Modelo N°1

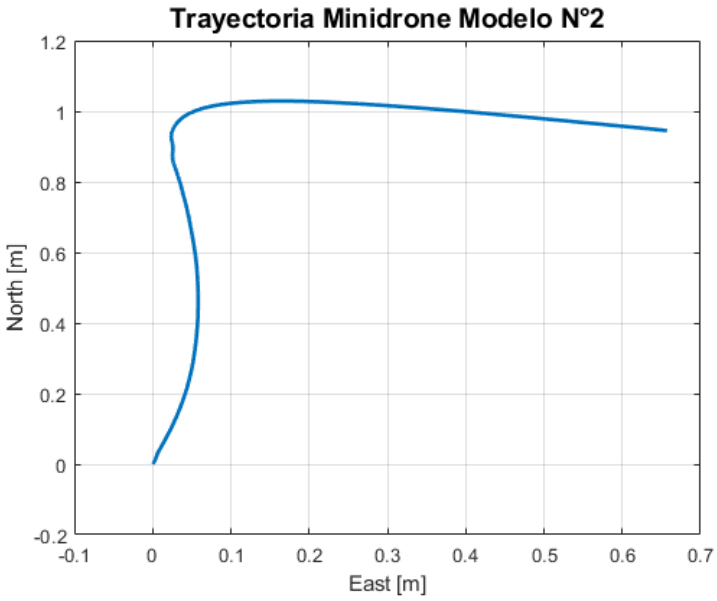


Figura 46 Trayectoria Minidrone Modelo N°2

Para determinar cuantitativamente la eficacia de seguimiento en los dos modelos de pista, se tomaron todos los puntos de la trayectoria realizada por el mini drone y se compararon con los valores de posición de referencia, para posteriormente calcular un porcentaje de error.

Para el primer caso se puede observar en la Figura 45 que la trayectoria realizada es bastante exacta a la esperada, solo que en el tramo final del recorrido el mini drone se desplaza un poco hacia la derecha, específicamente llega a desviarse 0.0363 m o 3.63 cm siendo la posición más alejada a la referencia en todo el recorrido. En general para este primer modelo de pista se obtuvo un porcentaje de error promedio de 0.72 %, teniendo así que la implementación de la inteligencia artificial en este caso es satisfactoria.

En el segundo modelo de pista se puede apreciar que el seguimiento de la trayectoria es un poco menos preciso en comparación al primer modelo, esto debido a que la segunda trayectoria es un tanto más compleja. Específicamente se puede apreciar en la Figura 46 que el mini drone presenta una leve desviación al momento de realizar el primer tramo en el sentido vertical, donde se obtuvo que la posición máxima de desviación en este tramo fue de 0.0582 m o 5.82 cm, en el resto de la trayectoria realizada se presentaron valores de desviación menores a este. Además, si se comparan todos los datos de posición con los de referencia se obtiene que el porcentaje de error promedio para este segundo caso es de 5.4%, con esto se puede concluir que la tarea realizada por parte de la inteligencia artificial es ciertamente adecuada.

El procedimiento para hallar el porcentaje de error del segundo modelo se realizó con las siguientes líneas de código:

```
pos_y = y(1:488,:);
error = pos_y*100;
error_y = mean(error)

pos_x = x(489:781,:);
error = abs(pos_x-1)/1*100;
error_x=mean(error)

error_total=error_y+error_x
```

Allí se observa que el error total es calculado por separado, primero se estima el error en el primer tramo de la trayectoria, que es cuando el mini drone debe desplazarse solo en la dirección vertical (eje X), esto significa que el mini drone no debe tomar valores positivos ni negativos en el eje horizontal (eje Y), por lo que cualquier valor diferente a cero será tomado como error en esta primera sección. Así que simplemente se procede a multiplicar por cien ya que la diferencia del error ya esta medida y después se determina el valor promedio.

Luego se evalúa el error en el segundo tramo de la pista, que es cuando el mini drone debe avanzar en el eje horizontal. Según las especificaciones de la trayectoria establecidas en el modelo de pista N°2, el mini drone cuando se está desplazando horizontalmente debe hacerlo cuando su posición en el eje vertical sea de 1m, de lo contrario estaría dirigiéndose a los límites de la pista. Por lo que si en esta sección la posición es diferente a 1 m se consideraría como un error y este se calcula con la fórmula de porcentaje de error, posteriormente se calcula el promedio de error de cada una las posiciones de este tramo. Finalmente se suman los dos porcentajes de error de cada una de las secciones, para así estimar el error total de seguimiento de la trayectoria N°2.

