

# **Plataforma hardware y software para el desarrollo y validación de algoritmos de control de actitud para cuadricópteros**

Miguel Fernández Cortizas

# Índice general

<b>1</b>	<b>Introducción</b>	<b>2</b>
1.1	Motivación . . . . .	2
1.2	Solución propuesta . . . . .	3
1.3	Objetivos . . . . .	4
<b>2</b>	<b>Estado del arte</b>	<b>6</b>
2.1	Plataformas de control de cuadricópteros . . . . .	6
2.1.1	Plataformas tipo gimbal . . . . .	6
2.1.2	Plataformas con unión esférica . . . . .	7
2.2	Aprendizaje por refuerzo . . . . .	9
<b>3</b>	<b>Fundamentos teóricos</b>	<b>10</b>
3.1	Controlador PID . . . . .	10
3.2	Redes neuronales artificiales . . . . .	12
3.3	Aprendizaje por refuerzo . . . . .	14
3.3.1	Algoritmos de <i>Q-learning</i> . . . . .	16
3.3.2	Algoritmos de gradiente de política . . . . .	18
<b>4</b>	<b>Hardware</b>	<b>21</b>
4.1	Cuadro . . . . .	22
4.2	Motores y hélices . . . . .	22
4.3	Variadores (ESC) . . . . .	24
4.4	Baterías . . . . .	25
4.5	Autopiloto . . . . .	25
4.5.1	Fase de Potencia . . . . .	25
4.5.2	El microcontrolador (ESP32) . . . . .	26
4.5.3	Sensores . . . . .	28
4.6	Banco de pruebas . . . . .	28
<b>5</b>	<b>Software</b>	<b>31</b>
5.1	Software del Autopiloto . . . . .	31
5.1.1	Estimación del Estado . . . . .	31
5.1.2	Interfaz WiFi . . . . .	31
5.1.3	Generacion de comandos <i>on-board</i> . . . . .	32
5.2	Software de la estación . . . . .	32
5.2.1	Entorno de simulación . . . . .	32
5.2.2	Agente (generación de comandos) . . . . .	34
5.2.3	Interfaz Estación-Autopiloto . . . . .	34
5.3	Descripción del equipo . . . . .	35

<b>6 Metodología (Problem Formulation)</b>	<b>36</b>
6.1 Diseño del estado . . . . .	36
6.2 Diseño de las acciones . . . . .	36
6.3 Diseño de la función de recompensa y ajuste de hiperparámetros . . . . .	37
<b>7 Experimentos</b>	<b>39</b>
7.1 Experimentos en simulación . . . . .	39
7.2 Experimentos en real . . . . .	39
<b>8 Discusión</b>	<b>45</b>
<b>9 Conclusiones y trabajo futuro</b>	<b>1</b>

## **Resumen**

hola

# Introducción

Un multirrotor es un vehículo aéreo no tripulado o UAV (por sus siglas en inglés *Unmanned Aerial Vehicles*) cuyos motores y hélices están orientadas de forma vertical. Estas aeronaves son mucho más maniobrables que una aeronave de ala fija, ya que, al poder mantenerse estables en una posición fija, pueden realizar trayectorias de forma más precisa en espacios reducidos. Comparado con un helicóptero, el cual también puede mantenerse estable en un punto fijo, los multirrotos cuentan con un mantenimiento mucho más sencillo debido a la ausencia de complejos mecanismos.

Generalmente los motores de estas aeronaves son eléctricos, ya que poseen una gran velocidad de reacción y son capaces de manejar una gran cantidad de energía con rendimientos muy elevados. La energía que requieren se proporciona a través de baterías, lo cual limita el tiempo de vuelo de estas aeronaves, cuya autonomía es significativamente inferior a la de las aeronaves de ala fija o a la de los helicópteros con motores de combustión.

Todo esto, unido a su reducido precio, ha permitido que se potencie el uso masivo de este tipo de aeronaves para labores de: agricultura de precisión, topografía, inspección industrial y rodajes cinematográficos, entre otros usos. Existen diversas topologías asociados a los multirrotos, dependiendo del número de motores y la disposición de éstos. En este trabajo se ha desarrollado un cuadricóptero o cuadrirrotor (cuenta con cuatro motores) con disposición en X (fig. 1.1).

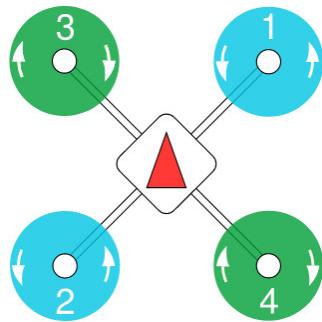


Figura 1.1: Esquema cuadrirrotor en X

Se ha escogido un multirrotor de cuatro motores debido a que posee el mínimo número de motores que permiten un modelo sencillo de la dinámica de la aeronave.

## 1.1. Motivación

La popularidad de estas aeronaves en ámbitos como la inspección, la topografía o el ámbito cinematográfico requieren que la nave sea muy estable y que se pueda manejar con precisión. Para este propósito, todas estas aeronaves cuentan con un sistema electrónico

que se encarga de que la nave sea estable y que facilite el pilotaje de la misma. A este sistema se le conoce como la controladora de vuelo o autopiloto.



Figura 1.2: Autopiloto comercial Pixhawk 4

Actualmente, los autopilotos emplean algoritmos de control clásicos, basados la gran mayoría en reguladores PID (Proporcional Integral y Derivativo) para asegurar la estabilidad de la aeronave y permitir que se pilote de forma precisa. Debido a la peligrosidad de los drones (hélices girando a miles de revoluciones por minuto) es altamente complejo probar algoritmos de control novedosos en un cuadricóptero, sin tenerlo anclado a una estructura base, debido a que se trata de un sistema inestable de forma inherente en bucle abierto.

Las teoría clásica de control empleada para estabilizar un cuadricóptero requiere de un conocimiento preciso del modelo del sistema, junto con fino ajuste de los parámetros del controlador. Los últimos avances en aprendizaje automático han permitido que se desarrollen nuevos algoritmos de control empleando técnicas de aprendizaje por refuerzo y redes neuronales.

## 1.2. Solución propuesta

Con el objetivo de poder desarrollar nuevos algoritmos para el control de cuadrirrotores se ha desarrollado una plataforma, tanto hardware como software, que permita diseñar y probar estos algoritmos de forma segura. Esta plataforma esta constituida de:

- **Entorno de simulación**, sobre el que se puedan probar los algoritmos de control en una aeronave simulada.
- **Cuadrirrotor con autopiloto de diseño propio**, será la aeronave donde se probarán los algoritmos diseñados. Al contar con una controladora de vuelo de diseño propio se pueden implementar los distintos algoritmos de control a bordo.
- **Interfaz con el autopiloto**, el cual comunica a la aeronave con un ordenador, el cual puede enviar comandos a la aeronave y recibir el estado de la misma. Esto permite la implementación de algoritmos *en tierra*, en los cuales la computación del algoritmo se realiza en un ordenador en vez de en el microcontrolador del autopiloto.
- **Banco de pruebas**, con distintas configuraciones en función del experimento que se desee realizar. Este banco permitirá probar los algoritmos de control de forma segura.

Para la validación de la solución propuesta, se ha probado el rendimiento de distintos algoritmos de control basados en técnicas de aprendizaje por refuerzo para poder compararlos con los algoritmos de control clásicos PID.

### 1.3. Objetivos

Para poder llevar a cabo esta plataforma y poder implementar algún algoritmo basado en aprendizaje por refuerzo es necesario desgranar los objetivos principales en tareas de alcance más reducido:

- **Entorno de simulación:**

- Estudio del estado del arte acerca de entornos de simulación para el diseño de algoritmos de control para cuadricópteros.
- Estudio del estado del arte acerca de las técnicas de aprendizaje por refuerzo y el control de drones.
- Adaptación del entorno de simulación a la plataforma escogida e integración con ROS.
- Implementación de algoritmos clásicos en el entorno de simulación.
- Diseño de nuevos algoritmos de control basados en aprendizaje por refuerzo y entrenamiento de los mismos.
- Comparativa de los algoritmos en simulación.

- **Cuadrirrotor con autopiloto:**

- Estudio del estado del arte acerca de los autopilotos comerciales.
- Diseño y construcción completa del autopiloto: componentes, placa de circuito impreso y soldadura de componentes.
- Programación del *firmware* del autopiloto.
- Diseño CAD de los componentes mecánicos del cuadrirrotor e impresión 3D de los mismos.
- Montaje y ensamblaje de todos los componentes de la aeronave.

- **Interfaz con el autopiloto:**

- Diseño del protocolo de comunicación WiFi
- Integración del protocolo con ROS y con el autopiloto.

- **Banco de pruebas:**

- Diseño CAD de las piezas de los distintos bancos de pruebas e impresión 3D de las mismas.

- **Experimentación de los algoritmos en el mundo real:** [reducir o ampliar con los experimentos disponibles](#)

- Implementación a bordo de los algoritmos clásicos.

- Implementación externa de los algoritmos clásicos.
- Implementacion externa de los algoritmos basados en técnicas de aprendizaje por refuerzo.

# Estado del arte

El desarrollo del trabajo realizado se puede subdividir en dos campos: la construcción de la plataforma de sujeción del cuadricóptero y la experimentación con nuevos algoritmos de control empleando técnicas de aprendizaje por refuerzo. Es por esto que, con el ánimo de contextualizar este trabajo dentro de estos dos campos, se ha tratado el estado del arte de forma separada.

## 2.1. Plataformas de control de cuadricópteros

Debido a la creciente popularidad de estas aeronaves, muchos institutos de investigación han trabajado en probar y desarrollar distintos algoritmos de control, para comparar el rendimiento entre ellos. Para poder testear estos algoritmos es muy conveniente contar con una plataforma de control, que te permita medir el rendimiento de estos algoritmos de control de forma precisa y segura.

Principalmente, existen 2 tipos de plataformas para esté propósito: Plataformas tipo gimbal y estructuras con unión esférica. A continuación se tratará mas detalladamente las características de estas estructuras:

### 2.1.1. Plataformas tipo gimbal

Estas plataformas están formados por 3 anillos unidos entre ellos dos a dos, de tal forma que el anillo interior gira respecto al anillo que lo rodea y éste a su vez gira sobre el anillo exterior, como se puede observar en la figura 2.1

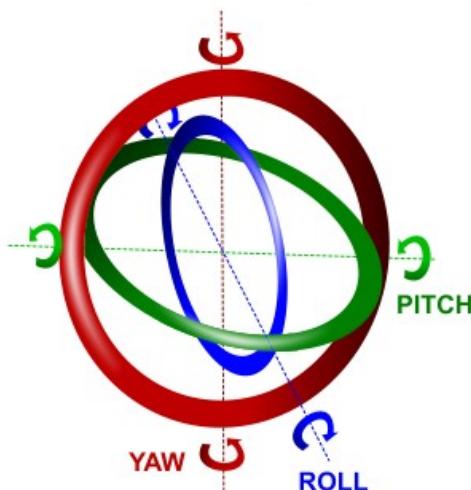


Figura 2.1: Esquema estructura gimbal

Estas plataformas permiten que la aeronave gire entorno a su centro de gravedad, por lo que se reduce la influencia del peso de la aeronave en el control. Además es posible medir los ángulos de euler de la aeronave de forma directa empleando encoders en los ejes de rotación.

Hay empresas que están empezando a comercializar estas plataformas, tanto orientadas para la labor investigadora en el campo de los cuadricópteros, como para la labor docente que se puede llevar a cabo empleando estas plataformas con ánimo didáctico, para el aprendizaje de algoritmos de control.

En cuanto a las plataformas orientadas a la investigación, se encuentra la plataforma FFT gyro, desarrollada por Eureka Dynamics.



Figura 2.2: Plataforma FFT Gyro de Eureka Dynamics

Esta plataforma cuenta con encoders en cada eje de rotación para poder monitorizar el estado de la aeronave con gran precisión y está construida en fibra de carbono para maximizar la rigidez de la estructura minimizando el peso.

### 2.1.2. Plataformas con unión esférica

Estas plataformas se unen a la aeronave a través de una unión esférica, la cual permite que el cuadricóptero pueda girar en tres ángulos, con algunas limitaciones. Por ejemplo, aunque en guinada la aeronave pueda girar libremente, en alabeo y cabeceo este movimiento se ve restringido por las limitaciones mecánicas de la unión, véase fig. 2.4. Esto es una gran diferencia con respecto a las plataformas de tipo gimbal, en las que este suceso no ocurre, dado que en sus articulaciones no existen restricciones en los ángulos de giro.

A pesar de esto, estas plataformas condensan todos los giros en un único punto, por lo que el tamaño de la estructura requerida para anclar un cuadricóptero, es significativamente menor que si se emplea una estructura de tipo gimbal.

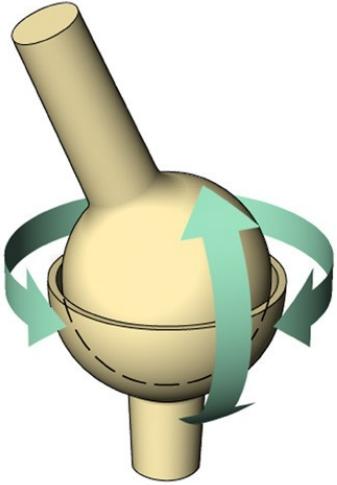


Figura 2.3: Esquema de una union esférica

En cuanto a plataformas comerciales de esté tipo, cabe destacar las producidas por la empresa Quanser, la cúal ha desarrollado una plataforma con fines educativos, que emplea esta unión. Esta plataforma cuenta, al igual que la FFT Gyro con encoders, con la finalidad de conocer con precisión el estado de la aeronave.



Figura 2.4: Plataforma 3 DOF Hover de la empresa Quanser

Entre las dos posibilidades, se ha optado por realizar una plataforma con una unión esférica debido a la mayor simplicidad y menor tamaño de este tipo de plataformas frente a su alternativa.

## 2.2. Aprendizaje por refuerzo

El aprendizaje por refuerzo o *reinforcement learning* es una rama del aprendizaje automático en la que un agente aprende a actuar a medida que va interactuando con su entorno. Es decir, el agente comienza realizando acciones aleatorias y va aprendiendo por ensayo-error. Para que el agente tenga noción de que acciones están “bien” y cuales no, el entorno proporciona una recompensa al agente en función de su comportamiento. Esta rama del aprendizaje autómatico se inspira en la psicología conductista.

Debido a esta capacidad de aprender de forma autónoma, sin necesidad de un conjunto previo de datos etiquetados,sino extrayendo información únicamente de su interacción con el entorno, estos algoritmos son muy empleados en el campo de la robótica para llevar a cabo tareas complicadas, tales como, aprender a andar o a manipular un cubo con destreza [citar](#).

Aunque existen una gran cantidad de ejemplos de uso de estas técnicas de aprendizaje en el ámbito de la robótica, en este trabajo se ha centrado en el empleo de estas técnicas al control de vehículos aéreos no tripulados (UAVs).

