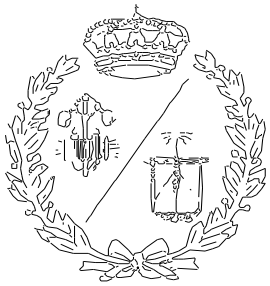
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

NAVEGACIÓN AUTÓNOMA DE DRONES BASADA EN DEEP REINFORCEMENT LEARNING

Autonomous drone navigation based on Deep Reinforcement Learning

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Aitor García Gómez

Septiembre - 2017

ÍNDICE GENERAL DEL PROYECTO

DOCUMENTO 1: MEMORIA

DOCUMENTO 2: ANEXOS

DOCUMENTO 1. MEMORIA

ÍNDICE DOCUMENTO 1. MEMORIA

INDICE DE FIGURAS

1. INTRODUCCIÓN Y OBJETIVOS

La inteligencia artificial está creando gran expectación por su capacidad para aumentar los niveles de automatización en muchas industrias. Además, estamos asistiendo al rápido desarrollo de los vehículos autónomos, especialmente en automóviles y en vehículos aéreos no tripulados, conocidos como drones o UAVs. Es la combinación de ambos elementos, la inteligencia artificial y la navegación autónoma de drones, lo que define el marco del presente proyecto.

Desde un punto de vista tecnológico, el desarrollo de sistemas de control de drones es complejo por, entre otros motivos, su modelo dinámico altamente influenciado por factores aerodinámicos difíciles de parametrizar.

Por otra parte, la aplicación de la inteligencia artificial a la robótica no es una tarea sencilla. Pasar de programar un robot mediante la definición de sus movimientos a “entrenarlo” mediante recompensas y penalizaciones electrónicas es un cambio de paradigma.

Se pretende, con este trabajo, contribuir al uso de la inteligencia artificial en la navegación autónoma de drones:

- Estudiaremos y extraeremos conclusiones sobre el uso de una novedosa tecnología, Deep Reinforcement Learning para construir sistemas capaces de controlar UAVs.
- Realizaremos el modelado dinámico de un vehículo aéreo no tripulado.

Un sistema de control basado en Deep Reinforcement Learning necesita ser parametrizado y ser expuesto a mucha experiencia de vuelo, lo que requiere gran cantidad de experimentos. La librería de Deep Reinforcement Learning utilizada para realizar los citados experimentos se denomina Primate, y es propiedad de la empresa Monkey from the Future, donde se ha realizado este trabajo.

1.1 MOTIVACIÓN

Uno de los aspectos que más me ha motivado para escoger este trabajo ha sido cómo conseguir resolver sistemas de control muy complejos aplicando técnicas vanguardistas.

También la posibilidad de formarme en el uso de redes neuronales y deep reinforcement learning, ya que cursé la especialidad en automática, aprendiendo a reconocer objetos mediante visión artificial.

Por otra parte, descubrir el potencial que tiene la inteligencia artificial y los distintos casos de uso me ha impulsado a continuar profundizando en este campo.

2. ESTUDIO FUNCIONAL DEL DRON

2.1 DEFINICIÓN

Un dron es una aeronave, con uno o varios motores, que no lleva a bordo tripulación. Es capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, utilizando las fuerzas aerodinámicas para generar la sustentación.

No se consideran drones los misiles balísticos, misiles crucero, proyectiles de artillería, planeadores (no usan propulsores), globos y dirigibles (se sustentan mediante fuerzas de flotabilidad) y objetos arriostrados (no tienen control remoto o autónomo). [1]

2.2 AVIACIÓN NO TRIPULADA A LO LARGO DE LA HISTORIA

La aviación no tripulada comenzó a lo largo de la primera mitad del siglo XIX con los primeros modelos construidos por inventores como Du Temple, Stringfellow, Cayley y otros exploradores de la aviación. [2]

A partir de las pruebas realizadas con estos vehículos se comenzaron a desarrollar modelos de mayor tamaño e incluso introdujeron un piloto a bordo para guiarlo.

Durante muchos años se utilizó el término vehículo aéreo pilotado remotamente (Remotely Piloted Vehicle, RPV). Es en los años 90 cuando comienza a hacerse común el nombre de vehículo aéreo no tripulado (Unmanned Aerial Vehicle, UAV) para describir a estas aeronaves robóticas.

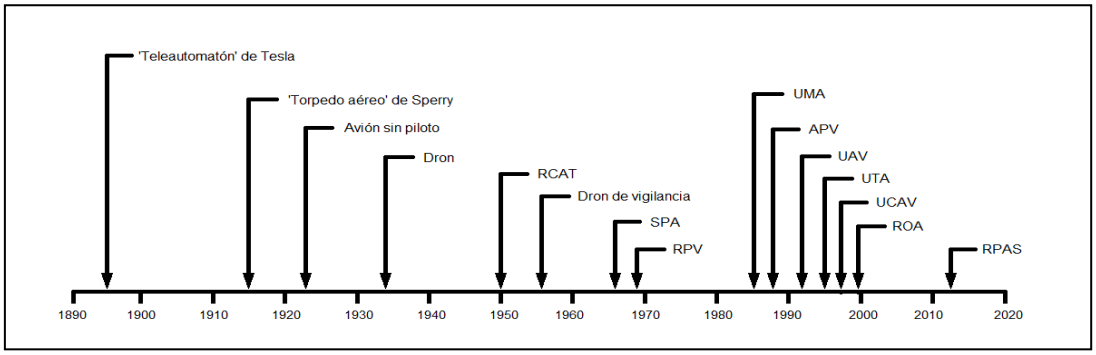


Figura 2-1: Diagrama temporal del dron

Los avances tecnológicos fueron dando impulso al desarrollo de los aviones no tripulados.

En 1972 se les equipó con tecnología de navegación de largo alcance, que mejoraron drásticamente su capacidad operativa.

En la década de los 2000 se le añadió un sistema de navegación más preciso, GPS, pudiendo volar más cerca del terreno sin problemas. También se introdujeron radares de apertura sintética, un tipo especial de radar que permite obtener imágenes de alta resolución a larga distancia.

En los últimos años el desarrollo de los sistemas de vuelo no tripulados (Drones) se está incrementando debido a la cantidad de aplicaciones. En la actualidad, son vehículos muy estables, la mayoría son simétricos y con varios motores.

A continuación se exponen las principales aplicaciones de los UAV. [4]

MISIONES MILITARES.

Dentro de aplicaciones militares destaca por encima de todo, la labor de espionaje, donde por sus características y reducido tamaño pasan inadvertidos para los radares.

Figura 2-2: Dron de uso militar

Por ejemplo:

- Han sido utilizados en Irak y Afganistán en multitud de acciones dirigidas contra Al Qaeda y misiones de escolta de convoyes militares, para espiar instalaciones militares, patrullar zonas a proteger y detectar situaciones de riesgo.
- Forman parte íntegra del arsenal militar y estrategias de defensa, reduciendo la presencia de tropas en los conflictos armados.
- Los drones, también, sirven en la ofensiva ya que se han usado para lanzar bombas teledirigidas contra blancos militares.

MISIONES CIVILES.

- Acceso y vigilancia de grandes zonas naturales o zonas de muy difícil acceso.
    - Los investigadores del volcán de Turrialba en Costa Rica, utilizan tres UAVs para sobrevolar y controlar el peligroso cráter. Éstos envían la información vía satélite y permite crear al momento los mapas de concentración y distribución de gases volcánicos.
    - El gobierno de Sudáfrica utiliza drones para vigilar los parques y reservas naturales. El Parque Nacional Kruger cuenta con un grupo de drones dedicados exclusivamente a detectar cazadores furtivos.
- Seguridad ciudadana y vigilancia de infracciones cívicas.
    - Las más comunes son las infracciones de tráfico, para cuyo control se está utilizando drones de vigilancia.
    - Los grafitis en los vagones de metro y tren cuestan millones de euros de las arcas públicas cada año, por ello, en Alemania están utilizando drones para vigilar y localizar a los autores de estas infracciones.
- Comercial.
    - Entregas a domicilio. Hay empresas que están desarrollando drones capaces de realizar repartos a domicilio. Entre las más importantes está la compañía Amazon.

Figura 2-3: Dron de uso comercial

- Lúdico. El carácter recreativo puede ser uno de los usos más extendidos en el futuro de los aviones no tripulados. Este tipo de drones suele venir con cámaras de vídeo para que se pueda grabar los recorridos surcando los cielos y haciendo acrobacias increíbles. Manejarlos es tremendamente sencillo ya que basta con un móvil o una tableta con iOS o Android que lo ponen al alcance de todos.

- Fotografía y vídeo. Montando una cámara sobre los UAVs nos permiten realizar tomas aéreas desde posiciones imposibles con otros instrumentos.
- Rescate y búsqueda de personas en labores de salvamento marítimo y apoyo en incendios forestales.
- Destacan especialmente en tareas de rescate y búsqueda después de una catástrofe natural como puede ser un tifón o terremoto ya que ayudan a localizar personas desaparecidas en terrenos accidentados y de difícil acceso como zonas montañosas. En esa misma línea se usan para transportar medicamentos y primeros auxilios a áreas que permanecen aisladas.

2.3 TIPOS DE DRONES

La primera clasificación se hace en función de las alas.

- Multirrotores**:** Son los más utilizados porque, proporcionan una gran versatilidad y estabilidad.

  A su vez este tipo de dron se puede clasificar según el número de motores en tricópteros, cuadricópteros, hexacópteros y octocópteros. Estos son los multirrotores más comunes aunque es posible encontrar alguno con más motores.

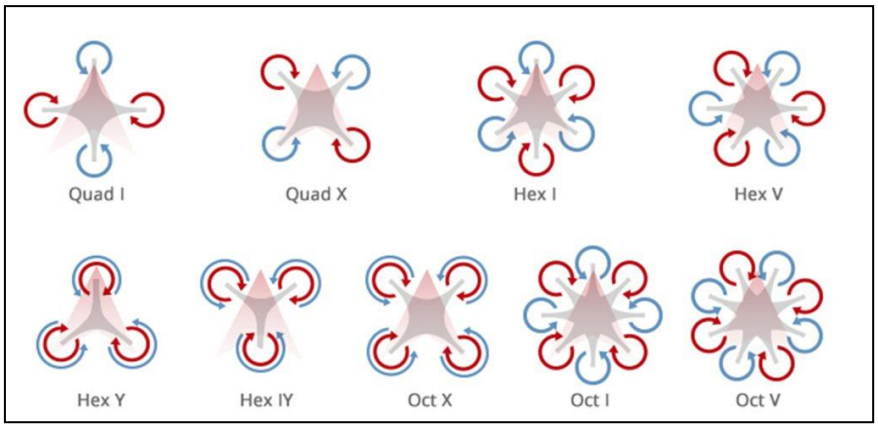


Figura 2-4: Clasificación multirrotores

Las tres figuras de la parte inferior izquierda de la figura 2-4 son multirrotores coaxiales porque tienen en cada brazo dos motores. La ventaja es que se reduce el peso al tener menos brazos.