5. Conclusiones

Se diseñó y elaboró una inteligencia artificial capaz de guiar autónomamente al mini drone Parrot Mambo en un entorno virtual, por medio de los diferentes Toolbox´s con los que cuenta el software Matlab. Específicamente en primera instancia se modificaron los sistemas de control del *Parrot Minidrone Toolbox*, allí se añadieron distintos bloques y funciones que hacen parte de la implementación de inteligencias artificiales entrenadas por medio de aprendizaje por refuerzo. El agente, uno de los elementos incorporados más importantes del modelo, fue adaptado para que pudiera interactuar con el ambiente o entorno virtual diseñado para el mini drone Parrot Mambo.

Por medio de la cámara incorporada con la que cuenta el mini drone en este espacio simulado, se logró obtener una serie de imágenes a tiempo real de la pista, así con una técnica de procesamiento digital de imagen que se encargaba de realizar el conteo de la cantidad de píxeles de color rojo, los cuales hacen referencia a la trayectoria que debía seguir el mini drone, se podía conocer datos de la posición actual con la elaboración de un sistema de sensores virtuales de posición. Con este tipo de información se determinó si el mini drone estaba siguiendo de manera adecuada las trayectorias propuestas, ya que dependiendo del recuento de píxeles rojos se pudo determinar en qué estado se encontraba.

Además, se realizó una clasificación de diferentes tipos de agentes en el ámbito de aprendizaje por refuerzo, en la que se estudió las diversas estructuras por las que puede estar compuesto, allí se determinó cuáles eran los parámetros necesarios para concebir un actor y un crítico, las cuales son estructuras internas que componen el agente y sus representaciones están basadas en redes neuronales. Precisamente se eligió el modelo de agente DDPG (Deep Deterministic Policy Gradient) ya que es el adecuado cuando se desea trabajar en espacios de acción de carácter continuo, como lo es el modelo del mini drone, ya que el agente constantemente tendrá que analizar las observaciones de la trayectoria para determinar y ejecutar las acciones óptimas.

La etapa de entrenamiento fue un proceso exitoso ya que se logró entrenar al agente para cada modelo de trayectoria, en esta etapa fue entrenado repetidamente donde para el primer modelo de pista se pudo concretar el agente tras 3.4 horas de entrenamiento, mientras que para el segundo modelo de pista que era más compleja el tiempo de entrenamiento duro 4.5 horas en total. En consecuencia, el agente a medida que interactuaba con el entorno mejoraba la política de seguimiento, de modo que, independientemente del estado en el que se encontrará siempre tomaría la acción

adecuada. Esto se consiguió realizar gracias al algoritmo de aprendizaje y a la función de recompensa adecuada para cada modelo de trayectoria, que fue resultado de varios diseños puestos a prueba. Asimismo, se comprobó la eficacia de la inteligencia artificial obteniendo los datos de posición, cuando se realiza la simulación del agente entrenado en el entorno virtual, como se enseña en la sección de validación. Allí finalmente se realizan unas estimaciones del error de seguimiento para cada trayectoria, teniendo que para el primer modelo el error fue de 0.72% y para el segundo modelo fue de 5.4%, concluyendo así que la implementación de la inteligencia artificial fue realizada de una manera correcta.

5.1 Consideraciones

En toda la metodología descrita del proyecto cabe aclarar que el proceso de diseño, construcción y entrenamiento de una inteligencia artificial por medio de aprendizaje por refuerzo, es una tarea que requiere de una gran cantidad de tiempo, ya que el desarrollo de cada una de las partes de la I.A se centran en una fase de prueba y error, en la que se deberán analizar qué parámetros son los que no permiten que aprenda el agente, para posteriormente ser modificados y vueltos a poner a prueba. En este caso se optó por entrenar a el agente en dos computadores a la vez con tal de agilizar todo el procedimiento de detección de errores y de poner a prueba nuevos parámetros, con el fin de hallar una inteligencia artificial que cumpliera con el objetivo de guiar a el mini drone.

También uno de los factores a tener en cuenta en la fase de entrenamiento es el requerimiento computacional, si el hardware con el que se cuenta no es muy adecuado el tiempo de entrenamiento podría demorarse mucho más de los esperado. A pesar de que se contaba con un hardware medianamente apto para la aplicación, los tiempos de entrenamiento se extendían por varias horas, donde en muchas ocasiones al finalizar la sesión de entrenamiento no se conseguían buenos resultados. Para esto se recomienda además de paciencia, tener un buen conocimiento de absolutamente todas las partes implicadas a la hora de abordar un proyecto de este tipo.