Los primeros experimentos en la utilización de algoritmos de control para UAVs, entrenados con aprendizaje con refuerzo, se llevaron a cabo con helicópteros. En 2004, HJ Kim et al. [1] emplearon algoritmos de *reinforcement learning* para estabilizar (*hoover*) un helicóptero y conseguir realizar maniobras acrobáticas, para ello desarrollaron el algoritmo Pegasus [cite](#). Años después, en 2006 Andrew Y. et al [2] continuaron la investigación, en esta ocasión emplearon algoritmos que aprendían a realizar maniobras acrobáticas complejas, como mantener invertido al helicóptero. Para ello, realizaron un modelo esto-cástico no-lineal de la dinámica del helicóptero, para, posteriormente, emplear ese modelo en simulación con el ánimo de generar un controlador capaz de estabilizar al helicóptero invertido.

En 2010 Travis Dierks et al. [3] desarrollaron un controlador no lineal, basado en redes neuronales, para estabilizar un cuadricóptero y seguir trayectorias. Para obtener un modelo completo de la aeronave, que fuera capaz también de tener en cuenta otros parámetros externos a la aeronave, como el coeficiente aerodinámico, el aprendizaje de la red neuronal se realizaba a tiempo real, mientras el cuadricóptero volaba.

Unos años después, en 2017 Jemin Hwangbo et al. [4] desarrollaron un método para controlar un quadricóptero con una red neuronal usando técnicas de *reinforcement learning*. Desarrollaron un algoritmo, basado en la optimización determinista de la política y empleando decenso de gradientes natural para optimizar la misma. Consiguieron que el cuadricóptero fuera capaz de estabilizarse aún cuando partía de condiciones adversas, como ser lanzado casi boca abajo contra el suelo. Sin embargo, para conseguirlo, necesitaban frecuencias de actualización muy elevadas, del orden de los 100Khz, dos órdenes de magnitud mayor que un algoritmo típico.

En 2018 William Koch et al. [5] desarrollaron un entorno de simulación, GYMFC, para el desarrollo de nuevos algoritmos de control basados en redes neuronales. En su artículo compararon el rendimiento, en simulación, de distintos algoritmos de aprendizaje por refuerzo a la hora de cerrar el bucle de control en velocidad de un cuadricóptero. Posteriormente, a comienzos de 2019, desarrollaron Neuroflight [citar](#), un *firmware* para autopilotos *open-source*. Este *firmware*, a diferencia del de los autopilotos convencionales emplea bucles de control desarrollados con técnicas de aprendizaje por refuerzo, con los que consiguen controlar un cuadricóptero real.

# Fundamentos teóricos

## 3.1. Controlador PID

Un regulador PID es un controlador lineal realimentado. Matemáticamente se expresa como

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (3.1)$$

donde  $e(t)$  es la señal de control,  $u(t)$  representa la salida del regulador y  $K_p, K_i, K_d$  son parámetros ajustables de los cuales depende la dinámica y la estabilidad del sistema. Como se puede observar, el regulador consta de tres partes: la parte proporcional (P) tiene en cuenta el error actual, la parte integral (I) tiene en cuenta el histórico de los errores y la parte derivativa (D) tiene en cuenta el “futuro” del error.

En este trabajo se han empleado controladores PID en dos tipos de bucles de control con topologías diferentes. Para el bucle de control más sencillo, se aplica el regulador PID sobre el error obtenido al realimentar la salida del bucle, veáse fig. 3.1.

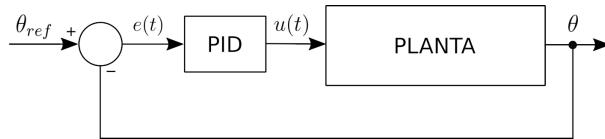


Figura 3.1: Bucle de control realimentado con regulador PID

La otra variante emplea un bucle de control en cascada, el cual consta de dos bucles cerrados de control. El bucle externo emplea un regulador P, este bucle genera la señal de referencia, para otro bucle de control interno, basado en la salida del sistema. El bucle interno emplea un regulador PID para controlar la magnitud de una variable interna. En el caso particular de este trabajo, el bucle externo intenta alcanzar una referencia de posición angular  $\theta_{ref}$ , para ello el regulador P, envía una referencia de velocidad angular  $\dot{\theta}_{ref}$  a un controlador de velocidad con un regulador PID, vease fig. 3.2.

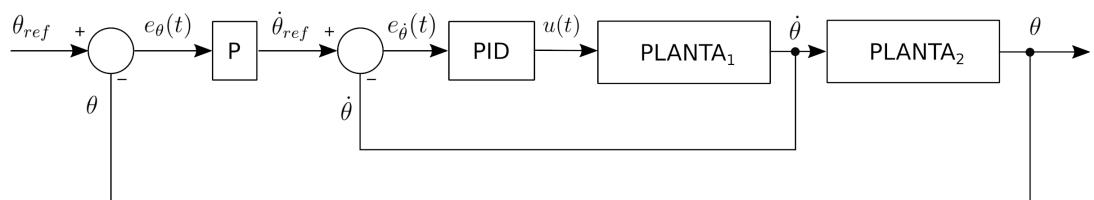


Figura 3.2: Bucle de control en cascada

## Control de la aeronave

Para el control de la aeronave es necesario cerrar simultáneamente los 3 bucles de control, uno para cada ángulo. Cada bucle de control produce una salida  $u(t)$  a la planta, en este caso, a la aeronave. Para traducir estas señales a los comandos que se le envían a los motores, es necesario realizar una transformación.

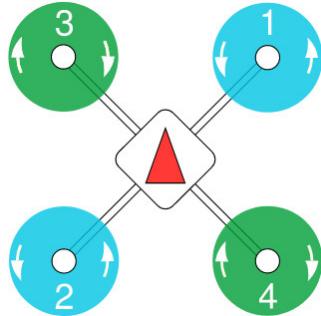


Figura 3.3: Esquema cuadrirrotor en X

Intuitivamente, si se desea que el dron incline el morro hacia delante, la velocidad de los 2 motores traseros deberá aumentar, mientras que la velocidad de los motores delanteros deberá disminuirse. Si consideramos que inclinar el morro hacia delante implica que la nave tenga un ángulo  $\theta > 0$  y que los motores estan dispuestos según la figura 3.3, entonces

$$\begin{aligned} w_1 &= -u_\theta(t) \\ w_2 &= +u_\theta(t) \\ w_3 &= -u_\theta(t) \\ w_4 &= +u_\theta(t) \end{aligned}$$

Si lo expresamos matricialmente:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} -1 \\ +1 \\ -1 \\ +1 \end{bmatrix} u_\theta(t) \quad (3.2)$$

siendo  $w_i$  la velocidad del motor i y  $u_\theta(t)$  la salida del controlador de *pitch*. Si se cierran los 3 bucles de control de forma simultánea y se suman las contribuciones de cada bucle sobre las acciones de control de forma similar

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & -1 & +1 \\ +1 & +1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -1 \end{bmatrix}}_{\text{Matriz de transformación}} \begin{bmatrix} u_\varphi(t) \\ u_\theta(t) \\ u_\psi(t) \end{bmatrix} \quad (3.3)$$

Esta matriz de transformación relaciona las salidas de los controladores con los comandos de los motores, modificando estos valores se puede aumentar la influencia de un bucle con respecto a otro.

## 3.2. Redes neuronales artificiales

Una red neuronal artificial (ANN) esta compuesta por un conjunto de nodos o perceptrones interconectados entre sí. Estos perceptrones se agrupan en capas “ocultas”, se les atribuye este nombre debido a que todos los nodos de una capa se interconectan con todos los nodos de la capa anterior, por lo que después del aprendizaje de la red no se sabe cuales son los perceptrones de la capa anterior que influyen en un nodo.

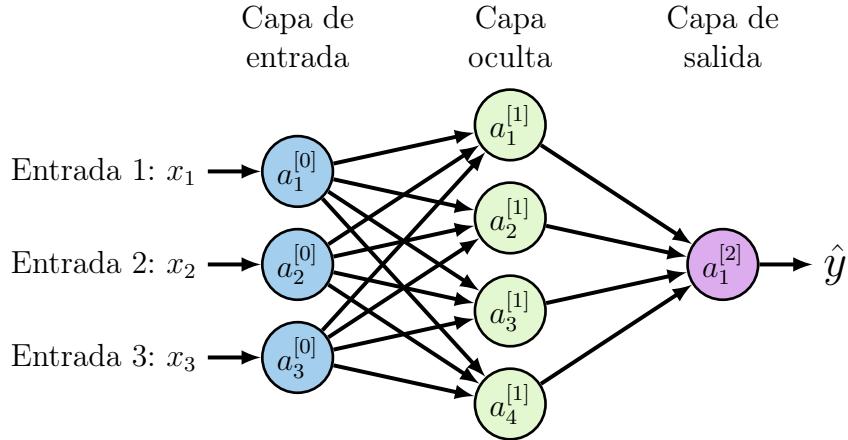


Figura 3.4: Esquema de una red neuronal artificial

Cada perceptron es la unidad mínima de computación de una ANN, estas unidades se dividen en dos partes, una parte lineal y una parte de activación o no lineal. En la parte lineal o parte “Z” se computa una regresión lineal de las salidas de los nodos anteriores.

$$z_i^{[l]} = \sum_{j=0}^{n^{[l-1]}} w_{ij} \cdot a_j^{[l-1]} + b_i \quad i = 0, \dots, n^{[l]} \quad (3.4)$$

$$a_i^{[l]} = g(z_i^{[l]}) \quad i = 0, \dots, n^{[l]} \quad (3.5)$$

El superíndice  $[l]$  hace referencia a la capa en la que se encuentra el elemento. Siendo  $n^{[l]}$  el número de nodos de la capa  $l$ -ésima.

A los coeficientes  $w_{ij}$  se les denomina los pesos del perceptrón y  $b_i$  es el término independiente de la regresión.

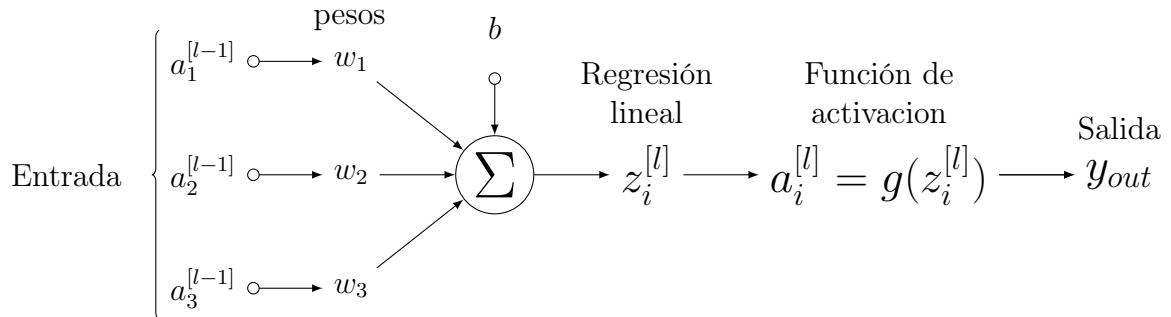


Figura 3.5: Esquema de un perceptrón

La función  $g(z)$  es la función de activación del nodo. Estas funciones proporcionan

no linealidad a la red neuronal, permitiendo a estas la capacidad de generar modelos con grandes no linealidades. Las funciones de activación más frecuentes en la literatura son:

- Función sigmoide:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \sigma(z) : \mathbb{R} \rightarrow [0, 1] \quad (3.6)$$

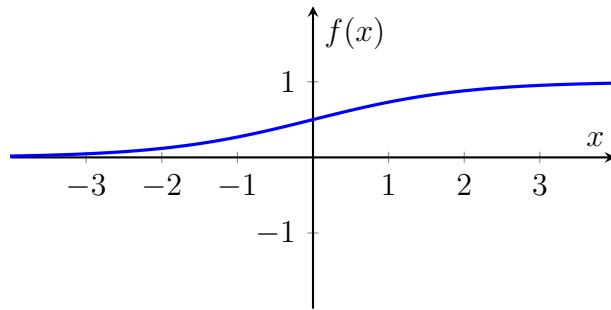


Figura 3.6: Función sigmoide

- Tangente hiperbólica:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \tanh(z) : \mathbb{R} \rightarrow [-1, 1] \quad (3.7)$$

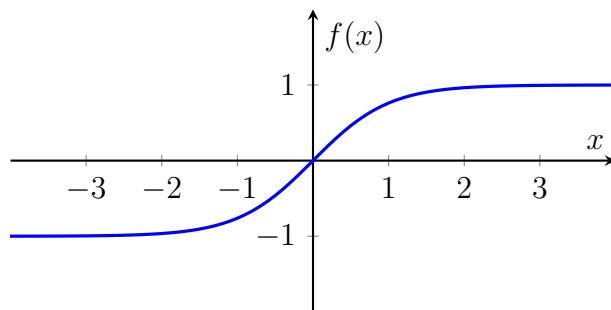


Figura 3.7: Función tangente hiperbólica

- ReLu (del inglés *Rectified Linear Unit*):

$$g(z) = \max(0, z) \quad g(z) : \mathbb{R} \rightarrow [0, +\infty] \quad (3.8)$$

Para conseguir que la red neuronal realice predicciones precisas es necesario ajustar los pesos de la red, a este proceso es al que se denomina entrenamiento o aprendizaje. En el paradigma del aprendizaje supervisado este entrenamiento se realiza sometiendo a la red a ejemplos cuya salida es conocida. El objetivo de la red es minimizar el error de la esperanza de la estimación con respecto a la salida real del ejemplo. Formalmente, se

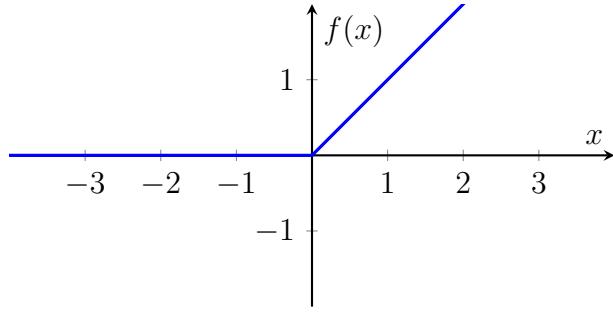


Figura 3.8: Función ReLu

define una función de coste  $\mathcal{J}$  la cual se quiere minimizar. Por ejemplo, una función de coste típica para problemas de clasificación binaria es la *binary cross-entropy*:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \quad (3.9)$$

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \quad (3.10)$$

donde  $\hat{y}$  denota la estimación de la salida realizada por parte de la red,  $y$  la salida conocida y  $m$  el número de ejemplos.