Para elegir el dron que más se ajuste a nuestras necesidades se debe saber que cuanto mayor sea el número de brazos mayor será la estabilidad y cuantos más motores mayor la propulsión.

- Ala fija: Mantienen una estructura simple, con buena aerodinámica. Suelen ser utilizados para cubrir grandes extensiones de terreno ya que permite tiempos de vuelo largos a velocidades constantes.

  Su principal desventaja es el sistema de aterrizaje y despegue, ya que, al no poder hacerlo verticalmente, se necesita una gran extensión de terreno plano y sin obstáculos.

Figura 2-5: Dron de ala fija

- Helicópteros: Se trata de aeronaves propulsadas por uno o varios rotores horizontales, los cuales constan de dos o más palas. Son vehículos con alas giratorias que crean sustentación al girar las palas sobre un eje vertical.

  La ventaja de este tipo de dron respecto el anterior es que el rotor es capaz de proporcionar sustentación sin que el vehículo se encuentre desplazándose. Gracias a la capacidad tanto de despegar y aterrizar verticalmente como de mantenerse en vuelo estacionario durante largos periodos de tiempo, se utiliza en servicios de búsqueda y rescate, vigilancia, videografía, transporte, usos militares, etc.

Figura 2-6: Helicóptero

Dependiendo del método de control los drones se clasifican en:

- Autónomo: El dron se guía mediante sus sistemas y sensores integrados, por lo que no necesita una persona que lo controle.
- Supervisado: El dron es capaz de realizar algunas tareas autónomamente pero es imprescindible un operador que se encargue de pilotarlo.
- Pre-programado: El dron es guiado por un plan de vuelo que se ha diseñado anteriormente. El problema de este método es que no se pueden hacer cambios instantáneos en el plan.
- Controlado remotamente: Mediante una consola o mando, un operador controla el dron directamente. Actualmente es el sistema más utilizado.

2.4 VENTAJAS E INCONVENIENTES FRENTE A OTRAS AERONAVES

Las principales ventajas que presentan los drones son:

- Costes: Los drones tienen presupuestos inferiores. El precio por hora de vuelo es más bajo en aviones no tripulados que los que tienen piloto.
- Tiempo: Se ahorra tiempo en las tareas previas al vuelo como verificaciones, listas de chequeo, preparación de la cabina, etc.

- Reducción del riesgo humano: Al no estar tripulado el riesgo de accidentes que involucren a humanos es mucho menor. Por ello los drones son una de las mejores opciones en entornos peligrosos.
- Contaminación: En estos momentos la mayoría de los drones funcionan con una batería eléctrica, pudiendo realizar muchas tareas para las que se necesitan aparatos movidos con motores de combustión. Además, la contaminación acústica producida por un dron es menor que en otros sistemas.
- Precisión: En condiciones estables se consiguen vuelos muy precisos y permite tener un vuelo muy controlado.

Las principales desventajas son:

- Autonomía: Las baterías eléctricas son uno de los grandes problemas debido a su escasa capacidad. Cuanto más pequeño sea el dron, de menos autonomía dispondrá. Es cierto que se están desarrollando aviones no tripulados para usos militares que disponen de energía para varias horas de vuelo.
- Normativas: En algunos países prácticamente no existen leyes sobre este tipo de vehículo mientras que en otros se encuentran muchas restricciones para su uso.
- Ética: Es posible que con la ayuda de la inteligencia artificial pueda determinar por sí mismo los objetivos. Es insensible a las consecuencias de sus acciones.

2.5 NORMATIVA Y LEYES PARA DRONES

Europa es el continente con más restricciones para la navegación de drones, tanto en actividades recreativas como profesionales.

Para poder pilotarlo en nuestro país con fines comerciales o profesionales, además de tener al menos 18 años, es necesario el carnet de piloto o acudir a un centro en el que te acrediten que puedes navegar con él. [5]

Está prohibido superar los 120 metros de altura, volarlo en zonas urbanas o de noche, ya que se reduce la visibilidad y aumenta el peligro y cerca de aglomeraciones de personas.

Incumplir alguna de estas normas, especialmente las que ponen en riesgo a terceros, conlleva multas económicas muy altas.

3. APRENDIZAJE AUTOMÁTICO

Es la rama de la inteligencia artificial centrada en el desarrollo de sistemas que “aprenden” a realizar predicciones o tomar decisiones en base a datos de entrada y sin haber sido especialmente programados para el programa concreto “aprendiendo” de forma autónoma. [6]

Estos sistemas son útiles, entre otras cosas, en problemas cuyo modelo matemático es complejo: problemas NP-Hard, transiciones estocásticas, problemas con un espacio de estados demasiado grande. Etc

El objetivo actual en la mayor parte de las aplicaciones de la inteligencia artificial, buscan conseguir un desempeño por parte de un sistema de una tarea compleja con un nivel de exactitud cercano o superior al humano, que resulte suficientemente bueno para ser útil en diversos casos de uso.

Para ello estos sistemas generalizan los patrones de comportamiento que aprende de los datos de entrada. Entendiendo generalizar como ser capaz de tener un rendimiento adecuado ante nuevos datos nunca vistos antes.

3.1 APLICACIONES

El uso del machine learning ha experimentado un auge estos últimos años.

Esto se debe, a la mejora del hardware para el procesamiento, con GPUs más potentes y baratas, gracias a la expansión de los videojuegos. También a la posibilidad de tener mayor acceso a gran cantidad de datos para los entrenamientos, gracias a Internet. Y unido a las importantes mejoras en los algoritmos y el software, que permiten obtener resultados de forma más sencilla y rápida.

El machine learning ha conseguido mejoras en el estado del arte de muchas aplicaciones, como la clasificación de imágenes para la visión artificial, problemas de regresión y clustering para agrupar datos para sistemas de recomendación.

Casos de éxito pueden ser el coche autónomo de Google, asistentes personales como Siri o Amazon Echo, y batir la inteligencia humana en juegos complejos como el Go. [7]

Otro campo en el que están causando gran repercusión es el de las telecomunicaciones, especialmente en el procesado de señales. La capacidad de las redes para clasificar objetos permite utilizarla en radares. También en sistemas de control complejos para modelizar, realizar predicciones de su comportamiento y eliminar el ruido mediante la aplicación de distintos filtros y sistemas adaptativos.

En economía se están usando para la concesión de créditos, ya que a partir de unos marcadores económicos es capaz de decidir la viabilidad del préstamo. También es común utilizarlas en previsión de stocks, ya que la falta o exceso de suministros es uno de los principales problemas de muchas empresas.

DEEPMIND: EJEMPLO DE UNA COMPAÑÍA DE DEEP LEARNING

La empresa que más se ha enfocado en el desarrollo de inteligencia artificial general es DeepMind.

Mientras que la mayoría de soluciones de machine learning se empezaron a enfocar a tareas concretas, con las observaciones y estados preprocesados a mano para un funcionamiento más sencillo, DeepMind ha conseguido desarrollar agentes que interactúan directamente con imágenes, sin pre-procesados manuales. [8]

Avances en el estado del arte han quedado publicados en papers, donde se consiguen agentes jugando a videojuegos Atari con mejores resultados que cualquier humano

3.2 MODELOS O PARADIGMAS DEL APRENDIZAJE AUTOMÁTICO

3.2.1 Supervisado

Donde la hipótesis es calculada a partir de unos datos de entrada etiquetados. Al sistema se le introducen durante el entrenamiento las entradas y salidas deseadas, siendo capaz de generalizar la hipótesis, al final del entrenamiento, a nuevos casos.

Este modelo suele ser más fácil de entender y desarrollar.

El problema lo encontramos en que los datos tienen que estar previamente etiquetados, y requieren gran cantidad de datos para ser entrenados.

Para facilitar esta tarea, varios organismos ofrecen distintos datasets ya etiquetados para que investigadores y desarrolladores entrenen y validen sus modelos, tales como MNIST, ImageNet o FERET. [9]

3.2.2 No supervisado

Donde el modelo descubre patrones en los datos. Al no estar etiquetados, no hay un conocimiento previo de los valores de salida, y no se puede calcular la precisión del modelo, que representa un modelo de densidad para el conjunto de datos.

3.2.3 Aprendizaje por refuerzo

Este es el tipo de aprendizaje con el que se ha trabajo durante los siete meses que ha durado este trabajo fin de grado.

En este tipo de aprendizaje, un agente interactúa con un entorno, aprendiendo por sí mismo mediante prueba y error. El agente aprenderá una estrategia que le permita obtener la mayor recompensa a la larga, buscando maximizar la recompensa esperada r, al pasar de un estado s a otro s' con una acción a. [10]

Los problemas se suelen representar como un proceso de decisión de Markov. [11]

Este proceso suele tener reglas de transición de estados estocásticas, no habiendo garantías de que al realizar la misma acción se acabe en el mismo estado.

Una ventaja del aprendizaje por refuerzo es que no se necesita conocer el modelo del entorno.

Otras ventajas de este sistema son que es adaptativo a lo largo del tiempo, y que es más robusto al ruido que presentan las medidas por la estocasticidad y por la falta de una observación completa de los estados.

Es importante señalar que ni siquiera es necesario conocer todos los elementos de la cadena de Markov, para llegar a una solución. [12]

Exploración

Durante la exploración, el agente atraviesa los distintos estados, de forma que aprende qué acciones ofrecen mayor recompensa en cada uno de ellos.

Debido al ruido, provocado por la estocasticidad del medio, debe realizar múltiples veces cada acción en cada estado, para que el valor sea fiable.

Con cadenas de Markov muy grandes, recorrer todos los estados para llegar a conocer el camino óptimo es imposible.

Aparece entonces un dilema: realizar la exploración de nuevas transiciones para buscar caminos con mejores resultados, o seguir explotando los caminos conocidos, y obtener su valor esperado real.

Para ello se desarrollan distintas estrategias de exploración que busquen un equilibrio, siendo ϵ-greedy una de las más comunes.