El agente concretado para cada modelo de pista puede ser mejorado para que al momento de realizar las trayectorias las efectué de manera más precisa, ya que como se pudo observar en la validación del modelo de pista N°2 el mini drone no se desplaza de una manera completamente recta, sino que al principio realiza una semi curva, aunque de todas maneras no logra salirse de los límites establecidos y consigue llegar a la meta. Esta mejora puede ser realizada probablemente con más periodos de entrenamiento, o también se puede aumentar la estructura de la red neuronal añadiendo más capas ocultas y neuronas, así se tendrá un agente más apto para realizar con precisión acciones adecuadas. También se deja planteado realizar todo el proceso de creación de una inteligencia artificial que consiga guiar a el mini drone, por la trayectoria del modelo de pista N°3 que es el que se enseña en la Figura 21 de la Sección 3.1.4.

6. Bibliografía

[1] Kerle N., Nex F., Gerke M., Duarte D. and Vetrivel A., «UAV-Based Structural Damage Mapping: A Review,» de *ISPRS International Journal of Geo-Information*, 2020.

[2] S. V. Sibanyoni, D. T. Ramotsoela, B. J. Silva and G. P. Hancke, «A 2-D Acoustic Source Localization System for Drones in Search and Rescue Missions,» IEEE Sensors Journal, 2019.

[3] P. Worakuldumrongdej, T. Maneewam and A. Ruangwiset, «Rice Seed Sowing Drone for Agriculture,» de *19th International Conference on Control, Automation and Systems (ICCAS)*, Korea, 2019.

[4] Y. Guo, «A Drone-Based Sensing System to Support Satellite Image Analysis for Rice Farm Mapping,» de *IEEE International Geoscience and Remote Sensing Symposium*, Japón, 2019.

[5] S. Jang and N. -S. Park, «Limit Action Space to Enhance Drone Control with Deep Reinforcement Learning,» de *International Conference on Information and Communication Technology Convergence*, Korea del Sur, 2020.

[6] R. E. Ochoa, «Sistema para la Ejecución de Trayectorias en Drones Aéreos,» Universidad de Guanajuato, Salamanca, 2018.

[7] F. Guevara, A. Reyes y A. Sánchez, «Seguimiento autónomo de personas con un robot aéreo no tripulado,» Universidad Autónoma de Puebla, Puebla, 2017.

[8] J. C. D. Alfonso, «Dron de vuelo autónomo con reconocimiento basado en inteligencia artificial,» Universidad Complutense, Madrid, 2020.

[9] L. Meng, T. Hirayama and S. Oyanagi, «Underwater-Drone With Panoramic Camera for Automatic Fish Recognition Based on Deep Learning,» Japón, 2018.

[10] A. A. Cuenca, «Diseño y construcción un cuadricóptero e implementación de un controlador de estabilidad para la aplicación de un algoritmo de vuelo autónomo,» Universidad de los Andes, Bogotá, 2016.

[11] W. Budiharto, A. A. S. Gunawan, J. S. Suroso, A. Chowanda, A. Patrik and G. Utama, «Fast Object Detection for Quadcopter Drone Using Deep Learning,» de *3rd International Conference on Computer and Communication Systems (ICCCS)*, Japón, 2018.

[12] Luis E. Romero, David F. Pozo and Jorge A. Rosales, «Quadcopter stabilization by using PID controllers,» Escuela Politécnica Nacional, Quito, 2014.

[13] F. Kudlačák and T. Krajčovič, «Error Behaviour in PID Controll Systems with Dynamic Processes,» Slovak University of Technology, Slovakia, 2016.

[14] Farzin M. Khortabi, Maaz Ahmed Khan and Vyacheslav V. Potekhin, «Comparative analysis of applying deep-learning on pid process,» Peter the Great St. Petersburg Polytechnic University, Russia, 2017.

[15] Kangbeom Cheon, Jaehoon Kim, Moussa Hamadache, and Dongik Lee, «On Replacing PID Controller with Deep Learning Controller for DC Motor System,» Kyungpook National University, Daegu, 2015.

[16] Richard S. Sutton and Andrew G. Barto, «Reinforcement Learning: An Introduction,» Cambridge, London , 2015.

[17] L. Rouhiainen, «Inteligencia Artificial,» Alienta, España, 2018.

[18] EASA, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado. [Último acceso: 20 08 2021].

[19] MATLAB, «Reinforcement Learning con MATLAB y Simulink,» [En línea]. Available: https://la.mathworks.com/campaigns/offers/reinforcement-learning-with-matlab-ebook.html.