Para minimizar esta función de coste, que depende de los pesos de la red, existen distintos métodos, uno de los más usados es el método del descenso de gradiente. Este método emplea la “propagación hacia atrás” (del inglés *back propagation*) de la red neuronal, esto consiste en obtener las derivadas parciales de la función de coste con respecto a los pesos de cada nodo y actualizar estos pesos en la dirección opuesta al máximo gradiente.

$$w := w - \alpha \frac{\partial \mathcal{J}(w, b)}{\partial w} \quad (3.11)$$

$$b := b - \alpha \frac{\partial \mathcal{J}(w, b)}{\partial b} \quad (3.12)$$

donde  $\alpha$  denota la tasa de aprendizaje, es decir, lo rápido que varian estos pesos.

### 3.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo o *Reinforcement learning* [6] es un área del aprendizaje automático o *Machine Learning* en el que un agente interactúa con un entorno buscando la mejor acción a realizar en función de su estado actual, de manera que maximice las recompensas acumuladas en el tiempo.

Se diferencia de otras técnicas de aprendizaje automático es su enfoque orientado a la interacción directa con el entorno, sin basarse en un modelo completo del entorno o en un conjunto de ejemplos supervisados.

#### Elementos del aprendizaje por refuerzo

Además del agente y el entorno se pueden identificar tres elementos principales más en un sistema de aprendizaje con refuerzo:

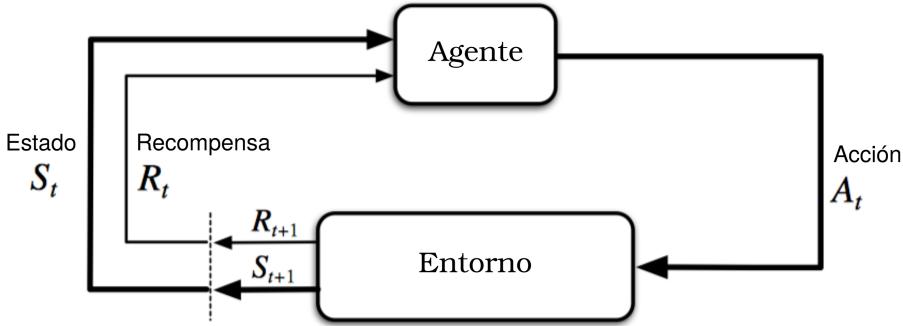


Figura 3.9: Diagrama canónico del bucle de interacción entorno-agente

- **Política ( $\pi$ )** (Proveniente del término anglosajón *policy*, el cual es el término empleado en el estado del arte). Define el conjunto de acciones que debe realizar el agente para conseguir maximizar su recompensa en función su estado, el cuál es percibido a través del entorno. La *policy* constituye el núcleo del agente y nos permite determinar su comportamiento. Estas políticas pueden ser estocásticas.
- **Recompensa ( $R_t$ )**. Define el objetivo del agente en un problema de aprendizaje por refuerzo. En cada salto de tiempo (*step*) el agente recibe una recompensa por parte del entorno.
- **Función de valor ( $V_s^\pi$ )**. Representa la máxima recompensa que puede esperar obtener un agente desde un estado concreto, empleando una política concreta, es decir, tiene en cuenta la recompensa a largo plazo, no solo la inmediata.

$$V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right] \quad (3.13)$$

Algunos algoritmos de aprendizaje por refuerzo requieren de la definición de un **modelo del entorno**. Este modelo permite predecir el comportamiento que va a tener el entorno a lo largo del tiempo. Los algoritmos que emplean un modelo se les denomina *model-based*, mientras que, a los algoritmos que no requieren de un modelo del entorno se les denomina *model-free*.

### Procesos de decisión de Markov

El aprendizaje por refuerzo emplea el marco formal de los procesos de decisión de Markov (*MDP*) en los cuales para definir la interacción entre en agente y el entorno en términos de estados, acciones y recompensas.

Un proceso de decisión de Markov está compuesto por la 4-tupla  $(S, A, P_a, R_a)$  donde:

- **S** representa el estado del agente.
- **A** es el conjunto de acciones que puede realizar el agente.  $A_s$  denota las acciones que puede realizar el agente desde un estado  $s$ .
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ . Partiendo de un estado  $s$ ,  $P_a$  representa la probabilidad de pasar al estado  $s'$  tomando la acción  $a$ .

- $R_a(s, s')$  denota la recompensa inmediata que recibiría el agente al realizar la transición de  $s$  a  $s'$  mediante la acción  $a$ .

Estos procesos cumplen la propiedad de Markov, es decir, que el pasado no influye en el agente, lo único relevante es el estado actual.

El problema principal de los MDP es encontrar la secuencia de acciones que debe realizar el agente para maximizar la recompensa a largo plazo, es decir, encontrar la política  $\pi$  óptima, que permita maximizar la recompensa. Una vez encontrada la política  $\pi$  óptima el problema se reduce a una cadena de Markov, dado que la acción  $a$  a realizar en un estado  $s$  viene completamente definida por el estado  $s$  y la probabilidad  $P_a$ .

La política  $\pi$  óptima es aquella que maximice la recompensa acumulada, es decir, la suma descontada de las recompensas instantáneas percibidas por el entorno:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3.14)$$

donde  $\gamma$  es el factor de descuento, el cual debe cumplir  $0 \leq \gamma \leq 1$ , cuanto mayor sea este factor menos importante es la recompensa inmediata.

### 3.3.1. Algoritmos de *Q-learning*

En este trabajo se han trabajado con 2 tipos de algoritmos de aprendizaje por refuerzo: métodos basados en *Q-learning* y métodos de gradiente de las políticas.

Los algoritmos de Q-learning se basan en la función estado-acción, también conocida como función Q, de ahí el nombre de estos algoritmos. Esta función denota el valor de tomar una acción para un estado concreto, siguiendo una política  $\pi$ . Formalmente, la función Q se define como

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right] \quad (3.15)$$

La diferencia entre la función Q y la función de valor radica en que la función de valor cuantifica el valor, en términos de recompensa acumulada, que posee un estado concreto, mientras que la función Q evalúa la conveniencia de tomar una acción concreta en un estado determinado. Este matiz se empleará posteriormente para definir la función de ventaja en el apartado 3.3.2.

El objetivo de estos algoritmos se basa en obtener la función óptima  $Q^*$ , esta función cumple la ecuación de Bellman

$$Q^*(s, a) = \mathbb{E}_{s'} \left[ r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right] \quad (3.16)$$

siendo  $s'$  y  $a'$  la secuencia siguiente de estados y acciones, respectivamente.

Basándose en esta ecuación la función  $Q^*$  se podría calcular como un proceso iterativo de la forma

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') \middle| s, a \right] \quad (3.17)$$

Aunque este proceso converge a la función estado-acción óptima,  $Q_i \rightarrow Q^*$  cuando  $i \rightarrow \infty$  [6], esto no es práctico, debido a que esta aproximación de la función Q se estima de forma independiente para cada secuencia concreta, sin generalizar. En lugar de esto, se suele emplear una función que se aproxime a esta función estado-acción. En los casos en

los que esta función sea aproximada por una red neuronal, ésta se denota por  $Q(s, a; \theta)$  siendo  $\theta$  los pesos de esta red-Q (*Q-network*).

La función de pérdida que se intenta minimizar para entrenar esta red es

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(s,a)} [(y_i - Q(s, a; \theta_i))^2] \quad (3.18)$$

donde  $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})]$  es el objetivo(*target*) para la iteración  $i$  y  $\rho(s, a)$  es la distribución del comportamiento, es decir, distribución de probabilidad de las secuencias  $a', s'$  a lo largo del tiempo.

Esta función de pérdida se puede minimizar con algoritmos de optimización como el Descenso de gradientes estocástico (SDG).

## DQN

El algoritmo DQN (*Deep Q-network*) es un algoritmo basado en *Q-learning* desarrollado por Mnih et al. [citar](#) en 2015. Con este algoritmo la empresa DeepMind consiguió desarrollar un agente capaz de jugar a los juegos de la clásica consola Atari, superando con creces el rendimiento de los jugadores profesionales en algunos juegos. Este algoritmo está diseñado para actuar sobre un conjunto de acciones discretas, como son las acciones de la consola Atari.

El nombre del algoritmo, hace referencia al empleo de una red neuronal profunda (*Deep Network*) como aproximador de la función Q (*Q-network*). Este algoritmo introduce, principalmente, dos modificaciones sobre los algoritmos Q que favorecen el aprendizaje:

1. **Creación de un “contenedor de repeticiones”** (*buffer replay*) en el que se almacenan las experiencias del agente en cada iteración temporal  $e_t = (s_t, a_t, r_t, s_{t+1})$  en un conjunto de datos  $D_t = \{e_1, \dots, e_t\}$  recabados durante varios episodios.

Este contenedor se emplea para obtener una muestra de aleatoriedad de experiencias y constituir un pequeño lote con el que optimizar la red-Q.

Emplear este método proporciona varias ventajas notorias:

- Permite el empleo de los datos recabados en cada paso temporal para optimizar los pesos en múltiples interacciones, lo que se traduce en un mejor aprovechamiento de las experiencias del agente.
- Al tomar muestras de forma consecutiva el aprendizaje se vuelve ineficiente debido a la alta correlación entre las muestras. Al tomar muestras aleatorias del contenedor se consigue romper esta correlación.

2. **Empleo de una red objetivo  $Q'$**  para la actualización de pesos. Cada  $C$  actualizaciones se clona la red  $Q$  para obtener la red objetivo  $\hat{Q}$ . Esta red objetivo  $\hat{Q}$  es la que emplea para realizar el cálculo de la expresión  $y_i$  durante la actualización de pesos.

$$y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-)] \quad (3.19)$$

En este algoritmo se realiza la optimización mediante el algoritmo SDG de la función de pérdida 3.18. Cada  $C$  pasos se reinicia la función Q con los valores del objetivo  $Q = \hat{Q}$ .

Al emplear una red con parámetros desactualizados, a la hora de actualizar la red, se reduce las oscilaciones en el entrenamiento y se favorece la convergencia.

En este algoritmo, la política  $\pi$  que dicta que acción  $a$  es la más conveniente en un estado  $s$  determinado se obtiene a partir de la función  $Q$  de forma inmediata

$$a_t(s) = \operatorname{argmax}_a Q(s, a | \theta) \quad (3.20)$$

es decir, se escoge la acción  $a$  que maximice la función  $Q$  en ese estado.

## DDPG

El algoritmo DDPG (*Deep Deterministic Policy Gradient*), creado por Lillicrap et al. [citar](#) en 2016, adapta las ideas del DQN a un dominio de acciones continuo. Este algoritmo emplea la filosofía *actor-critic*, en la que una parte evalúa el valor de una acción en un estado (crítico) y otra define la política que debe realizar el agente (actor).

La función actor  $\mu(s | \theta^\mu)$  devuelve la acción que se debe realizar en el estado  $s$ , es decir, la política  $\pi$  que debe seguir el agente. El crítico  $Q(s, a)$  se entrena empleando la identidad de Bellman al igual que en los algoritmos de *Q-learning*.

Para actualizar los pesos de la función actor se emplea el gradiente de la función de coste empleados por Silver et al. [citar DPG paper](#)

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho} [\nabla_{\theta^\mu} Q(s, a | \theta^Q)|_{s=s_t, a=\mu(s_t | \theta^\mu)}] \\ &= \mathbb{E}_{s_t \sim \rho} [\nabla_a Q(s, a | \theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s | \theta^\mu)|_{s=s_t}] \end{aligned} \quad (3.21)$$

Para favorecer la exploración del algoritmo se genera una política de exploración  $\mu'$  se le añade ruido  $\mathcal{N}$  a la política del actor

$$\mu'(s_t) = \mu(s_t | \theta_t^\mu) + \mathcal{N} \quad (3.22)$$

este ruido se puede generar de varias formas, en el artículo se emplea ruido generado mediante el proceso de Ornstein-Uhlenbeck ([citar](#)) para generar ruido correlado temporalmente.

Este algoritmo mantiene el contenedor de repeticiones y el empleo de redes objetivo del DQN aunque añade alguna modificación en esta última. En vez de reiniciar el valor de la red con el valor de la red objetivo cada  $C$  épocas, en el DDPG, esta actualización de las redes  $Q$  y  $\mu$  se realiza de forma suave mediante una media ponderada móvil.

$$\theta^{\hat{Q}} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{\hat{Q}} \quad (3.23)$$

$$\theta^{\hat{\mu}} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\hat{\mu}} \quad (3.24)$$

con  $\tau \ll 1$ . De esta forma los pesos de la red cambian de forma suave, lo que favorece la estabilidad del entrenamiento.

### 3.3.2. Algoritmos de gradiente de política

En los algoritmos de *Q-learning* es objetivo es encontrar una función  $Q(s, a | \theta)$  que se aproxime lo máximo a la función  $Q^*$ , para a partir de esta función extraer la política óptima. En cambio, en los algoritmos de gradiente de política (*policy gradients*) lo que se parametriza, es una política  $\pi_\theta$ , por lo que el objetivo de estos algoritmos es encontrar los parámetros que maximizan la recompensa acumulada. La función de coste a maximizar de estos algoritmos es directamente

$$J(\theta) = \mathbb{E}_\pi[r(\tau)] \quad (3.25)$$

siendo  $r(\tau)$  la recompensa total obtenida al realizar una trayectoria  $\tau$ . Si se emplea un algoritmo de optimización basado en el descenso de gradientes los pesos se actualizarian de la siguiente forma

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \quad (3.26)$$

por lo que el objetivo es encontrar el gradiente  $\nabla J(\theta_t)$  e iterar en el tiempo para obtener la política optima. El teorema del gradiente de política [citar](#) se obtiene que

$$\nabla J(\theta_t) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) A^{\pi_\theta}(s_t, a_t) \right] \quad (3.27)$$

Siendo  $A^\pi$  la función de ventaja, definida como

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \quad (3.28)$$

intuitivamente, la función de ventaja representa el beneficio o perjuicio de elegir la acción  $a_t$  en lugar de seguir la política  $\pi$  en un estado determinado. Al emplear esta función en la expresión 3.27 la política evoluciona hacia las acciones que consiguen una recompensa mayor que la acción media.

## TRPO

El algoritmo TRPO (*Trust Region Policy Optimization*) es un algortimo desarrollado por Schulman et al. [citar](#) en 2017, que emplea el concepto de la región de confianza (*Trust Region*) para favorecer la convergencia del entrenamiento.

La región de confianza limita el cambio que puede sufrir la política en cada iteración, de esta forma, primero se establece el tamaño máximo del paso y posteriormente se localiza el punto óptimo dentro de esta región.

En este algoritmo se intenta

$$\text{Maximizar}_{\theta} \quad \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t \right] \quad (3.29)$$

$$\text{Sujeto a} \quad \hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{old}}(\cdot | s_t), \pi_\theta(\cdot | s_t)]] \leq \delta \quad (3.30)$$

La expresión 3.29 es una función límite inferior a  $J(\theta^*)$ , es decir, para cualquier valor de  $\theta \neq \theta^*$ , todos los puntos de la expresión 3.29 se encuentran por debajo de la función  $J(\theta^*)$ . Esto implica que se pueden emplear métodos de minorización-maximización (algoritmos MM), los cuales van aproximando una función limite inferior, a otra, límite superior mediante iteraciones (fig. 3.10).