Con ϵ-greedy, la acción a realizar depende no solo de la recompensa esperada, sino también de un número ϵ [13]. Su valor varía a lo largo del entrenamiento, partiendo normalmente de $\epsilon = 1$, en el que las acciones se toman de manera aleatoria, y se va

reduciendo según una función dada. A medida que se reduce este valor, el agente se vuelve cada vez más avaricioso, realizando más veces las acciones con mayor beneficio.

4. Q-LEARNING

Este es uno de los algoritmos que hemos utilizado y probado durante el proyecto. Permite obtener los valores de la recompensa esperada en cada estado si nos comportamos de manera óptima, y de esta forma encontrar la ruta que maximiza el valor de la recompensa total recibida a la larga.

Siendo T el momento en el que el episodio termina, el agente tiene en cuenta no solo la recompensa del estado actual, sino también la de los siguientes pasos, descontada por el factor de descuento $0 < \gamma \leq 1$, que marca la importancia de las recompensas futuras. [16]

Cuando su valor es 0, el agente no percibe más recompensa que la actual.

Si el valor es uno, y el entorno no es episódico, la solución diverge, pues el sumatorio no daría un número finito. [17]

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

La función que calcula el valor estado-acción óptimos $Q^*(s,a)$ queda definida como la mayor recompensa obtenida siguiendo la política óptima.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[R_t \,|\, s_t = s,\ a_t = a,\ \pi \right]$$

Siguiendo la ecuación de Bellman, si el valor óptimo de $Q^*(s',a')$ para el siguiente estado *s'* es conocido para todas las posibles acciones *a'*, la política óptima se extraerá seleccionando la acción que proporcione un mayor valor de $Q^*(s',a')$. La ecuación de Bellman nos ayuda a estimar la recompensa estimada futura atenuada por el factor de descuento, $r + \gamma Q^*(s',a')$. [19]

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

El valor Q del estado no se calcula de forma exacta con la primera medida, pues el medio es estocástico y presenta ruido. Se realizan, cada vez que se atraviesa dicho estado, nuevas iteraciones, teniendo en cuenta el valor anterior:

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

Está demostrado que, para una MDP finita, es posible atravesar todos los estados y calcular los valores Q que convergen a su solución final

$$Q_i \to Q^* \ as \ i \to \infty$$

El learning rate es el factor que decide cuánto se debe aproximar el valor Q al valor calculado. Si el ambiente fuera determinista el valor ideal sería 1 ya que la solución final es calculada al instante. Pero con ambientes estocásticos este valor debe disminuirse para evitar inestabilidades, de forma que los valores se van actualizando parcialmente a cada paso.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{old\ value} + \underbrace{\alpha}_{learning\ rate} \cdot \left(\overbrace{\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discount\ factor} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{estimate\ of\ optimal\ future\ value}}^{learned\ value} - \underbrace{Q(s_t,a_t)}_{old\ value} \right)$$

Al ser un algoritmo iterativo se presenta el problema del valor inicial de Q_0. [20] Si se asignan valores elevados, se producen unas condiciones iniciales optimistas que incentivan la exploración, ya que para cualquier acción el valor es menor que para la otra alternativa. Una solución posible es usar el primer valor obtenido de r como Q_0.

4.1 EXPERIMENTO 1: RANDOM WALK

Hemos desarrolado el entorno Random Walk para comprobar algoritmos desarrollados con qlearning.

En él, el agente se encuentra en un pasillo de 10 cuadrículas de largo por 1 de alto. En un extremo, la posición 0, recibe un castigo, mientras que al otro extremo, en la posición 9, recibe un premio.

El agente empieza en el medio, la posición 5, sin ningún conocimiento previo del entorno. Las acciones permitidas son dos, avanzar hacia la derecha o hacia la izquierda. La política que elige estas acciones es aleatoria. [17]

Al ser un entorno tan pequeño, no se necesita ninguna estrategia compleja para recorrer todos los estados.

El agente no parará de realizar acciones aleatorias hasta llegar a uno cualquiera de los dos extremos, momento en el que recibe la recompensa o el castigo, finalizando el episodio, y volviendo a la posición de partida.

Los algoritmos desarrollados constan de 3 hiperparámetros: learning rate, la tasa de descuento y el número de episodios.

Figura 4-1: Entorno RandomWalk

Como se puede apreciar, en el primer instante los valores son nulos, y el agente desconoce aún el valor de los extremos.

```
Valores iniciales
[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
```

Tras 4 episodios, el agente ha llegado a ambos extremos, y el valor de la recompensa esperada se empieza a expandir a los estados cercanos. El learning rate utilizado es de 0.1 y la tasa de descuento de 0.9.

```
Valores con 4 episodios
[[0, -0.1], [0.0, 0], [0.0, 0], [0.0, 0], [0.0, 0.0],
 [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.02], [0.19, 0.1]]
```

Después de 100 episodios, el agente ha llegado prácticamente al valor de la recompensa final. Con los mismos valores de learning rate y tasa de descuento.

```
Valores con 100 episodios
[[-0.86, -0.89], [-0.64, 0.16], [0.1, 0.23], [0.16, 0.3], [0.23, 0.38],
 [0.3, 0.47], [0.38, 0.57], [0.47, 0.68], [0.57, 0.81], [0.94, 0.95]]
```

El learning rate y el nuevo valor calculado indican cuánto debe ser modificado el valor.Q. Subiendo el learning rate a 0.5, los cambios son mayores, consiguiendo que el agente aprenda 5 veces más rápido, con un resultado parecido en 20 pasos.

```
Valores con 20 episodios
[[-0.97, -0.98], [-0.83, 0.09], [0.08, 0.33], [0.29, 0.38], [0.32, 0.49],
 [0.39, 0.58], [0.5, 0.65], [0.58, 0.76], [0.57, 0.88], [0.5, 0.99]]
```

Si reducimos a 0,5 el factor de descuento, la cantidad de recompensa repartida al estado anterior se reduce. Con esto es fácil de entender porque en muchos experimentos son usados valores altos en el factor de descuento, cercanos a 1. Reducir este factor implica que el agente compartirá menos información con los siguientes pasos.

5. REDES NEURONALES

Durante el proyecto hemos entrenado redes neuronales de manera continua.

Los algoritmos de optimización de 'gradient descent' para redes neuronales se están haciendo cada vez más populares [21]. Los optimizadores utilizados por los experimentos principalmente se encuentran en Keras, aunque en alguna ocasión ha sido necesario usar Tensorflow para ajustar en mayor grado los parámetros propios de dicho optimizador.

Durante el proyecto se han probado los siguientes optimizadores:

- SGD, denominado optimizador estocástico de la pendiente del gradiente. Es el más general.
- RMSprop, es la mejor opción para redes de tipo recurrente. En Keras funciona bien dejando sus parámetros por defecto, excepto el learning rate.
- Adagrad, es una adaptación muy usada del SGD.
- Adadelta, tiene una tasa computacional mínima y es ideal para tareas de clasificación.
- Adam, se utiliza en problemas que requieren una gran cantidad de datos, ya que es un método simple.
- Nadam, es similar al Adam sólo que se le añade un parámetro de carácter booleano denominado impulso de Nesterov.

5.1 DYING RELUS

Las Relus son las funciones de activación que hemos utilizado en la mayoría de experimentos.

Analizando el Pong, se observó que en los entrenamientos se generaban gradientes muy elevados, por lo que se estudió las dying Relus, pensando que era una de las posibles causas de este problema.

Un gran gradiente que fluye a través de una neurona Relu podría hacer que los pesos se saturen, de manera que la neurona podría no volverse a activar nunca.

Si todas las entradas ponen la Relu en esta zona:

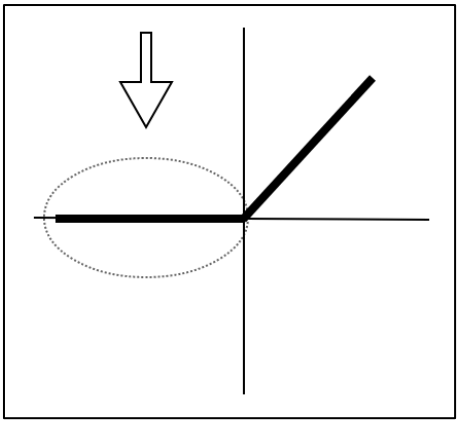


Figura 5-1: Dying Relu

No hay esperanza de que los pesos cambien, y por tanto la neurona muere.

Cuanto mayor sea el “learning rate”, menos posibilidades hay de que las neuronas se reactiven una vez muertas, ya que una actualización brusca puede mover el valor del bias de tal manera que vuelvan a caer en la zona muerta.

5.2 EXPERIMENTO 2: APROXIMADOR DE FUNCIONES NEURONAL

Realizamos este experimento para demostrar la habilidad de las redes neuronales para aproximar funciones. Para ello, se genera una nube de 500 puntos con una función seno, y se le añadió ruido para dificultar la aproximación, y así que la red tenga que generalizar, y no memorizar (overfitting).

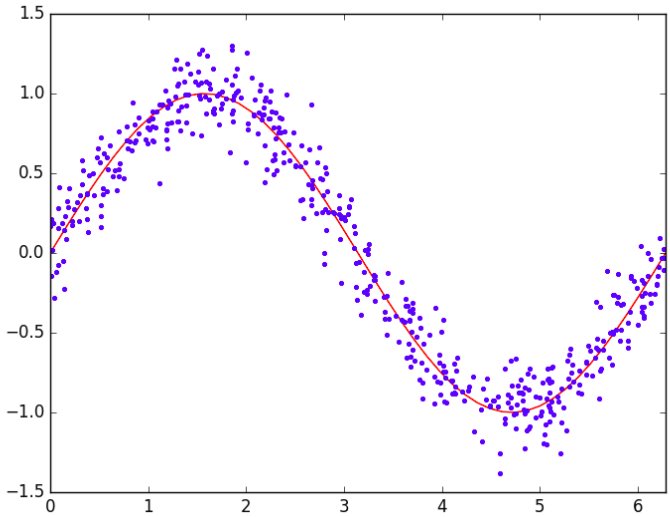


Figura 5-2: Nube de puntos

Para entrenar la red se utilizan 400 datos de la nube de puntos, y los restantes 100 se usan para calcular el error y validar el entrenamiento. Con cada época, la red entrena con los puntos y calcula el error con los datos de validación.

Esta red utiliza un optimizador de descenso del gradiente, y calcula el error cuadrático medio.

Con una capa de 3 neuronas, y 150 épocas, los datos de salida calculados por la red aparecen en la figura con línea amarilla. El error cuadrático medio total conseguido es, 0.0848551048539. Si subimos el número de neuronas a 16, el error sube hasta 0.10229545171.

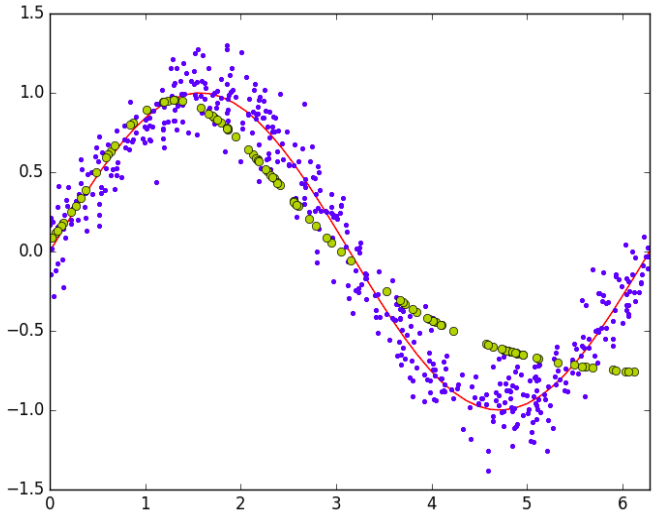


Figura 5-3: Aproximación de la función seno con 3 neuronas, 150 épocas

Con una capa de 3 neuronas, y 1000 épocas, los datos de salida calculados por la red aparecen en la figura con línea amarilla. El error cuadrático medio total conseguido es, error: 0.0556941323677.

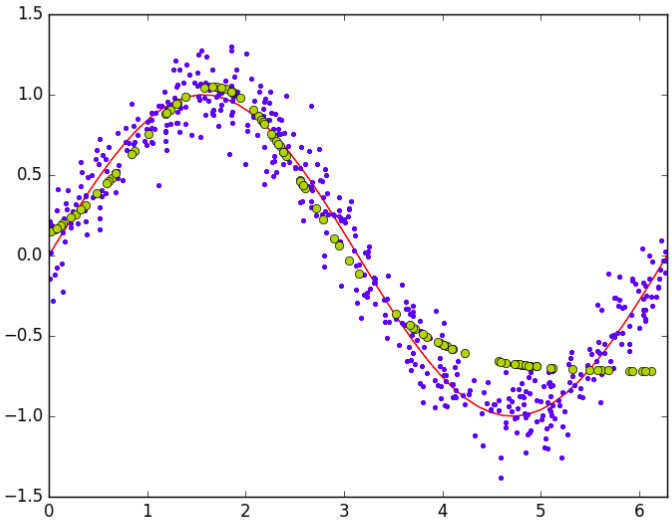


Figura 5-4: Aproximación de la función seno con 3 neuronas1000 épocas

Con una capa de 3 neuronas, y 5000 épocas, los datos de salida calculados por la red aparecen en la figura con línea amarilla. El error cuadrático medio total conseguido es

error: 0.0245432047916.

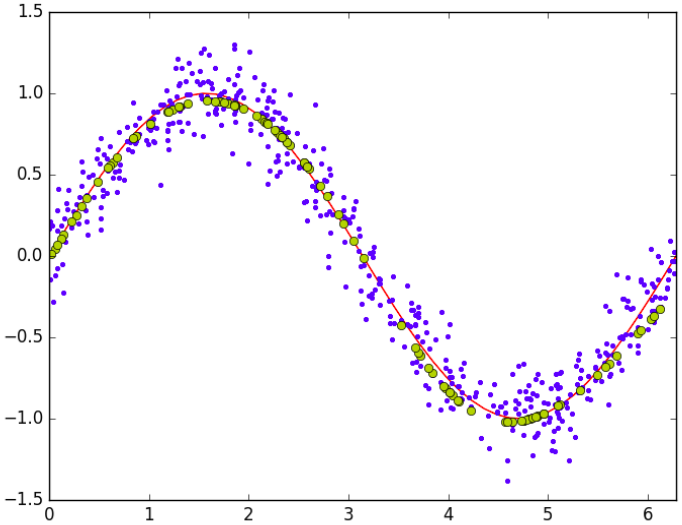


Figura 5-5: : Aproximación de la función seno con 3 neuronas 5000 épocas

Con dos capas de 16 neuronas, la red consigue un mejor resultado en mucho menos tiempo, 1500 épocas. El error cuadrático medio total conseguido es 0.0228717872759.

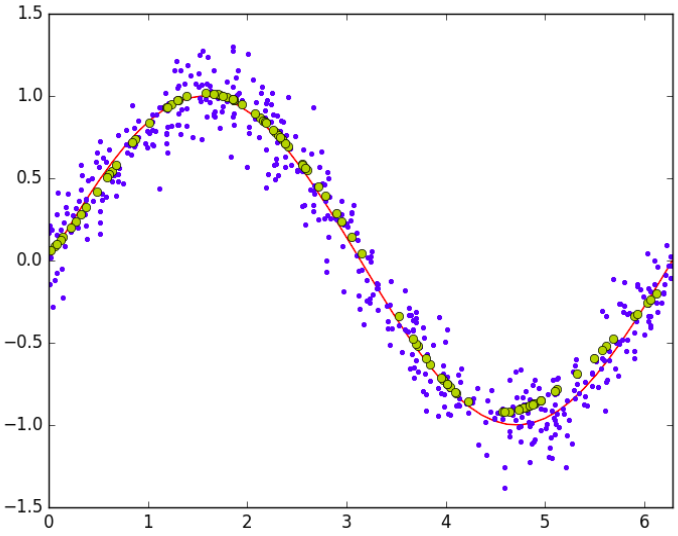


Figura 5-6: Aproximación de la función seno con 2x16 neuronas y 1500 épocas

Para ilustrartrar que las redes neuronales son un aproximador universal de funciones, se le ha aplicado una función distinta, coseno, a la misma red de antes y se ha obtenido un error: 0.0244400188308.

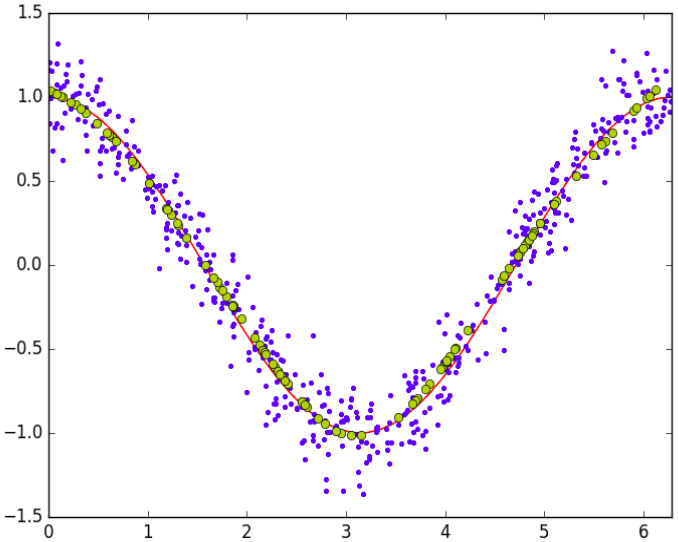


Figura 5-7: Aproximación de la función coseno

6. DQN

En la práctica se ha llegado a la conclusión de que la implementación del q-learning no es útil por tener que representar el valor de cada estado por separado, sin ninguna clase de generalización, algo inviable en entornos grandes. Por lo tanto se optó por estudiar y comprobar el algoritmo DQN.

Se puede obtener una aproximación de los valores utilizando un aproximador de funciones como una red neuronal, que prediga los valores de Q, de forma que:

$$Q(s, a; \theta) \approx Q^*(s, a)$$
$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s, a; \theta_i)\right)^2\right]$$
$$y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1}) \mid s, a]$$

Se entrena con una función de error $L_i(\theta_i)$, cuyo coste es el cuadrado de la diferencia entre los valores Q_s y $Q_{s\prime}$. [24]

Si diferenciamos el error visto por la red en función de sus pesos, obtenemos el gradiente para ajustar sus valores:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\varepsilon}\left[\left(r + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

Cuyo valor se suele calcular mediante optimizadores como el descenso del gradiente o RMSprop.

Al depender los nuevos ajustes de los pesos del valor anterior de la red, se crea un lazo cerrado que suele presentar un comportamiento inestable, al juntar la red neuronal con los algoritmos de Q-learning.

Si entrenamos la red con los valores de estado que ve el agente en ese momento (entrenamiento online), puede producirse una fuerte correlación en los datos, que lleve a la inestabilidad.

Es necesario romper esa correlación.

Utilizando una replay memory, una memoria donde se van guardando las observaciones de los estados, de la que se sacan transiciones de forma aleatoria se consigue mayor estabilidad y un entrenamiento más óptimo, ya que los datos no son descartados después de cada uso. Se sacan en grupos, llamados batches. Cuanto mayor sea el tamaño del batch, más se agilizará el entrenamiento.

Además la replay memory optimiza el uso de los datos durante el entrenamiento.

Como el tamaño de la memoria es limitado, su comportamiento es FIFO, borrando las primeras muestras según entran otras nuevas.

6.1 EXPERIMENTO 3: LABERINTO

Hemos desarrollado otro entorno para probar el funcionamiento del DQN.

El laberinto es una matriz de 3 por 4 con un premio de +1 en la posición (0, 3), y un castigo de -1 en la posición (1, 3). Al caer en cualquiera de los dos estados termina el episodio, volviendo el agente a la posición (2, 0).

Un obstáculo en la posición (1, 1) impide al agente atravesar o posicionarse en dicho estado. Si se elige una acción que choque con el obstáculo el agente no avanza.

Las acciones permitidas son subir, bajar, avanzar a la derecha o a la izquierda.

Las acciones a lo largo del episodio se eligen de manera aleatoria, ya que el entorno no es muy grande.

Al igual que en el random walk, el agente irá propagando la recompensa y castigo de los estados finales, pero en este caso, este valor será aproximado por una red de 4 capas de 128 neuronas con activación Relu, seguidas de una capa lineal de 4 acciones, igual al número de salidas.

La función de pérdida es el error cuadrático medio, mse, y el optimizador utilizado es el RMSprop, con un learning rate de 0.005, y un factor de descuento de 0.9.

De esta forma el agente da importancia a la recompensa futura a la hora de decidir qué acción ejecutar.

La replay memory consta de 1000 muestras, que salen en forma de batches, es decir, en agrupaciones de 64. Por ello, los primeros 1000 pasos se invierten en llenar la replay memory, sin entrenar a la red. Ésta se entrena durante los siguientes 300 pasos, suficientes para converger a la solución final, como se aprecia en la matriz.