Al emplear este método, junto con las limitaciones impuestas con la región de confianza, se consigue mejorar la convergencia del entrenamiento a la solución óptima.

## PPO

El algoritmo PPO (*Proximal Policy Optimization*), fue desarrollado por Schulman et al. [cite](#) en 2017, como una mejora del algoritmo TRPO explicado anteriormente. El algoritmo TRPO tiene una complejidad de implementación muy elevada, el objetivo PPO era simplifica esta complejidad y consigue mejorar el rendimiento de su predecesor.

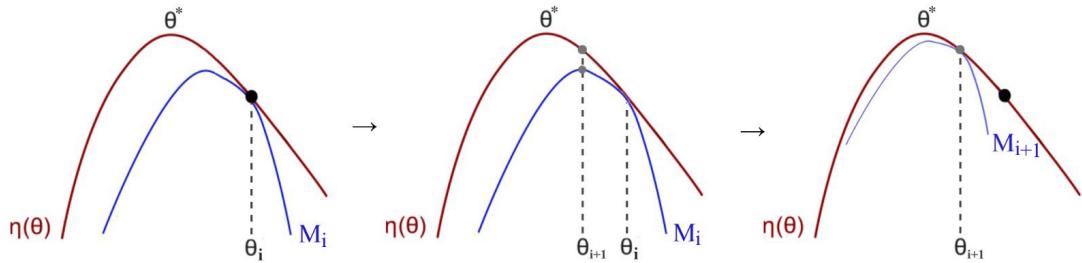


Figura 3.10: Evolución de una función límite inferior con un algoritmo MM.

Este algoritmo tiene dos variantes, aunque, en este trabajo, solamente se tratará sobre la variante que “recorta” el objetivo (*Clipped Surrogate Objective*).

Sea  $r_t(\theta)$  el radio de probabilidad  $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ , empleando esto en la expresión de la función objetivo del TRPO (eq. 3.29)

$$L^{\text{CPI}}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t [r_t(\theta) \hat{A}_t] \quad (3.31)$$

Si no hubiera ninguna limitación, al maximizar  $L^{\text{CPI}}$ , se realizarían cambios muy grandes de la política. Es por esto que Schulman et al. se plantean como modificar esta función para penalizar cambios grandes en las políticas, los cuales distancian  $r_t(\theta)$  de 1. La función objetivo que proponen

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (3.32)$$

donde  $\epsilon$  es un hiperparámetro, por ejemplo,  $\epsilon = 0.2$ . De esta forma, se limita que se potencien mucho las acciones realizadas, cuando la ventaja es positiva y que, cuando la ventaja sea negativa, no se rechacen de forma permanente las acciones realizadas, véase fig. 3.11.

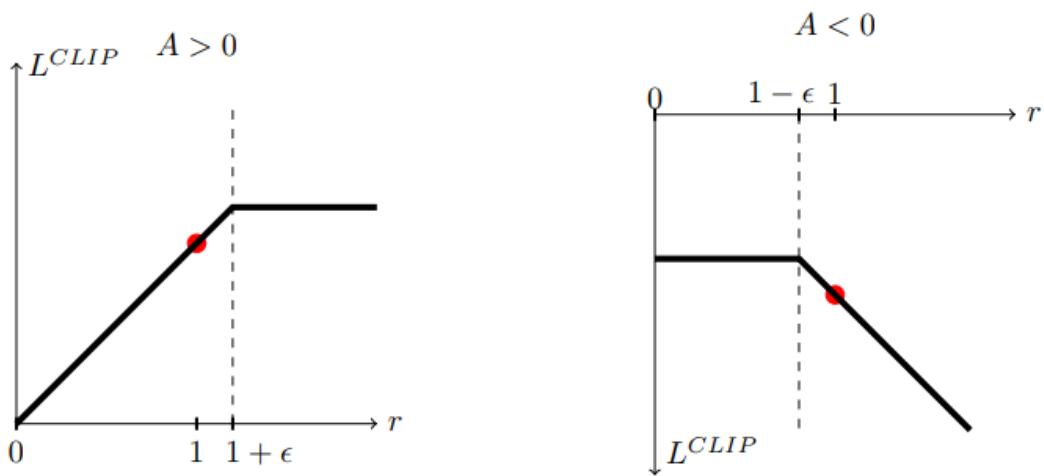


Figura 3.11: Gráficas de la función objetivo  $L^{\text{CLIP}}(\theta)$  para distintos valores de  $r$ , dependiendo del signo de la ventaja  $A$

De esta manera, la función objetivo se puede optimizar con un optimizador de primer orden, por lo que, aunque a veces se tomen algunas acciones erróneas, se consigue obtener un mejor rendimiento de una forma mucho más sencilla al TRPO.

# Hardware

Con el ánimo de tener una plataforma real sobre la que probar el rendimiento de los algoritmos de control diseñados, se ha diseñado y construido un cuadrirroto *ad hoc* para este propósito. Un cuadricóptero convencional cuenta con: un chasis o cuadro que lo sostenta, cuatro motores y la electrónica necesaria para controlarlos, una controladora de vuelo que lo comanda y baterías que le proporcionan energía.



Figura 4.1: Cuadricóptero diseñado con el autopiloto incoporado.

A continuación se detallará como son las distintas partes físicas del dron que se ha fabricado.

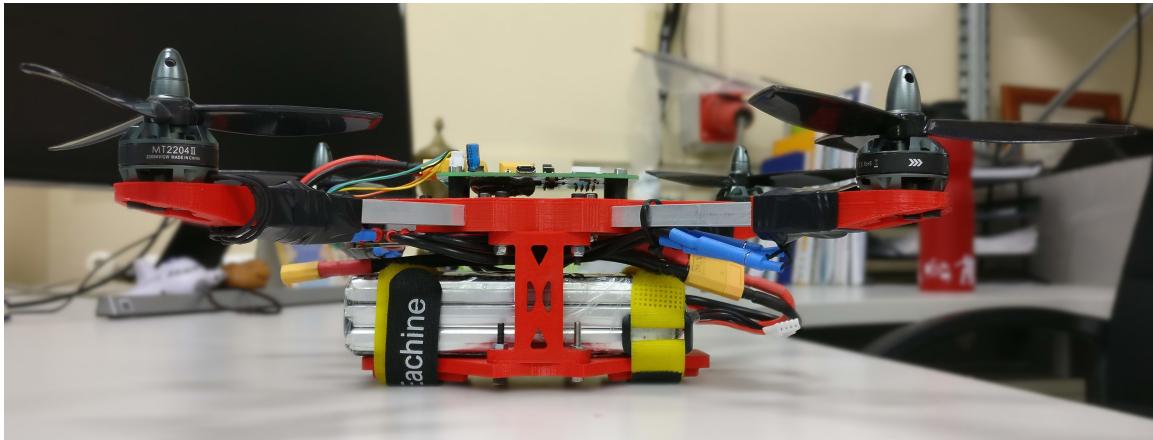


Figura 4.2: Cuadricóptero diseñado con el autopiloto incoporado.

## 4.1. Cuadro

El *frame* está compuesto por perfiles de aluminio y piezas de PLA fabricadas mediante impresión 3D de diseño propio. La estructura básica está formada por 5 conjuntos de piezas distintos:

- **Portamotores:** son las piezas donde se alojan los motores. Para conseguir una buena fijacion los motores se atornillan a los portamotores empleando 4 tornillos dispuestos según los vértices de un rombo.

[Imagen](#)

- **Brazos:** se encargan de unir los portamotores con el *núcleo* de la estructura. En este diseño, los brazos consisten en perfiles de aluminio de sección cuadrada de 8mm de lado.

- **Núcleo:** Es la pieza principal del diseño, en la que se anclan el resto de las partes y donde se alojan los componentes electrónicos. Esta pieza sustenta los brazos y el porta-baterias de la aeronave. Cuenta con agujeros a medida para poder situar el autopiloto y los variadores.

[Imagen](#)

- **Separadores:** su propósito es mantener unidos el *núcleo* y el porta-baterías manteniendo una separación fija entre ellas.

[Imagen](#)

- **Porta-baterías:** Es la parte inferior del cuadricóptero. En ella se apoya la batería Li-Po que alimenta al dron y se mantiene anclada durante el vuelo. Además posee unas protuberancias cuya función se asemejaría a las de un tren de aterrizaje.

[Imagen](#)

## 4.2. Motores y hélices

El dron cuenta con 4 motores sin escobillas (*brushless*) LHI MT2204 II de 2300KV con una tensión de alimentación entre 7.2 V y 11.1 V (2s -3s en una batería LiPo) y una

corriente continua máxima de 16A.



Figura 4.3: Motores LHI MT2204 II empleados

Se ha estimado que el peso aproximado de la aeronave se encuentra entorno a los 800 gramos. La literatura recomienda que los motores que se escogen deben tener empuje suficiente para poder levantar el doble del peso de la aeronave. Observando la tabla de especificaciones de los motores, se observa que, la hélice HQ5040 proporciona empuje suficiente con un ratio empuje/potencia bastante elevado, como se puede observar en la fig. 4.4.

Motor type	The voltage (V)	Propeller size	current (A)	thrust (G)	power (W)	efficiency (G/W)	speed (RPM)
MT2204 II - 2300KV	8	HQ5040	4.9	210	39.2	5.4	13840
		HQ6045	8.2	320	65.6	4.9	11300
		6030 CF	6.4	240	51.2	4.7	11910
	12	5030 CF	7.5	310	90.0	3.4	20100
		6030 CF	11.5	440	138.0	3.2	16300
		<b>HQ5040</b>	<b>8.4</b>	<b>390</b>	<b>100.8</b>	<b>3.9</b>	<b>19040</b>
		HQ6045	13.2	530	158.4	3.3	14600
	14.8	HQ5040	10.7	510	158.4	3.2	22180
		HQ6045	15.7	620	232.4	2.7	16100

Figura 4.4: Tabla de especificaciones motor MT2204 II.

Es por esto que se han empleado hélices tripala HQ5040 de policarbonato y fibra de vidrio (fig. 4.5).



Figura 4.5: Hélices tripala HQ5040

### 4.3. Variadores (ESC)

Los motores que se han escogido son trifásicos, es decir, se alimentan con 3 corrientes alternas monofásicas de igual frecuencia y amplitud, desfasadas  $120^\circ$  eléctricos. Para obtener estas formas de ondas a partir de la corriente continua de las baterías, se utilizan los variadores.

Un variador o *ESC* (*Electronic Speed Control*) es un circuito electrónico que se encarga de generar las ondas eléctricas necesarias para controlar y regular la velocidad de un motor eléctrico. Cuenta con un microcontrolador, el cual se encarga de conmutar los interruptores de potencia, con el ánimo de alimentar las distintas fases del motor de forma sincronizada, haciendo que gire a la velocidad deseada, véase la fig. 5.1.

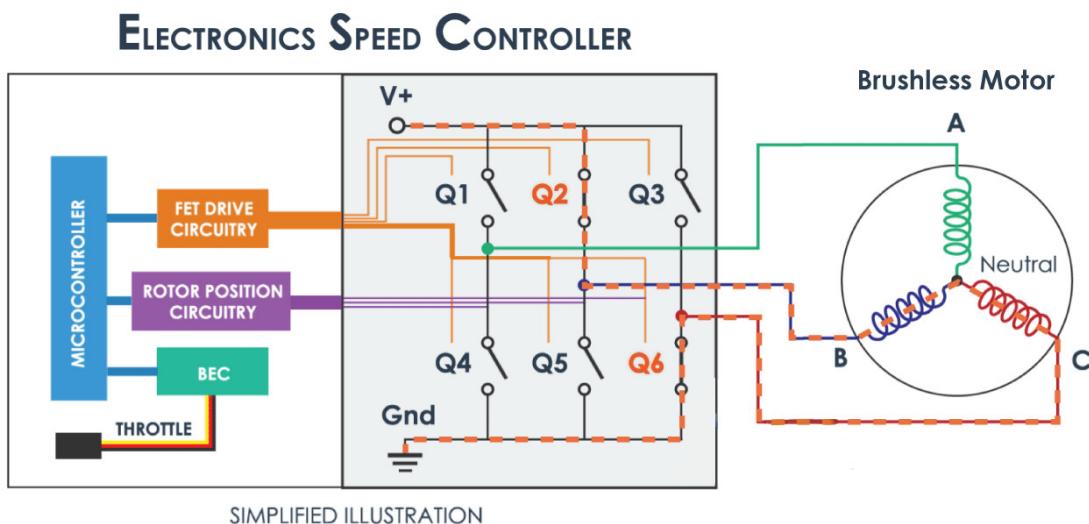


Figura 4.6: Funcionamiento ESC ([www.hwtomechatronics.com](http://www.hwtomechatronics.com))

Para el cuadricóptero se ha optado por emplear un variador BLHeli Multistar Race (fig. 4.7) que integra 4 variadores en uno, es decir se pueden alimentar 4 motores trifásicos con él. Estos variadores soportan una corriente de hasta 30 A cada uno.



Figura 4.7: ESC Multistar Race 4 in 1 30A BLHeli empleado

## 4.4. Baterías

Para alimentar al dron, se han elegido baterías de litio-polímero (LiPo) de 3 celdas, lo que supone una tensión nominal de 11.1V (cada celda proporciona 2.7V). La tensión máxima que puede suministrar la batería es de 12.6V y la tensión mínima es de 9.6V. Si la tensión de alguna celda desciende de 3.2V la batería se puede dañar permanentemente. La batería escogida es una ZNACE de 3 celdas, con una capacidad de 5200mah y una tasa de descarga de 35C, por lo que es capaz de proporcionar una corriente de salida máxima de  $5.2\text{Ah} \cdot 35\text{C} = 182\text{ A}$ . La batería empleada cuenta con un conector XT60, por lo que el máximo de corriente que soporta en régimen continuo es de 60 A.



Figura 4.8: Batería LiPo 3s 35C 5200 mah de la marca NZACE empleada

## 4.5. Autopiloto

En los drones, el sistema que se encarga de estabilizar al cuadricóptero y hacerlo pilotable se denomina la controladora de vuelo o el Autopiloto. Existe una gran variedad de controladoras en el mercado, pero para este trabajo se ha diseñado una controladora propia con el fin de poder tener acceso a todos los sensores y a implementar el algoritmo de control de forma óptima. El autopiloto consta de 3 partes diferenciadas: la electrónica de potencia, el microcontrolador y los sensores. A continuación [se tratará](#) sobre estas partes con más detalle.

### 4.5.1. Fase de Potencia

Con el fin de poder gestionar la potencia entregada por las baterías a la placa y a los motores se ha diseñado una etapa de potencia en la que se debe mencionar dos partes: el interruptor de potencia y el regulador a 3.3 Voltios.

#### Interruptor de potencia

Los motores del dron pueden llegar a consumir 12 Amperios cada uno, lo que los cuatro motores pueden llegar a consumir 48 Amperios. Un interruptor con tamaño reducido no puede manejar tanta corriente, por ello se ha empleado un transistor MOSFET de canal P por el que pueden circular hasta 100 Amperios, con el fin de abrir o cerrar el paso

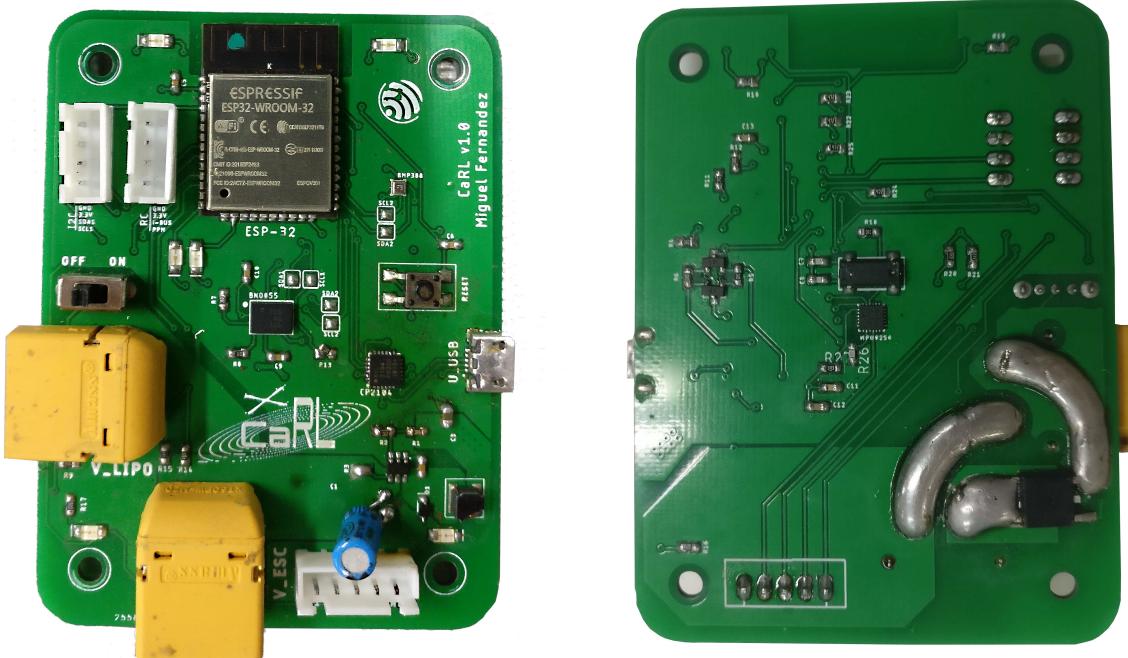


Figura 4.9: PCB autopiloto CaRL, anverso y reverso.

de corriente desde las baterías al resto de la placa. El MOSFET se controla con un interruptor de poca potencia entre drenador y puerta. Cuando se cierra el interruptor se alimenta directamente al ESC y al regulador de tensión.

Adicionalmente, a continuación del MOSFET se encuentra un divisor de tensión para poder medir la tensión de la batería a través del conversor analógico digital (ADC) del microcontrolador.

### Regulador a 3.3V

La electrónica digital de la PCB se alimenta y emplea lógica a 3.3 Voltios, por lo que no la podemos conectar a las baterías de 11.1 Voltios. Para adecuar la tensión se ha escogido un regulador Step-down de tipo Buck (Figura 4.11) ([¿explico como funciona un convertidor Buck?](#)). El circuito integrado que se encarga de conmutar la fuente es el chip AP3211.

#### 4.5.2. El microcontrolador (ESP32)

El microcontrolador por el que se ha optado para este Autopiloto es el ESP32, un microcontrolador de doble núcleo con dos CPUs XTensaL6 con arquitectura Harvard [7]. El ESP32 tiene una frecuencia de reloj de hasta 240MHz ,y cuenta con una antena WiFi a 2,4 GHz y conexión Bluetooth 4.2 BLE [8]. Los motivos por los que se ha decidido emplear este microcontrolador son:

- Elevada frecuencia de procesamiento y dos nucleos de procesamiento.
- Antena WiFi incorporada.
- Bajo consumo de potencia.

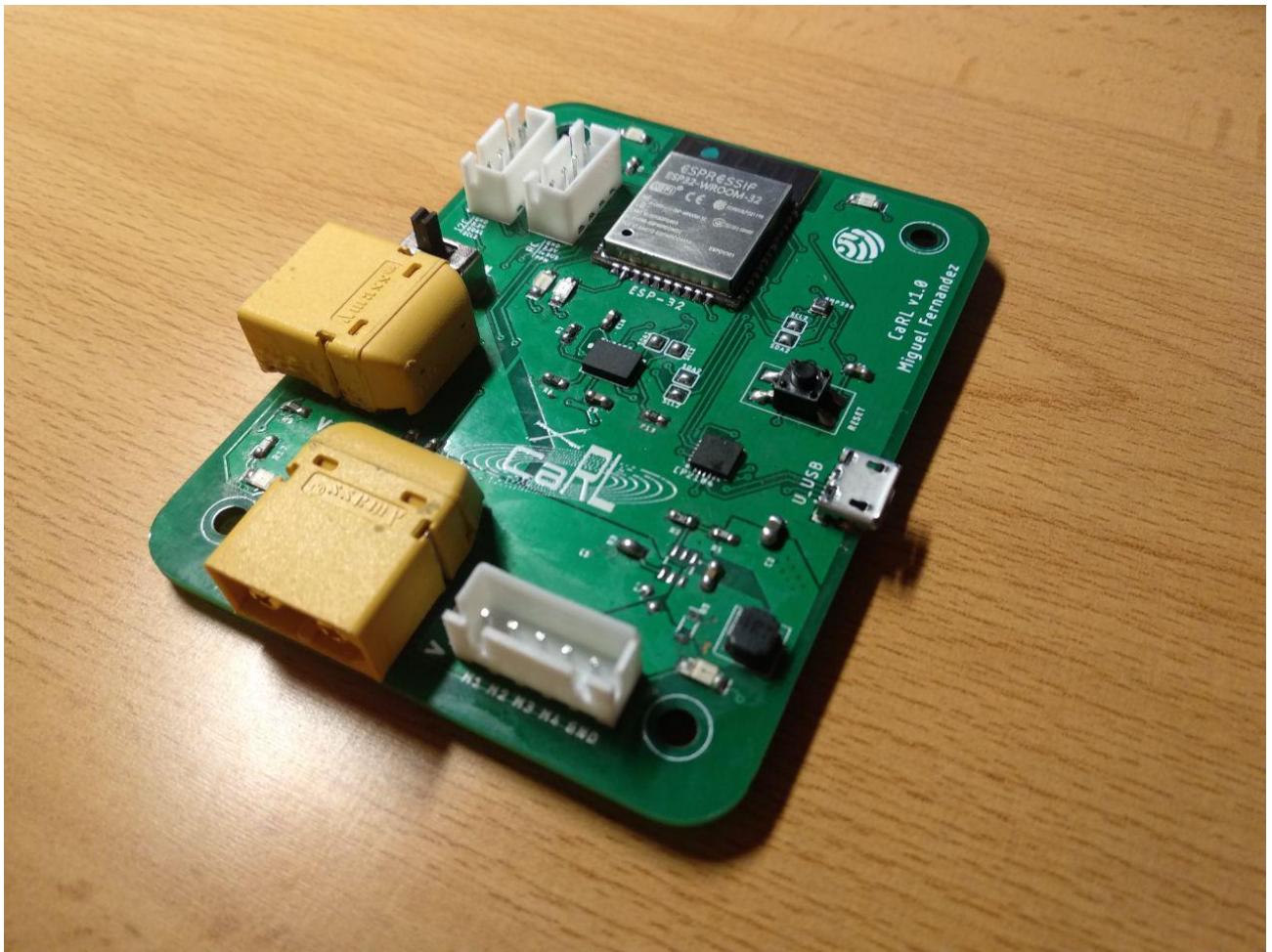


Figura 4.10: Autopiloto CaRL (*Cuadcopter with autopilot based on Reinforcement Learning*).

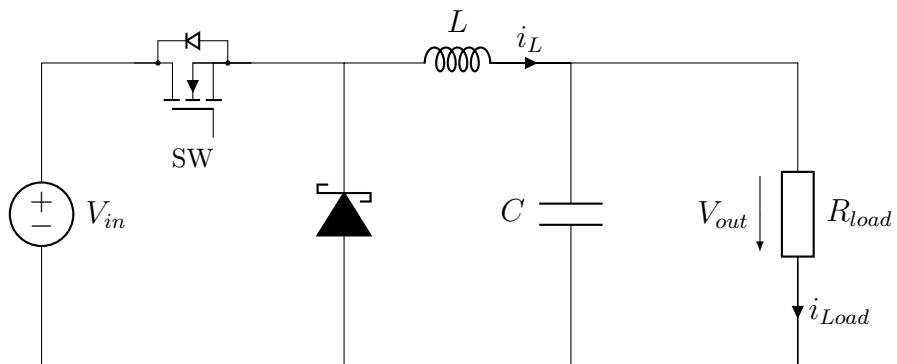


Figura 4.11: Esquema de un convertidor Buck

Para poder programar el microcontrolador se utiliza un convertidor USB (Bus Serie Universal) a UART (Transmisor-Receptor Asíncrono Universal) que permite conectar por USB el microcontrolador para poder programarlo y hacer depuración utilizando comunicaciones Serial. El chip que realiza esta función es el CP2104.

### 4.5.3. Sensores

La principal fuente de información procedente del exterior que recibe una controladora de vuelo se la proporcionan las unidades de medición inercial (IMU). Las IMUs son dispositivos electrónicos que son capaces de medir aceleraciones, velocidades y detectar la orientación de un sistema. El principal problema de estos sensores es que sufren error acumulativo a la hora de estimar posición y velocidad. Para corregir este error acumulativo en los drones, se suelen fusionar estas medidas con otras provenientes de mediciones absolutas tales como GPS o Láser, aunque en este autopiloto únicamente emplearemos las medidas de las IMUs.

Otros sensores utilizados frecuentemente en los autopilotos son brújulas (se encuentran integrados en la IMU para corregir errores de orientación) y barómetros (para estimar la altitud a la que se encuentra el dron).

El autopiloto cuenta con dos IMUs de 9 Grados de Libertad y un barómetro para conseguir una mejor estimación del estado del cuadricóptero:

1. **BNO 055 (BOSCH)**: El circuito integrado de Bosch es un sensor “inteligente” que incluye los sensores y la fusión de las lecturas de los distintos sensores en un único componente. Este encapsulado cuenta con: un acelerómetro, un giróscopo y un magnetómetro triaxial. Además integra un microcontrolador de 32 bits en el que se ejecuta el algoritmo de fusión integrado. El sensor se encuentra en un encapsulado LGA de 28 pines con una huella (*footprint*) de  $3,8 \times 5,2 \text{ mm}^2$ .

Este sensor nos proporciona estimaciones del estado completo de la aeronave con una frecuencia de refresco de 100Hz.

2. **MPU 9250 (TDK InvenSense)**: El sensor inercial de TDK es un módulo multi-chip compuesto por un MPU6050 (Contiene un acelerómetro y un giróscopo triaxial con un *procesador digital de movimiento* (DMP)) y un AK8963 (un magnetómetro digital triaxial). El sensor posee un encapsulado QFN de  $3 \times 3 \times 1 \text{ mm}$  con 24 pines.

Este dispositivo nos proporciona medidas del acelerómetro y el giróscopo a una frecuencia superior al BNO055 pero con una estimación peor de los ángulos que el dispositivo de BOSCH.

3. **BMP388 (BOSCH)**: El BMP388 es un barómetro digital de 24 bits con bajo consumo y bajo ruido. Se encuentra en un encapsulado LGA de 10 pines de dimensiones  $2 \times 2 \times 0,75 \text{ mm}^3$ .

Aunque el autopiloto cuenta con este sensor, éste no se ha utilizado en este trabajo debido a que no se tiene en cuenta la altitud en los controladores.

## 4.6. Banco de pruebas

Para poder realizar la experimentación real de forma segura, se han diseñado distintas estructuras para poder sujetar al cuadricóptero permitiéndole rotar con distintos grados de libertad (GdL) en función de la estructura. Estas uniones han permitido poder probar distintos controladores de forma segura y controlada.

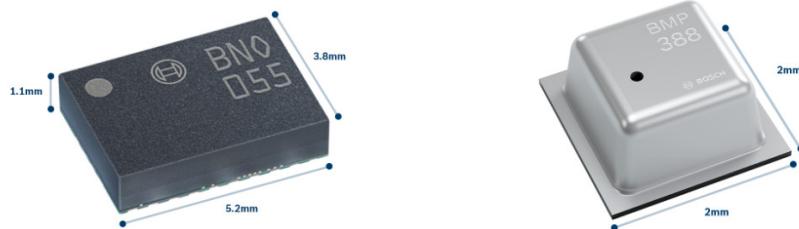


Figura 4.12: Sensores BNO055 y BMP388 respectivamente

Se pueden distinguir 2 tipos de estructuras en función de sus grados de libertad:

- **Rótulas con 1 único grado de libertad:** Para las primeras pruebas de los reguladores es fundamental poder descomponer el control en problemas más sencillos, en este caso permitiendo al sistema rotar en 1 grado de libertad. Se han construido 2 versiones de la misma rótula, una permite el movimiento en Pitch y la otra lo permite en Roll. Esto permite desacoplar los bucles de control y poder probar distintos algoritmos de control.



(a) Motores LHI MT2204 II empleados



(b) ESC Multistar Race 4 in 1 30A BLHeli empleado

Estas uniones son sencillas y robustas lo que nos da seguridad a la hora de poder probar y ajustar los reguladores. Las restricciones de movimiento de estas uniones permiten rotaciones de  $\pm 60^\circ$  en el angulo permitido.

- **Rótula con múltiples grados de libertad:** Después de conseguir estabilizar al cuadricóptero en pitch y en roll de forma individual la siguiente aproximación consiste en emplear rotulas con 3 GdL. Sobre estas uniones también se han realizado 2 versiones. La primera consta de una única rotula con sus 3 grados de libertad con restricciones de movimiento de:

$$\begin{aligned} -60^\circ \leq \varphi \leq 60^\circ & \quad \text{Roll} \\ -60^\circ \leq \theta \leq 60^\circ & \quad \text{Pitch} \\ -180^\circ \leq \psi \leq 180^\circ & \quad \text{Yaw} \end{aligned}$$

La segunda consta de acoplar 2 juntas esféricas como las anteriores una a continuación de la otra, lo que permite, además de disminuir las restricciones de movimiento en las rotaciones, pequeños desplazamientos en el espacio tridimensional.