TRAINER_GAMMA = 0.90

LEARNING_STEPS = 1300

MAX_SIZE = 1000

lr = 0.005

```
[[ 0.06256774  0.12598079  0.20843653  1.07937133]
 [ 0.00764117  0.02882275 -0.21701086 -0.83428693]
 [ 0.04434161 -0.04628441 -0.21481034 -0.21849652]]
```

En las gráficas posteriores se observa que en el estado 3 está la zona con recompensa positiva de 1 mientras que en el estado 7 se encuentra el castigo de -1. Esto complica el aprendizaje del agente, al propagarse valores positivos y negativos en una zona pequeña, y la posibilidad de avanzar hacia el premio y que una acción aleatoria lo arroje al castigo. Tras los suficientes episodios, concluye que el camino más seguro es la parte superior del laberinto, camino más alejado del hoyo.

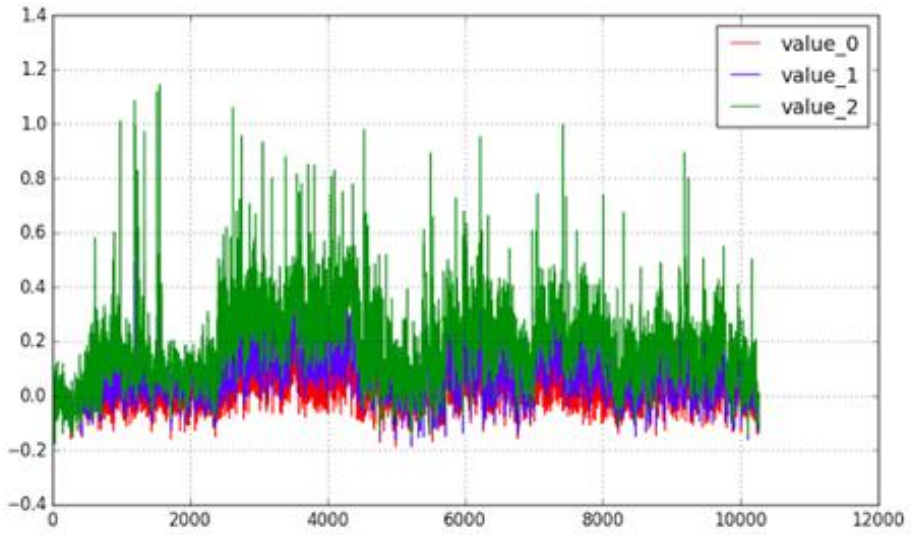


Figura 6-1: Representación de los valores 0, 1 y 2

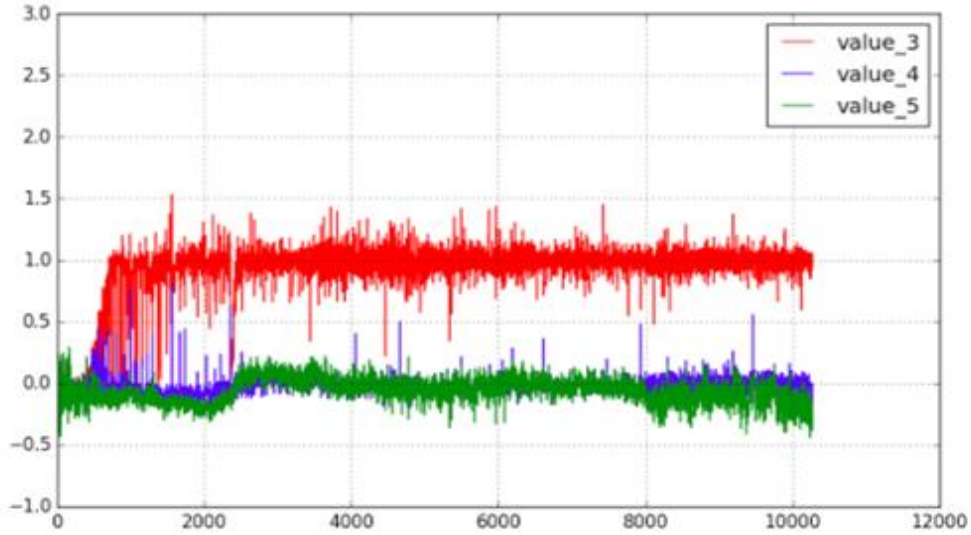


Figura 6-2: Representación de los valores 3, 4 y 5

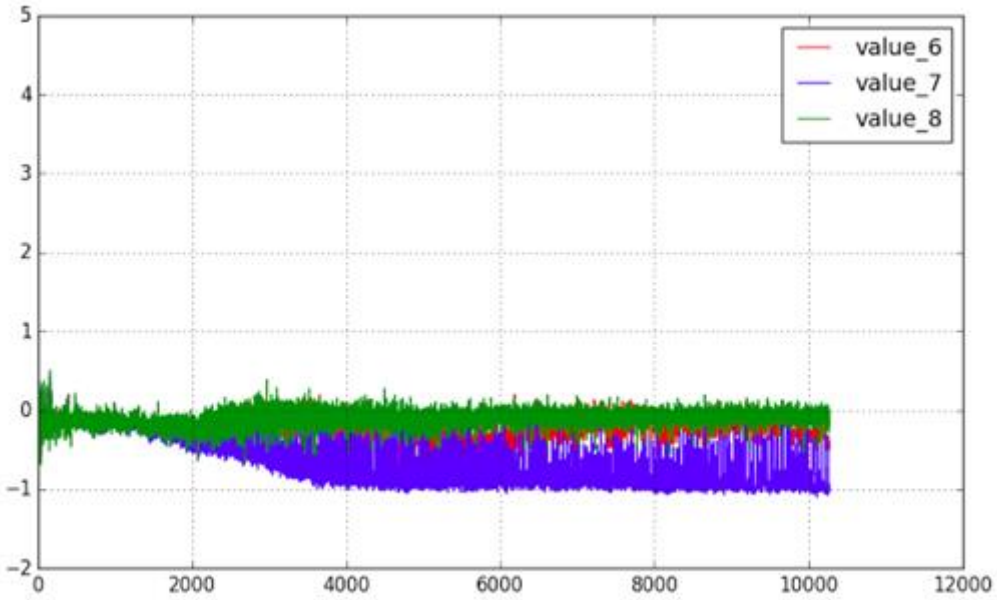


Figura 6-3: Representación de los valores 6, 7 y 8

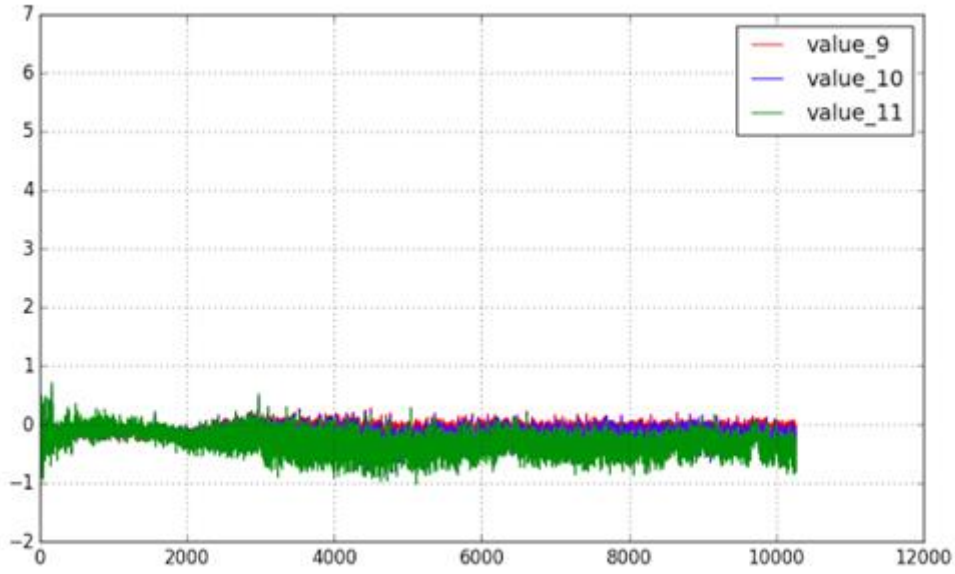


Figura 6-4: Representación de los valores 9, 10 y 11

Se realiza una nueva prueba con DQN, reduciendo el épsilon y los pasos de aprendizaje.

El resultado obtenido es peor que el anterior, ya que ha explorado menos, y al ser más avaricioso, centrándose en los episodios con recompensas positivas, propaga menos la recompensa por el resto de los estados, lo cual puede producir que el agente se quede viciado en un mínimo local. En este caso, el agente acaba desaprendiendo. El obstáculo situado en el (1, 1), que impide que el agente se pueda situar en ese estado, en todos los experimentos va a tener un valor de estado cercano a cero.

EPSILON = 0.4
REPLAY_MEMORY_SIZE = 1000
REPLAY_MEMORY_BATCH_SIZE = 64
TRAINER_GAMMA = 0.90
LEARNING_STEPS = 1100
MAX_SIZE = 1000
lr = 0.005

```
[[-0.37273794 -0.37273794 -0.37273794 -0.37273794]
 [-0.31122229 -0.40928376 -0.55972612 -1.0260365 ]
 [-0.26583195 -0.27291676 -0.33595583 -0.40072271]]
```

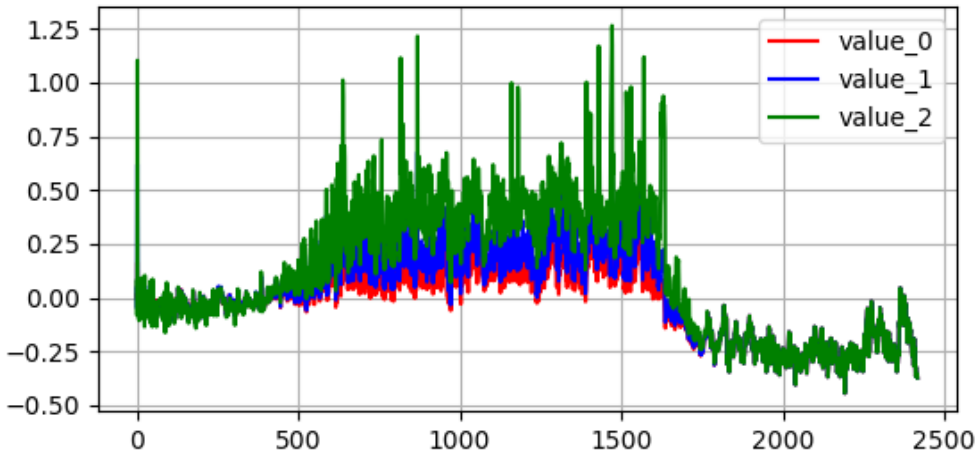


Figura 6-5: Valor estado 0,1,2 (1)