(a) Motores LHI MT2204 II empleados



(b) ESC Multistar Race 4 in 1 30A BLHeli empleado

imagen rotulas y/o CAD

# Software

Durante el transcurso de este trabajo se ha dividido el desarrollo del software en varias partes, en las cuales se profundizará a continuación.

## 5.1. Software del Autopiloto

El autopiloto es la parte del multirrotor que se encarga de generar las acciones de control o comandos, que permitan que el dron llegue a un determinado estado. Para generar esas acciones de control, el autopiloto estima el estado de la aeronave en cada instante mediante las lecturas de las IMUs y en función del estado genera las acciones de control pertinentes para mantener la estabilidad de la aeronave. Además el autopiloto que se ha desarrollado cuenta con un interfaz WiFi, por lo que permite enviar y recibir datos del autopiloto a una estación de tierra. A continuación se profundizará en como se han llevado a cabo estas tareas.

### 5.1.1. Estimación del Estado

Para estimar el estado, el autopiloto toma las medidas procedentes de las 2 IMUs a través del protocolo I2C con una frecuencia de comunicación de 400KHz. Las lecturas del BNO055 proporcionan un estado completo de la aeronave con una frecuencia de refresco de 100Hz. Sin embargo, el MPU9250 nos proporciona medidas relativas a las velocidades angulares y aceleraciones lineales con una tasa de refresco más elevada. Es por esto que se han empleado ambos sensores para conseguir el estado deseado con la mayor precisión y menor tasa de refresco posible.

Con este método se consigue estimar el estado deseado:

$$S_t = (\underbrace{\varphi, \theta, \psi}_{\text{BNO}}, \underbrace{\dot{\varphi}, \dot{\theta}, \dot{\psi}}_{\text{MPU}}) \quad (5.1)$$

consiguiendo minimizar la deriva de la estimación, empleando las lecturas muy estables del BNO055, sin perder la reactividad de las medidas que proporciona el MPU9250.

### 5.1.2. Interfaz WiFi

Con la intención de poder transmitir datos entre el autopiloto y la estación de tierra se ha diseñado un sencillo protocolo de comunicación WiFi basado en el envío de paquetes de tamaño fijo. El ESP32 cuenta con un sistema operativo en tiempo real (RTOS) el cual se encarga de mantener activa la comunicación WiFi de forma transparente al usuario. La estructura de los paquetes es la siguiente:

- **Autopiloto → Estación:** Este paquete contiene 6 datos de tipo *float* (32-bits). Se emplea para enviar la estimación del estado a la aeronave en tiempo real. Esto se emplea para tanto monitorización del desempeño de los algoritmos, como para poder cerrar el bucle de control en la estación base y enviar las acciones de control a la aeronave posteriormente.
- **Autopiloto ← Estación:** Este paquete contiene 4 datos de tipo *float* (32-bits) y un dato de tipo *byte*. El dato de tipo *byte* se emplea para controlar el modo de funcionamiento de la aeronave, mientras que los otros 4 datos tienen distintos usos en función del modo:
  - **MODO 0 (Desarmado):** La aeronave se encuentra “desarmada” por lo que los motores no reciben acciones de control. En este modo los demás datos del paquete son irrelevantes.
  - **MODO 1 (*Off-board*):** Las acciones de control son emitidas por la estación de tierra, es decir, la aeronave envía la estimación del estado al ordenador y envía el comando recibido a los variadores. En este modo los datos de tipo *float* del paquete representan los comandos de los motores.
  - **MODO 2 (*On-board*):** El algoritmo de control se ejecuta en la aeronave y se generan las señales de control necesarias a una frecuencia fijada previamente. En el caso de los controladores PID a bordo, los valores de los 4 datos del paquete permiten variar los valores de las constantes del regulador en tiempo real, lo que facilita el ajuste de las mismas.

### 5.1.3. Generacion de comandos *on-board*

Para poder maximizar la frecuencia del algoritmo de control y permitir el control de la aeronave sin necesidad de una estación de tierra, los algoritmos de control validados en la simulación se pueden implementar directamente en el autopiloto, fijando previamente la frecuencia a la que se desea que se ejecute el algoritmo. [actualizar en el momento](#) En este trabajo se han implementado a bordo dos algoritmos de control distintos: un regulador PID clásico y un regulador PID en cascada.

## 5.2. Software de la estación

La estación base se ha empleado tanto como para la simulación del entorno, permitiendo diseñar y probar algoritmos de control de forma segura, como para monitorizar los rendimientos de estos algoritmos o incluso, aprovechando la mayor capacidad computacional de estas estaciones, ejecutar los algoritmos de control y aprendizaje por refuerzo en estas. A continuación se profundizará con más detalle en la implementación de estas herramientas.

### 5.2.1. Entorno de simulación

En cualquier proyecto de robótica es conveniente contar con un entorno de simulación que permita realizar pruebas en distintas situación de forma sencilla y rápida, sin necesidad de tener ni modelo real, ni el entorno concreto en el que quieras probar tu robot. Cuando se trabaja con drones, la simulación pasa de ser conveniente a ser muy necesaria. Ésto

es debido a la peligrosidad intrínseca de estas aeronaves, ya que, cuentan con hélices que giran a gran velocidad.

Debido a la naturaleza del aprendizaje por refuerzo, este requiere de la interacción de un agente con un entorno para que el agente aprenda que acciones son las que debe tomar en cada estado. Esto significa que al comienzo del entrenamiento el agente realiza acciones aleatorias para poder explorar cuales son las que le proporcionan una recompensa mayor.

Debido a esta forma de explorar, el agente requiere de una gran cantidad de pruebas, de ensayo y error, hasta que consigue aprender, por lo que no es conveniente realizar todas estas iteraciones en un modelo real.

Por estas razones se necesita de un entorno de simulación para poder validar los algoritmos y poder generar modelos entrenados en simulación sin deteriorar el equipo real. Además el entorno en simulación te permite entrenar distintos agentes simultáneamente y reducir los tiempos de entrenamiento.

Para el entorno de simulación nos hemos basado en Gym [citar](#),una librería escrita en Python y desarrollada por la compañía OpenAI, que permite desarrollar y comparar algoritmos de aprendizaje por refuerzo. Esta librería te permite generar un entorno con el cual interactúe el agente, sin importar la implementación de éste, permitiendo comparar el rendimiento de distintos agentes sobre el mismo entorno. [IMAGEN GYM](#)

[revisar](#) Como entorno de simulación se ha partido de GymFC, desarrollado por William Koch et al. [5]. Este entorno de simulación utiliza Gazebo 9, un entorno de simulación 3D de código abierto ampliamente utilizado en el campo de la robótica. En este entorno se simula el comportamiento de un multirrotor, concretamente el de un modelo del cuadricóptero IRIS. [imagen GYMFC](#)

Para acercar la simulación a la configuración de los experimentos que se realizarían posteriormente en la plataforma real, se han realizado algunas modificaciones sobre el modelo predeterminado de la aeronave:

1. Se ha modificado la forma de anclaje del cuadricóptero. Originalmente el dron se encontraba anclado entorno a una articulación situada en su centro de gravedad. Esto permitía que el peso de la aeronave apenas tuviera influencia a la hora de rotar la aeronave, todo el peso estaba sustentado por la articulación, lo cual no se asemeja con los bancos de pruebas que se han diseñado. En estos bancos de prueba el anclaje se encuentra desplazado con respecto al centro de gravedad, por lo que cuanto mayor sea el valor de los ángulos  $\varphi$  y  $\theta$  mayor es la influencia del peso en el par necesario para estabilizar la aeronave. Es por esto que se ha desplazado la articulación del centro de gravedad y se ha conectado a la aeronave mediante una unión rígida, asemejando así el entorno de simulación al entorno de pruebas real.
2. Se ha modificado parámetros dinámicos de la aeronave. Los parámetros que mayor discrepancia tenían entre la simulación y el mundo real eran principalmente los parámetros iniciales. Tanto la masa como los momentos de inercia del cuadricóptero eran mucho inferiores a los de la aeronave real. Esto se manifestaba principalmente cuando al hacer pruebas con los algoritmos PID el comportamiento simulado no se parecía con el real, por ejemplo en simulación un regulador P era capaz de estabilizar al sistema, mientras que en la realidad un regulador P era inestable de por sí y requería de las acciones derivativa e integral para conseguir estabilizarlo.

### 5.2.2. Agente (generación de comandos)

Siguiendo con la terminología del aprendizaje por refuerzo se va a denominar agente al encargado de generar las acciones de control. Dependiendo del algoritmo de control que se use, se dispone de distintos tipos de agente.

#### Agente con algoritmos PID

En este caso, el agente recibe las observaciones del entorno a través del estado y en función de éste, produce la salida a los motores. Este agente es el más sencillo, ya que, una vez ajustadas las ganancias, éste solamente debe enviar los comandos de control a la aeronave.

#### Agente con algoritmos de RL

Este agente, se encarga, tanto de generar las acciones como de optimizar la política que las genera.

El agente, basado en el estado observado  $s_t$  y la política  $\pi$  actual, genera una acción  $a_t$ , tras realizarla, el entorno le proporciona un estado  $s_{t+1}$  y una recompensa  $r_t$ . Con esta tupla  $[s_t, a_t, r_t, s_{t+1}]$  el agente emplea el algoritmo de aprendizaje por refuerzo definido para actualizar la política  $\pi$  y así intentar mejorarla con cada iteración.

Para probar distintos algoritmos se han empleado la implementación de los algoritmos del estado del arte, del repositorio de GitHub *Stable-Baselines* [9]. Inicialmente se comenzó empleando las *Baselines* de OpenAI [10] pero se decidió cambiar a las *Stable Baselines* por la limpieza del código y la facilidad para realizar experimentos con diversos algoritmos e hiperparámetros. Con esta herramienta se ha podido experimentar con distintos algoritmos para comparar el rendimiento conseguido de cada uno.

### 5.2.3. Interfaz Estación-Autopiloto

Para las pruebas reales se ha empleado ROS (Robotic Operative System) Melodic para gestionar la comunicación WiFi con la aeronave y la comunicación con el agente. Para esto se emplea la estructura de *topics*: un *topic* se emplea para enviar datos a la aeronave y otro *topic* publica los datos recibidos por ésta. De esta manera se pueden utilizar las herramientas de ROS para poder monitorizar el proceso además de aprovechar la estructura de *topics* para poder integrar código escrito en distintos lenguajes de forma sencilla.

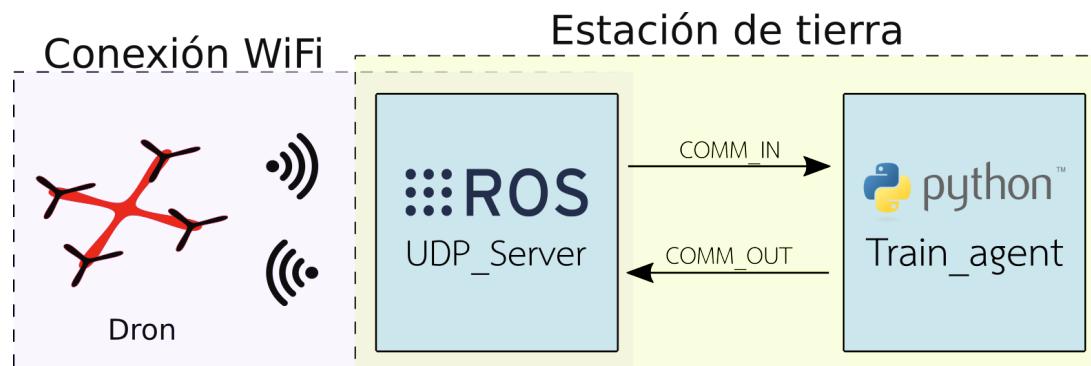


Figura 5.1: Esquema interfaz Estacion-Autopiloto

### **5.3. Descripción del equipo**

Para el desarrollo del trabajo se ha empleado un portatil MSI-GE62 empleando Windows 10 para el desarrollo CAD y Ubuntu 18.04 LTS para el resto de las tareas. El equipo cuenta con 16GB de RAM DDR4, un procesador Intel i7-6700HQ de 8 núcleos a 2.60GHz y una GPU GeForce GTX 970M con 3GB de memoria dedicada.

# Metodología (Problem Formulation)

Además del desarrollo de la plataforma de la aeronave, otro objetivo del trabajo es intentar estabilizar un UAV usando algoritmos de control basados en algoritmos de aprendizaje automático. Los principales componentes que intervienen en el agente son el estado, las acciones y la recompensa, para cada problema hay un conjunto que estados, acciones y funciones de recompensa que pueden llevar a que el agente aprenda.

## 6.1. Diseño del estado

El autopiloto cuenta con 2 IMUs para poder obtener datos sobre su estado. Se quiere estabilizar el dron en una orientación concreta, por lo tanto el estado que se ha diseñado consta de 6 parámetros:

$$S = (\varphi, \theta, \psi, \dot{\varphi}, \dot{\theta}, \dot{\psi}) \quad \varphi, \theta, \psi, \dot{\varphi}, \dot{\theta}, \dot{\psi} \in [-1, 1] \quad (6.1)$$

Siendo  $\varphi, \theta$  y  $\psi$  los ángulos de alabeo (*roll*), cabezeo (*pitch*) y guiñada (*yaw*) del dron y  $\dot{\varphi}, \dot{\theta}$  y  $\dot{\psi}$  sus respectivas velocidades. Para favorecer la convergencia del aprendizaje, se ha normalizado el estado para que todas sus componentes estén comprendidas dentro del intervalo  $[-1, 1]$ .

Los ángulos proporcionan información sobre el estado actual y la velocidad angular sobre los estados pasados y los posibles estados futuros, es decir, proporciona cierta información temporal.

## 6.2. Diseño de las acciones

Al trabajar con un quadricóptero podemos actuar sobre la potencia que se le entrega a los motores, por lo que cada acción que realice el agente constará de 4 campos:

$$A = (T_1, T_2, T_3, T_4) \quad T_i \in [-1, 1] \quad (6.2)$$

Siendo  $T_i$  la potencia (*Thrust*) normalizada entregada a cada motor. Un valor de  $T_1 = -1$  significa que el motor 1 estaría girando a la mínima potencia permitida y un valor de  $T_1 = 1$  corresponde a que el motor 1 estaría girando a la máxima potencia.

Las señales de control que admiten los variadores es una señal PWM de 50Hz cuyo ancho de pulso debe estar entre 1ms y 2ms. Un ancho de pulso de 1ms se corresponde con la mínima velocidad el motor y un ancho de 2ms con la máxima, véase fig. 6.1.