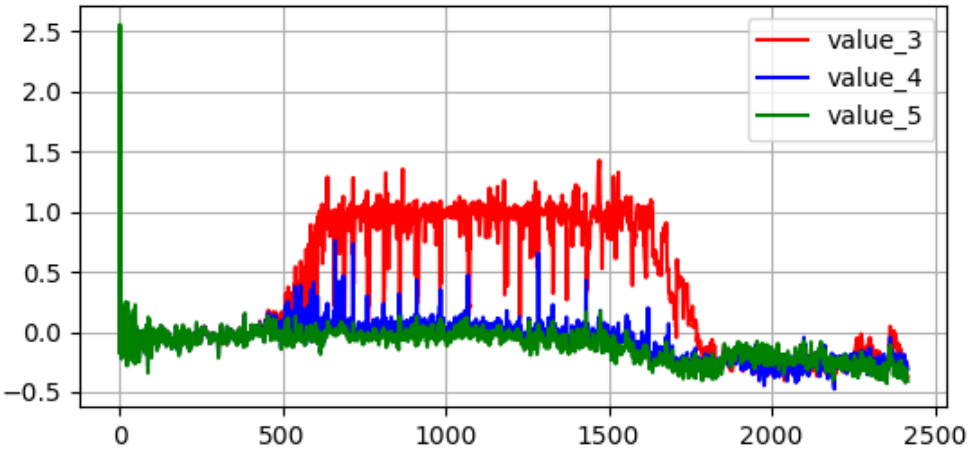


Figura 6-6: Valor estado 3, 4, 5 (1)

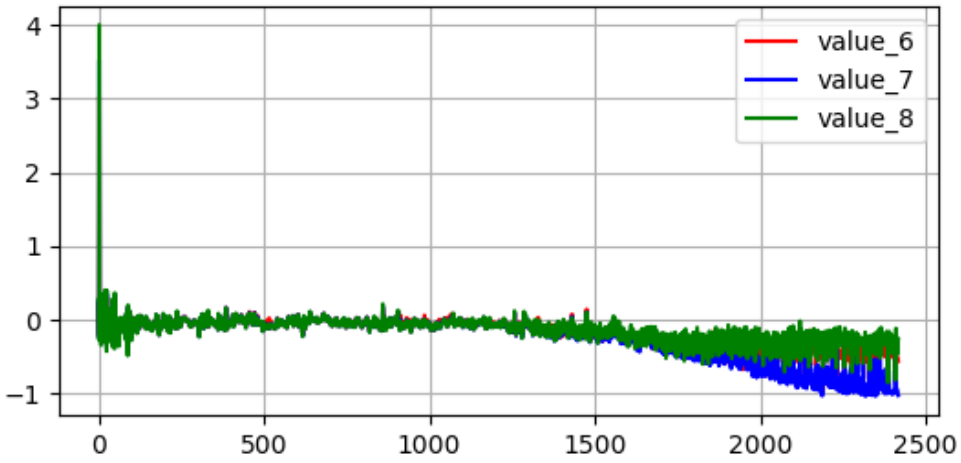


Figura 6-7: Valor estado 6, 7, 8 (1)

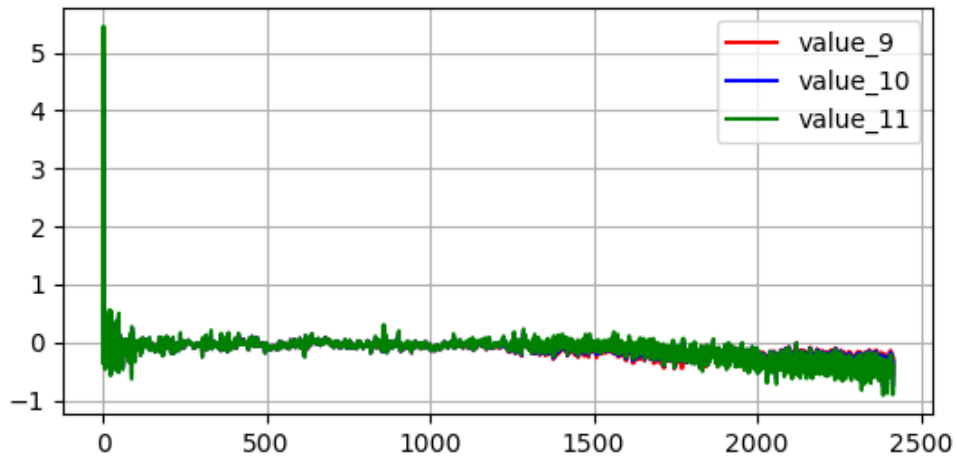


Figura 6-8: Valor estado 9, 10, 11 (1)

En este experimento se reduce la tasa de descuento. De esta manera no tiene tanto en cuenta el camino realizado para llegar al estado con recompensa, y el reparto del valor del estado no es tan ecuánime, sino que se limita a aquellos que están cerca del final. Como se puede comprobar en la matriz solución, los estados finales recogen prácticamente toda la recompensa sin distribuirla a los demás, quedando éstos en valores cercanos a cero. Generalmente esto dificulta el aprendizaje al comienzo de cada episodio, pues el camino hacia el premio apenas está marcado.

EPSILON = 1

REPLAY_MEMORY_SIZE = 1000

REPLAY_MEMORY_BATCH_SIZE = 64

TRAINER_GAMMA = 0.50

LEARNING_STEPS = 1300

MAX_SIZE = 1000

lr = 0.005

[[-0.00951993 -0.00752822 0.02895202 1.08753276]

[0.02409166 0.02409166 -0.10315387 -0.8654213]

[-0.02930263 -0.05762146 -0.09440972 -0.18589728]]

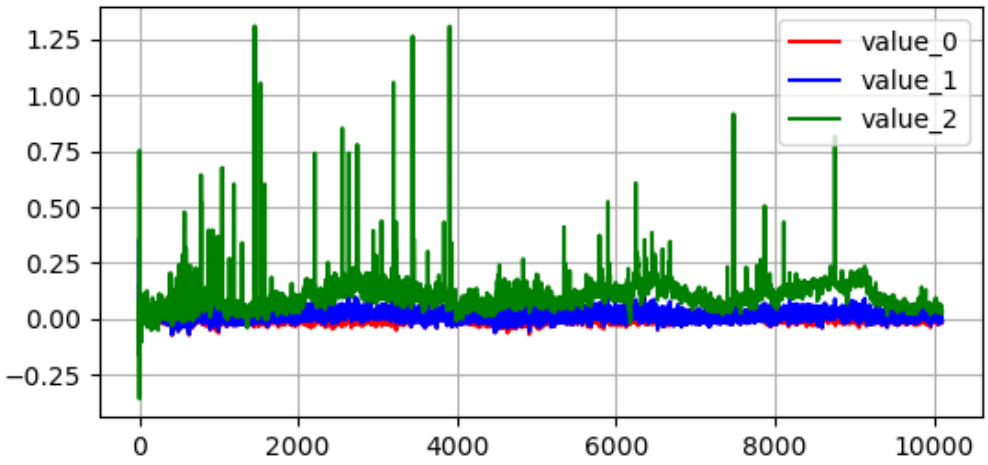


Figura 6-9: Valor estado 0, 1, 2 (2)

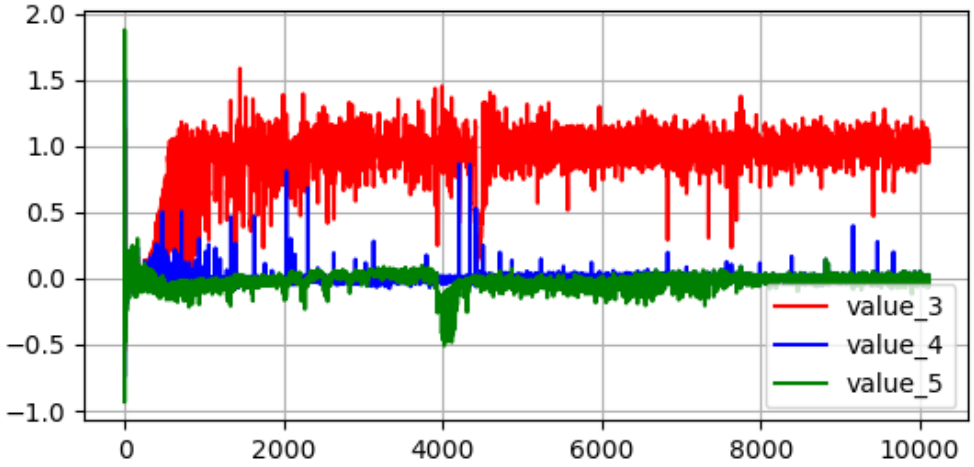


Figura 6-10: Valor estado 3, 4 ,5 (2)

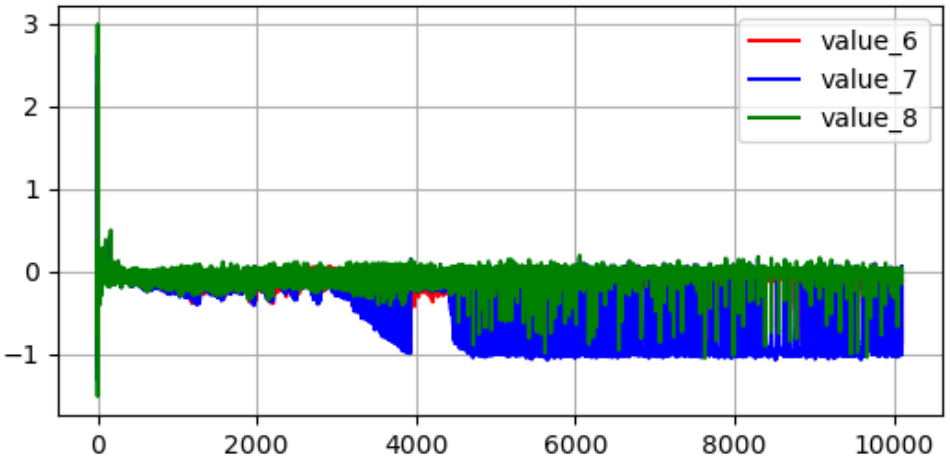


Figura 6-11: Valor estado 6 ,7 ,8 (2)

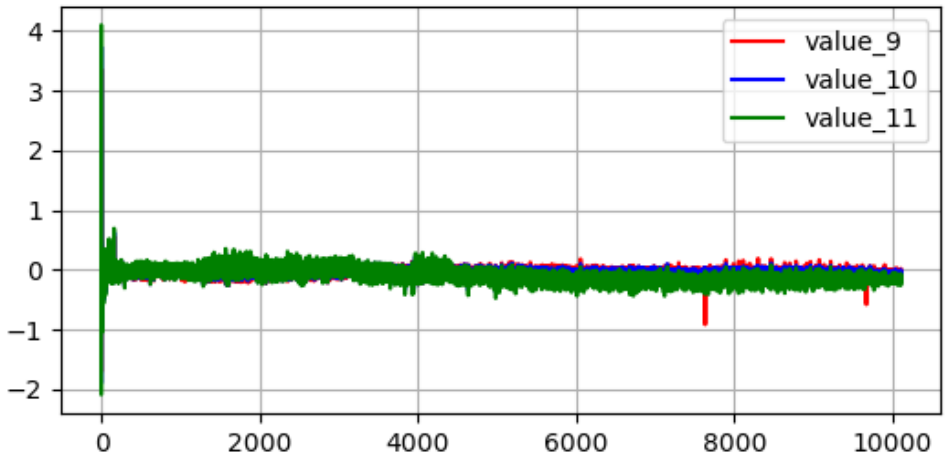


Figura 6-12: Valor estado 9, 10, 11 (2)

Se reduce el número de capas a 2 y las neuronas por capa a 32. Los valores de los demás hiperparámetros son los mismos que el primer experimento del laberinto con DQN. Los resultados presentan mucho más ruido y el tiempo de convergencia de los estados a sus valores finales es mayor.

EPSILON = 1.0
REPLAY_MEMORY_SIZE = 1000
REPLAY_MEMORY_BATCH_SIZE = 64
TRAINER_GAMMA = 0.90
LEARNING_STEPS = 1300
MAX_SIZE = 1000

```
[[ 0.06635167  0.1366484   0.23995326  0.87029088]
 [ 0.0385315  -0.08049333 -0.23734166 -0.98124397]
 [-0.01011869 -0.10009263 -0.23915301 -0.37647539]]
```

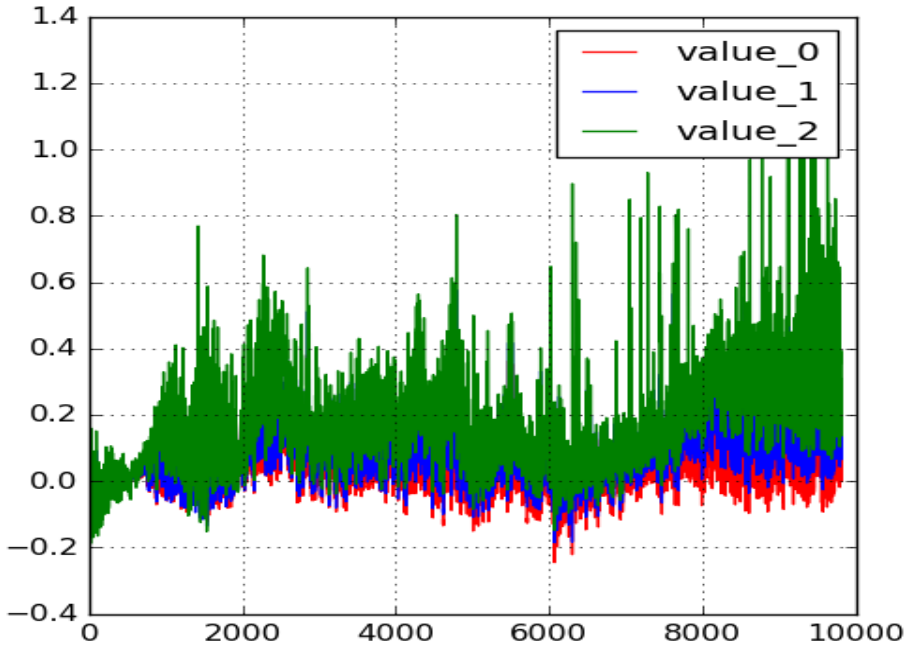


Figura 6-13: Valor estado 0, 1, 2 (3)

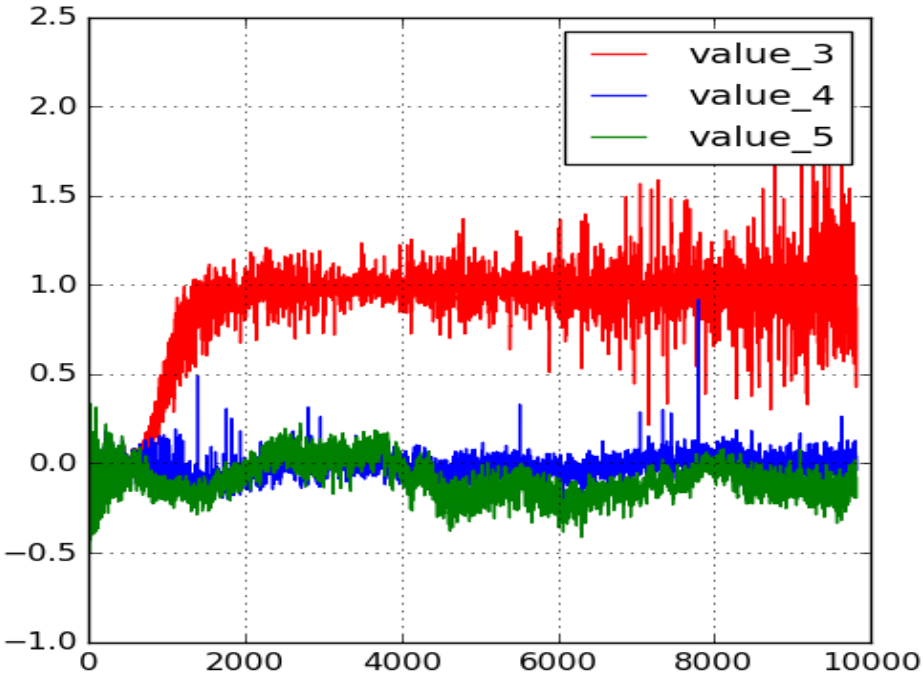


Figura 6-14: Valor estado 3, 4, 5 (3)

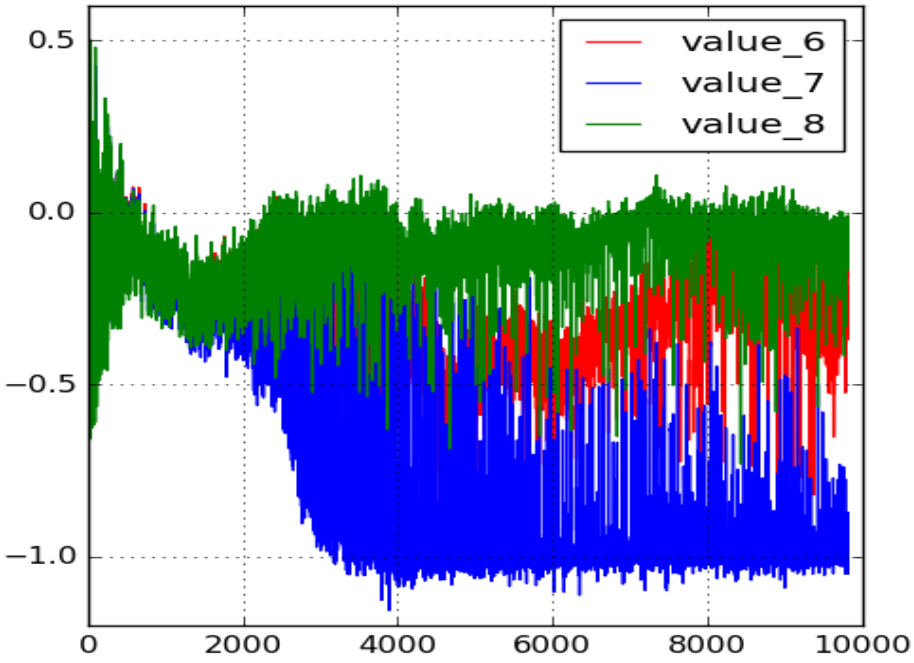


Figura 6-15: Valor estado 6, 7, 8 (3)

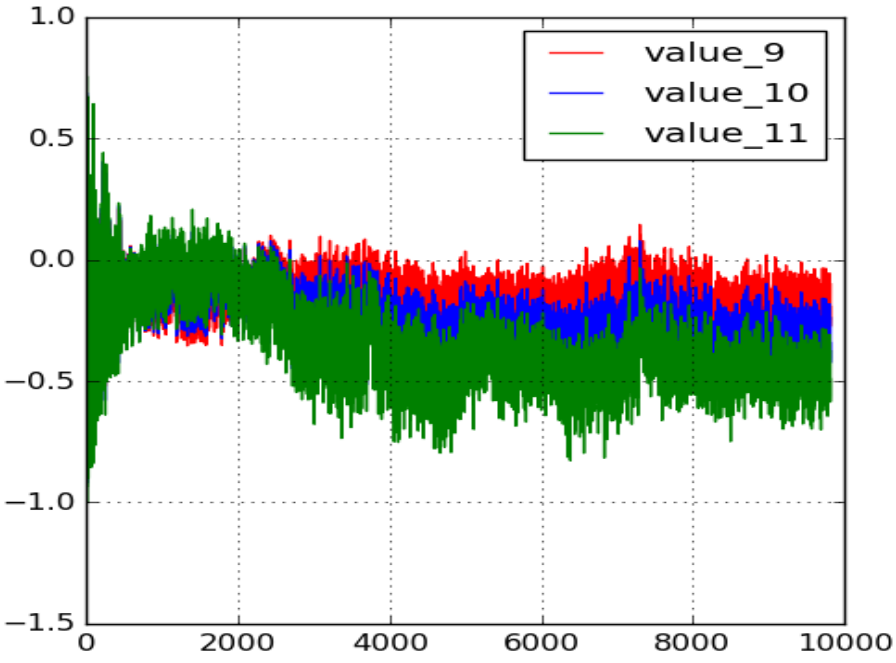


Figura 6-16: Valor estado 9, 10, 11 (3)

6.2 EXPERIMENTO 4: CARTPOLE

También hemos utilizado el entorno CartPole de OpenAI gym, que se compone de un carrito cuyo objetivo es mantener un palo, unido a su cuerpo por un eje, erguido.