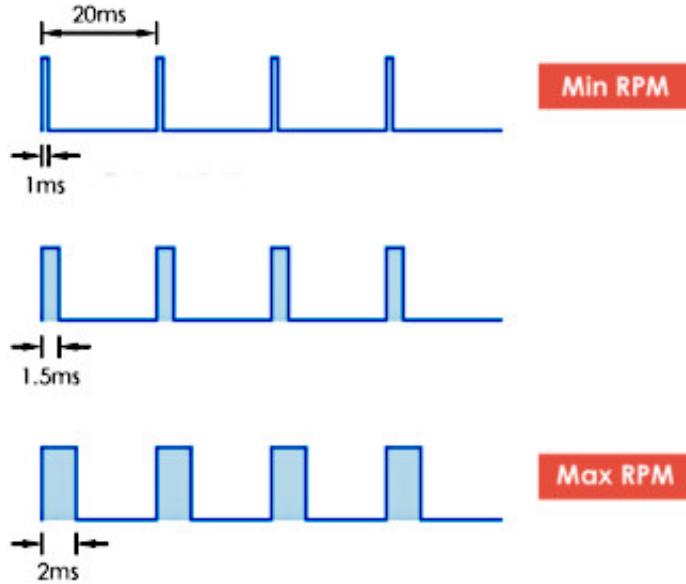


Figura 6.1: Formas de onda que recibe un variador

Para realizar esta transformación del dominio  $[-1, 1]$  al dominio  $[1000, 2000] \mu\text{s}$  se han empleado la siguiente expresión

$$P_i = T_i \cdot 1000 \cdot \alpha + \beta \quad i = 0, \dots, 3 \quad (6.3)$$

siendo  $P_i$  el ancho del pulso enviado al ESC,  $\alpha \in [0, 1]$  es el porcentaje de la potencia total que puede manejar el controlador y  $\beta \in [0, 2000]$  es la velocidad base que tienen los motores. Esta velocidad base permite que el dron pueda variar su altura y que se pueda autosustentar.

### 6.3. Diseño de la función de recompensa y ajuste de hiperparámetros

La búsqueda de la función de recompensa y la elección de los hiperparámetros que aseguraban la convergencia de los distintos algoritmos, se ha realizado mediante un proceso cíclico, en el que, se realizaba una hipótesis sobre la posible forma de la función de recompensa o el posible valor de un hiperparámetro que podría mejorar el algoritmo. Posteriormente se ejecutaba el algoritmo y se comparaban los resultados de este entrenamiento con los resultados previos. Para disminuir el tiempo de los entrenamientos, se simplificaba el problema del control del cuadricóptero a un único eje de giro. Con esto se redujo el tiempo que se tardaba en completar el ciclo.

[diagrama ciclo](#)

#### Función de recompensa

La función de recompensa rige la forma en la que la red va a configurar sus pesos, por lo tanto, cómo se va a comportar el agente en un estado determinado. Para conseguir que el agente responda de la forma deseada se han probado una gran variedad de funciones

de *reward*, optando finalmente por:

$$R_t = \left(1 - \frac{|\varphi| + |\theta| + |\psi|}{3}\right)^n \quad (6.4)$$

Sea  $\gamma = \frac{|\varphi| + |\theta| + |\psi|}{3}$  entonces  $R_t = (1 - \gamma)^n$ , con  $n \in \mathbb{N}$ . Aumentando el valor de  $n$  podemos conseguir funciones de recompensa con pendientes más pronunciadas.

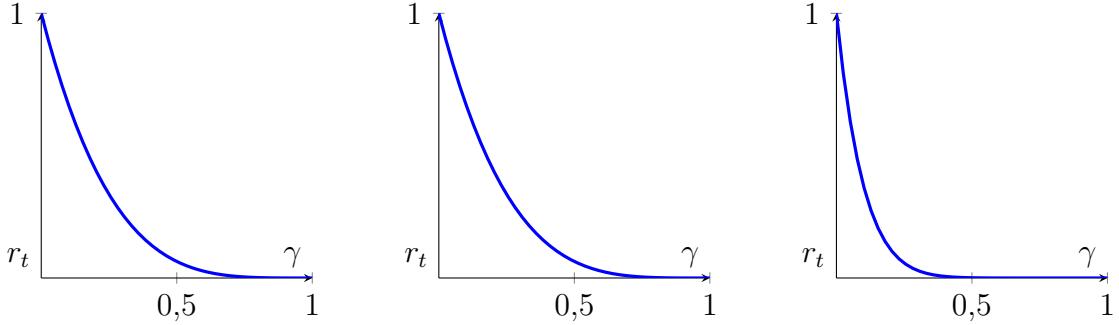


Figura 6.2: Funciones  $R_t$  para distintos valores de  $n = 2, 4, 10$  respectivamente

Esta función  $R_t : [0, 1] \rightarrow [0, 1]$   $\forall n \in \mathbb{N} > 0$  por lo que al proporcionar valores entre 0 y 1, favorece la velocidad del entrenamiento. Además toda la recompensa es positiva, por lo que el agente intentará mantenerse la mayor cantidad de tiempo posible sin reiniciar el episodio, esto es muy importante para conseguir la convergencia del entrenamiento en los escenarios en los que se desea que el cuadricóptero se encuentre siempre dentro de una región concreta del espacio.

### Ajuste de hiperparámetros

Debido a las diferentes naturalezas de cada uno de cada uno de los algoritmos que se han empleado: DDPG, TRPO y PPO , cada uno cuenta con un conjunto distinto de hiperparámetros que ajustar. La implementación de estos algoritmos que se ha empleado (*stable-baselines*) contiene unos hiperparámetros por defecto, los cuales suelen hacer que los algoritmos converjan.

Por lo general, no ha sido necesario modificar mucho estos parámetros por defecto, a excepción del valor de la constante  $\epsilon$  en el algoritmo PPO. Este parámetro modifica el tamaño de la región de confianza del algoritmo, acotando el tamaño de los pasos que se toman en el curso de cada actualización de la política. Se observó que el valor por defecto de este parámetro  $\epsilon_{defecto} = 0,2$  era demasiado grande, por lo que la política divergía. A medida que este parámetro se disminuye, se aumenta la estabilidad del aprendizaje, sin embargo, también se ralentiza mucho. Es por esto que el valor de éste hiperparámetro se ha ido modificando a lo largo de los experimentos, para poder conseguir el entrenamiento más rápido, capaz de converger de forma estable.

# Experimentos

Debido a que el trabajo tiene dos partes principales claramente diferenciadas, se han diseñado distintos tipos de experimentos para la evaluación de cada parte: un conjunto de experimentos en simulación para probar el rendimiento de los algoritmos de aprendizaje por refuerzo y un conjunto de experimentos en real para probar la plataforma hardware y el diseño del autopiloto.

## 7.1. Experimentos en simulación

Con el ánimo de probar los distintos algoritmos de control basados en aprendizaje por refuerzo, se han diseñado diversos experimentos para comparar el rendimiento de estos algoritmos con el de un PID clásico.

### Control en 1 GdL (*Roll*)

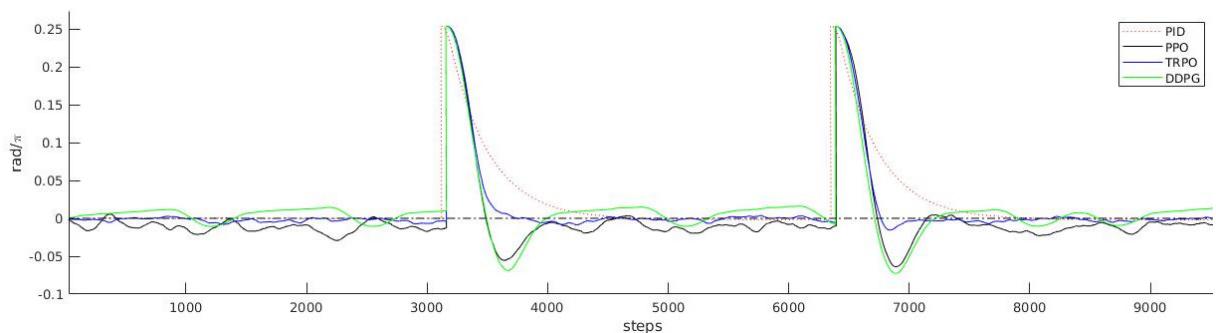


Figura 7.1: Estabilización en *roll* en un entorno simulado.

### Control en 1 GdL (*Pitch*)

### Control en los 3 GdL

## 7.2. Experimentos en real

Para la evaluación de la plataforma de vuelo, el diseño del cuadricóptero y del autopiloto, se han realizado distintos experimentos para intentar controlar la aeronave con algoritmos PID en cascada a una frecuencia de 70Hz.

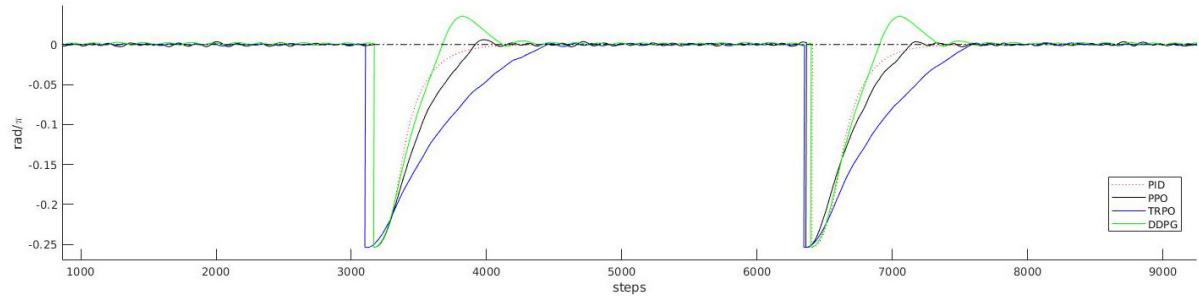


Figura 7.2: Estabilización en *pitch* en un entorno simulado.

### Control en 1 GdL (*Pitch*)

En este experimento se ha empleado la rótula de 1 GdL que únicamente permite el movimiento en *pitch*.

Se observa que el sistema es capaz de estabilizarse y que reacciona ante las perturbaciones externas de forma ágil. Por otro lado, también podemos observar ruido en la zona estable, además de poseer un pequeño error de posición de unos  $2^\circ$ .

### Control en 1 Gdl (*Roll*)

En este experimento se ha empleado la rótula de 1 GdL que únicamente permite el movimiento en *roll*.

El comportamiento observado es similar al del experimento anterior: el sistema se estabiliza y consigue llegar a la referencia después de perturbarlo de una forma ágil, pero también se observa ruido y un pequeño error de posición en el régimen permanente.

### Control en los 3 Gdl

En este experimento se ha empleado la rótula de 1 GdL que permite el movimiento en *roll*, *pitch* y *yaw*.

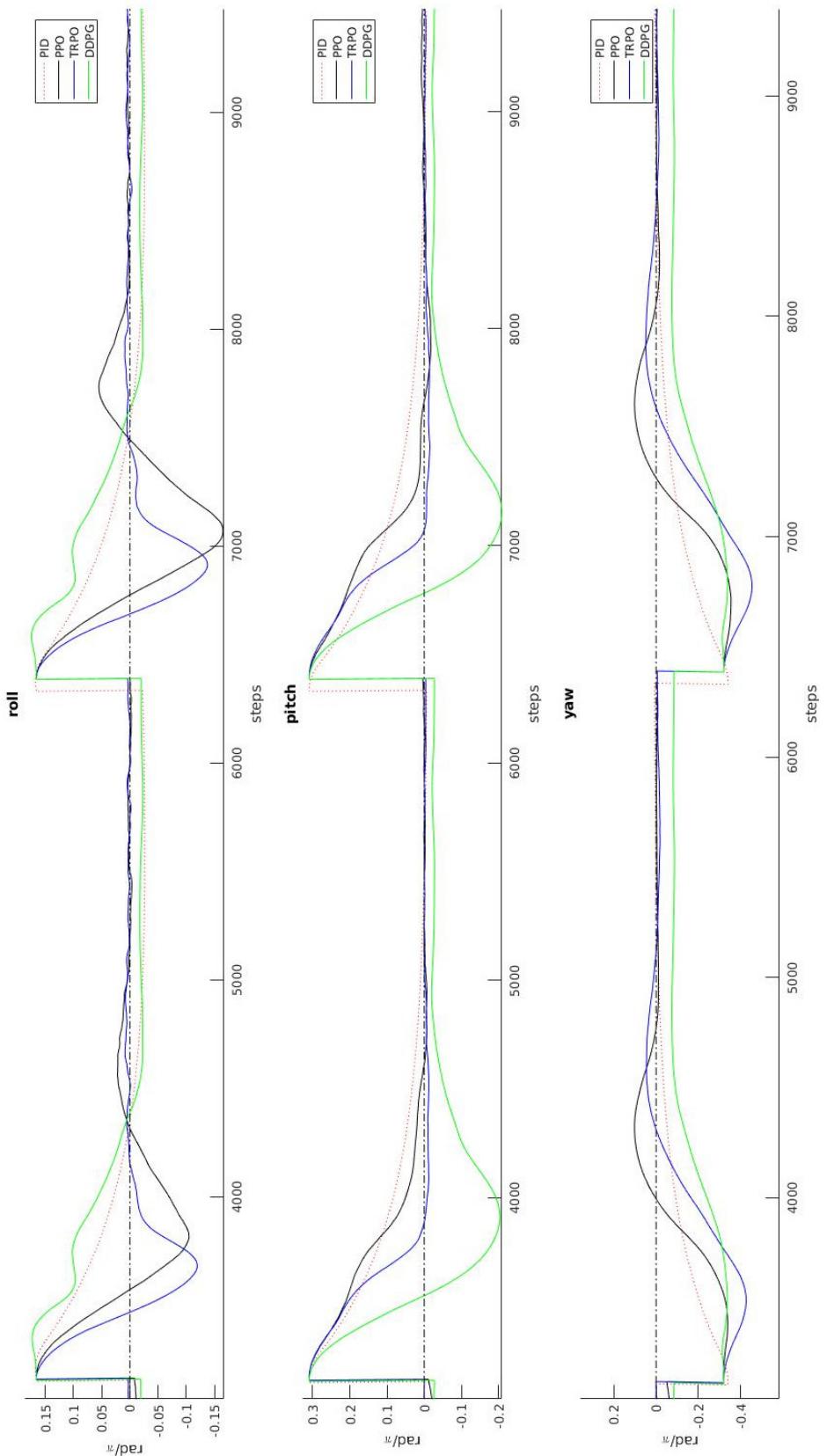


Figura 7.3: Estabilización en *roll*, *pitch* y *yaw* simultáneamente

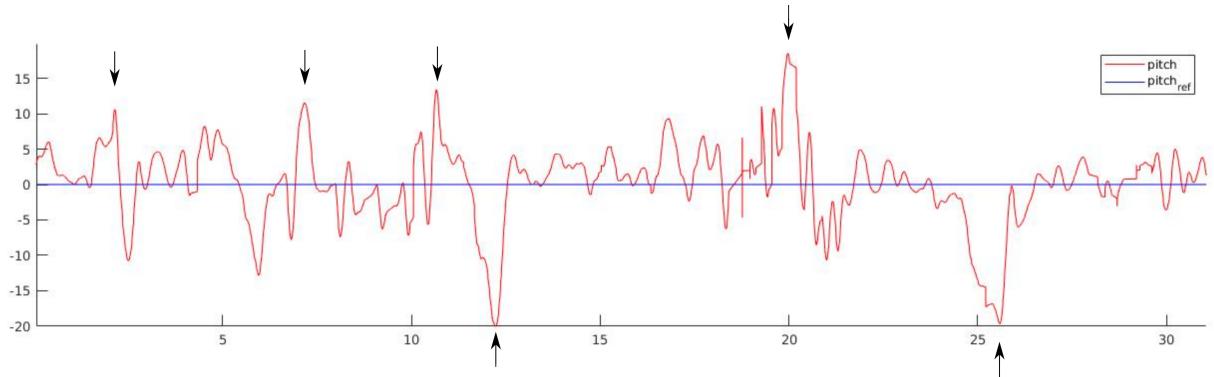


Figura 7.4: Estabilización en *pitch*. Las flechas marcan los tiempos en los que se perturbó al sistema.

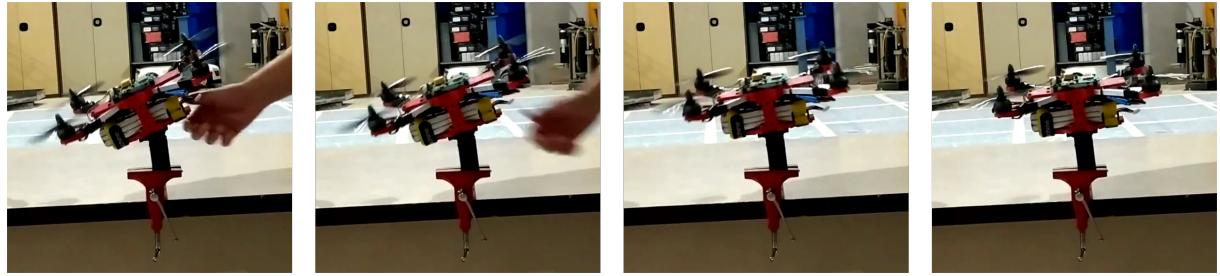


Figura 7.5: Estabilización en *pitch*

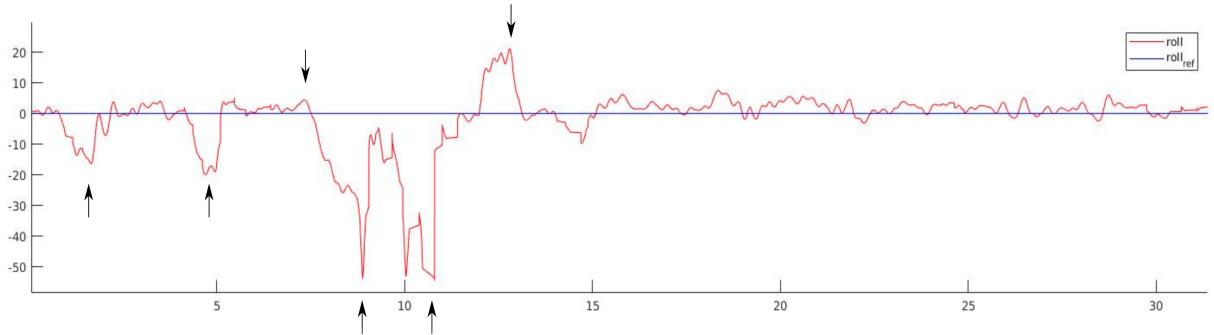


Figura 7.6: Estabilización en *roll*. Las flechas marcan los tiempos en los que se perturbó al sistema.

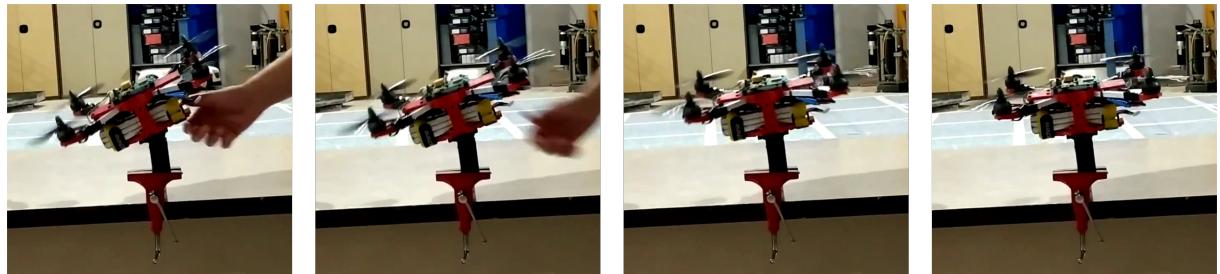


Figura 7.7: cambiar

En este experimento observamos que aumenta el ruido que encontramos en el régimen permanente en *pitch* y en *roll*, aunque se disminuye el error de posición que veíamos en los experimentos anteriores. Por otro lado el controlador de *yaw* es el menos ruidoso, pero

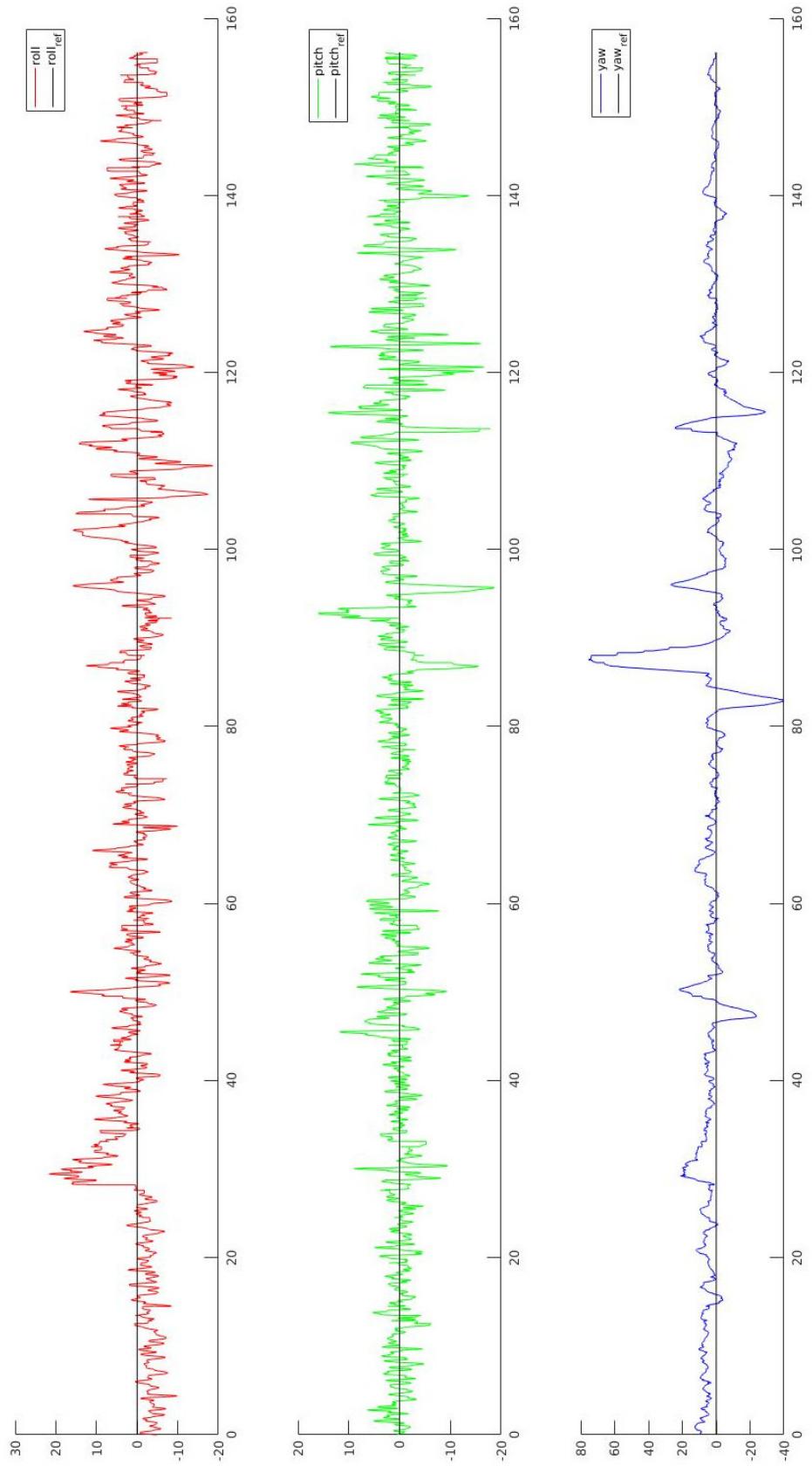


Figura 7.8: Estabilización en *roll*, *pitch* y *yaw* simultáneamente

también el más lento.

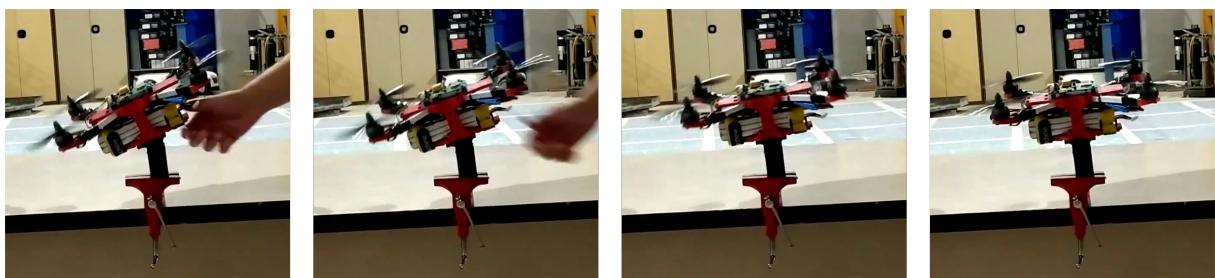


Figura 7.9: [cambiar](#)

# Discusión

# Conclusiones y trabajo futuro

Durante el transcurso de este proyecto se ha conseguido desarrollar una plataforma, que permite estudiar los distintos algoritmos de control, con los que se puede estabilizar a un cuadricóptero. A continuación se hablará sobre las conclusiones que se han extraído durante este proceso y las posibles líneas de mejora que permite

## Índice de figuras

1.1	Esquema cuadrirrotor en X . . . . .	2
1.2	Autopiloto comercial Pixhawk 4 . . . . .	3
2.1	Esquema estructura gimbal . . . . .	6
2.2	Plataforma FFT Gyro de Eureka Dynamics . . . . .	7
2.3	Esquema de una union esférica . . . . .	8
2.4	Plataforma 3 DOF Hover de la empresa Quanser . . . . .	8
3.1	Bucle de control realimentado con regulador PID . . . . .	10
3.2	Bucle de control en cascada . . . . .	10
3.3	Esquema cuadrirrotor en X . . . . .	11
3.4	Esquema de una red neuronal artificial . . . . .	12
3.5	Esquema de un perceptrón . . . . .	12
3.6	Función sigmoide . . . . .	13
3.7	Función tangente hiperbólica . . . . .	13
3.8	Función ReLu . . . . .	14
3.9	Diagrama canónico del bucle de interacción entorno-agente . . . . .	15
3.10	Evolución de una función límite inferior con un algoritmo MM. . . . .	20
3.11	Gráficas de la función objetivo $L^{\text{CLIP}}(\theta)$ para distintos valores de $r$ , dependiendo del signo de la ventaja $A$ . . . . .	20
4.1	Cuadricóptero diseñado con el autopiloto incoporado. . . . .	21
4.2	Cuadricóptero diseñado con el autopiloto incoporado. . . . .	22
4.3	Motores LHI MT2204 II empleados . . . . .	23
4.4	Tabla de especificaciones motor MT2204 II. . . . .	23
4.5	Hélices tripala HQ5040 . . . . .	23
4.6	Funcionamiento ESC ( <a href="http://www.hwtomechatronics.com">www.hwtomechatronics.com</a> ) . . . . .	24
4.7	ESC Multistar Race 4 in 1 30A BLHeli empleado . . . . .	24

4.8	Batería LiPo 3s 35C 5200 mah de la marca NZACE empleada . . . . .	25
4.9	PCB autopiloto CaRL, anverso y reverso. . . . .	26
4.10	Autopiloto CaRL ( <i>Cuadcopter with autopilot based on Reinforcement Learning</i> ). . . . .	27
4.11	Esquema de un convertidor Buck . . . . .	27
4.12	Sensores BNO055 y BMP388 respectivamente . . . . .	29
5.1	Esquema interfaz Estacion-Autopiloto . . . . .	34
6.1	Formas de onda que recibe un variador . . . . .	37
6.2	Funciones $R_t$ para distintos valores de $n = 2, 4, 10$ respectivamente . . . . .	38
7.1	Estabilización en <i>roll</i> en un entorno simulado. . . . .	39
7.2	Estabilización en <i>roll</i> en un entorno simulado. . . . .	40
7.3	Estabilización en <i>roll, pitch y yaw</i> simultáneamente . . . . .	41
7.4	Estabilización en <i>pitch</i> . Las flechas marcan los tiempos en los que se perturbó al sistema. . . . .	42
7.5	Estabilización en <i>pitch</i> . . . . .	42
7.6	Estabilización en <i>roll</i> . Las flechas marcan los tiempos en los que se perturbó al sistema. . . . .	42
7.7	<b>cambiar</b> . . . . .	42
7.8	Estabilización en <i>roll, pitch y yaw</i> simultáneamente . . . . .	43
7.9	<b>cambiar</b> . . . . .	44

# Índice de tablas

# Bibliografía

- [1] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, “Autonomous helicopter flight via reinforcement learning,” in *Advances in neural information processing systems*, 2004, pp. 799–806.
- [2] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, “Autonomous inverted helicopter flight via reinforcement learning,” in *Experimental robotics IX*. Springer, 2006, pp. 363–372.
- [3] T. Dierks and S. Jagannathan, “Output feedback control of a quadrotor uav using neural networks,” *IEEE transactions on neural networks*, vol. 21, no. 1, pp. 50–66, 2010.
- [4] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.
- [5] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement learning for uav attitude control,” *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, p. 22, 2019.
- [6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [7] “Esp32 technical reference manual,” Espressif Systems, 2018. [Online]. Available: [https://www.espressif.com/sites/default/files/documentation/esp32\\_technical\\_reference\\_manual\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf)
- [8] “Esp32 datasheet,” Espressif Systems, 2019. [Online]. Available: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)
- [9] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines,” <https://github.com/hill-a/stable-baselines>, 2018.
- [10] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, “Openai baselines,” <https://github.com/openai/baselines>, 2017.