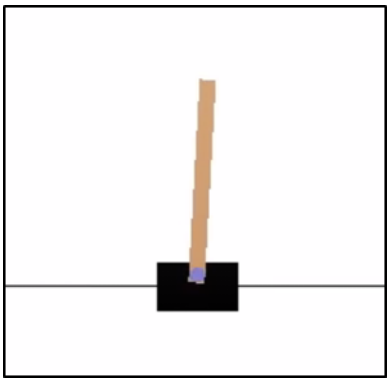


Figura 6-17: Entorno Cart Pole

El palo se balancea con el movimiento del carro, y no puede exceder un ángulo de ±15 grados de la vertical, momento en el que se reinicia el entorno.

El agente recibe un premio de +1 por cada segundo que mantenga el palo entre estos límites, siendo la puntuación máxima 200, momento en el que termina el episodio.

Los datos de entrada que definen el estado y vistos por la red son la posición y velocidad del carro, y el ángulo y velocidad en la punta del palo. Las acciones posibles para el agente son avanzar un paso a la izquierda o a la derecha.

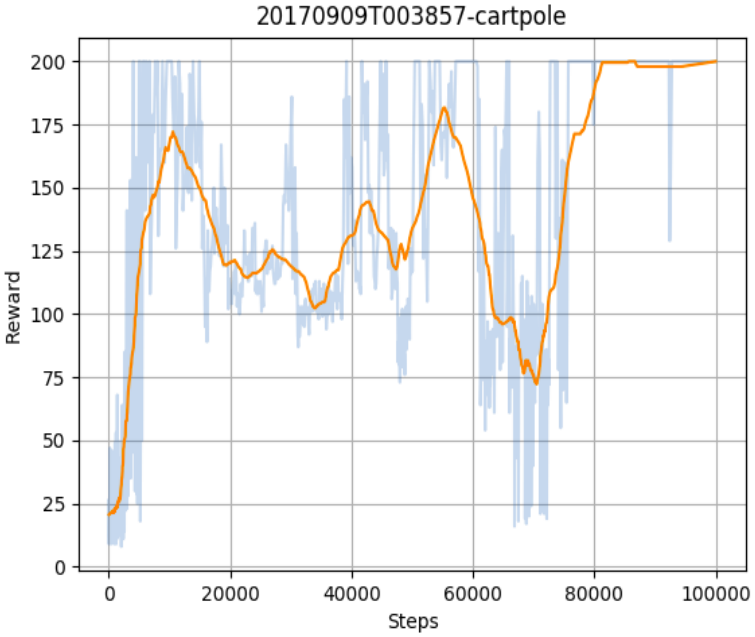


Figura 6-18: Recompensa dqn

Podemos observar la recompensa que ha ido recibiendo el agente a lo largo de los 100000 mil pasos del experimento.

La suave línea azul es la recompensa de cada paso, siendo la naranja una media a lo largo de 100 pasos para tener una visión más clara del resultado con el mínimo ruido.

Estas estrategias de representación son útiles en entornos donde la recompensa cambia mucho por la exploración aleatoria o el ajuste de estados, y que en ocasiones no deja ver la tendencia.

Al final del experimento, se aprecia como el agente ha conseguido la puntuación máxima.

Previamente también tiene fases de descenso y ascenso, debidas sobretodo a la inestabilidad que genera el sobreoptimismo de las redes DQN, algo que puede ser corregido con un DDQN.

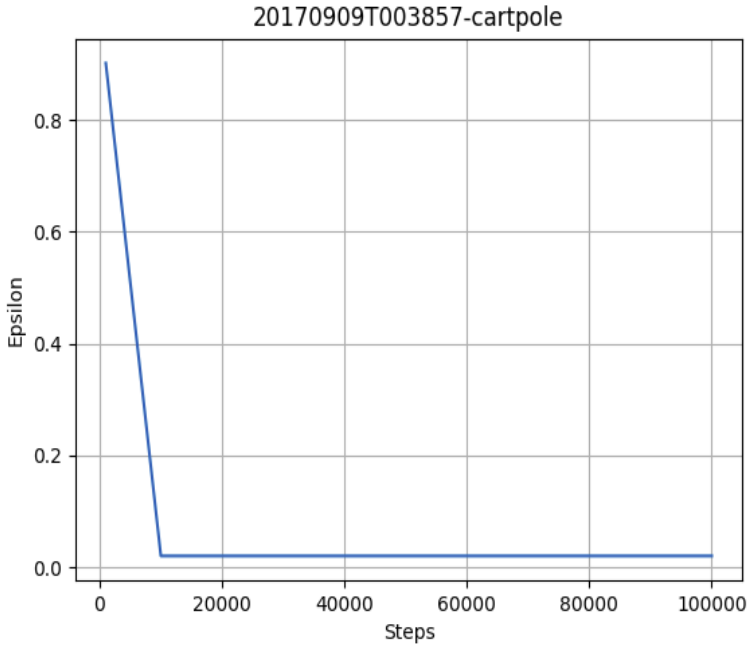


Figura 6-19:Épsilon dqn

Al inicio del experimento deben introducirse suficientes muestras aleatorias en la replay memory para el entrenamiento de la red. Se aprecia de hecho como el agente llega a un máximo relativo al estabilizarse la exploración en 0.01.

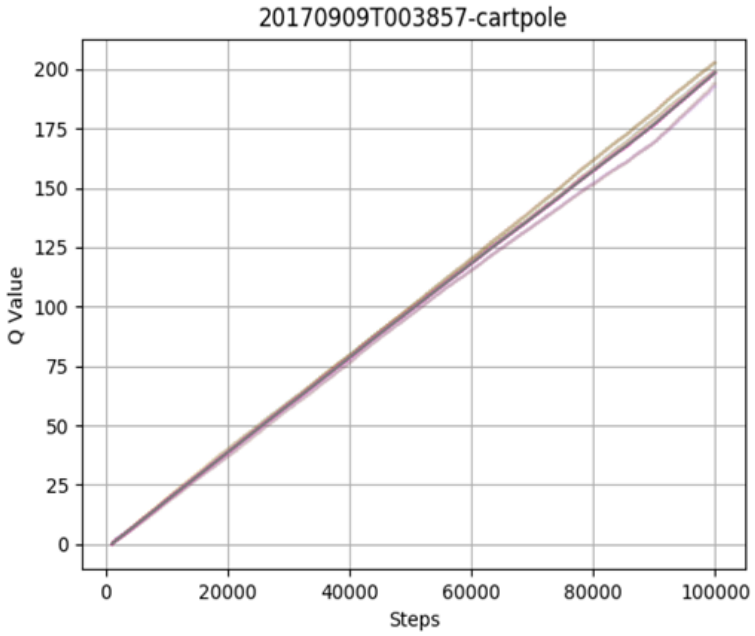


Figura 6-20: Q values dqn

En ocasiones la curva de recompensa no es un buen indicador del desempeño del agente, principalmente por el ruido. Los valores de las Q representan la recompensa esperada por ejecutar una serie de acciones, y permiten observar si la red ha aprendido, se ha estancado o sobreestima su comportamiento. En este caso se comprueba el aprendizaje, ya que el valor esperado es 200.

El learning rate es constante durante todo el experimento e igual a 10^{-3}.

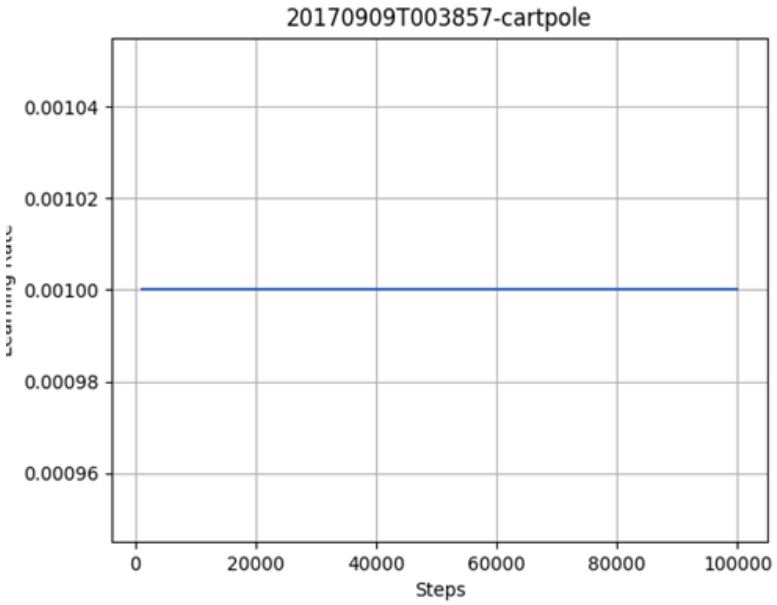


Figura 6-21: Representación del Learning rate

Esta red entrenada con una exploración más aleatoria no termina de converger a la solución final, y presenta más ruido en sus recompensas.

Se puede apreciar, además, como el valor esperado no es muy elevado.

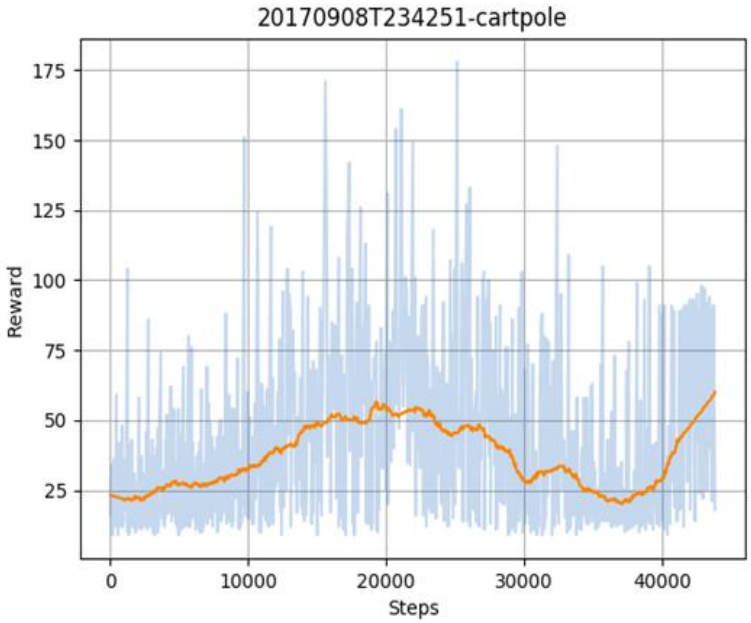


Figura 6-22: Gráfica de recompensa con ruido

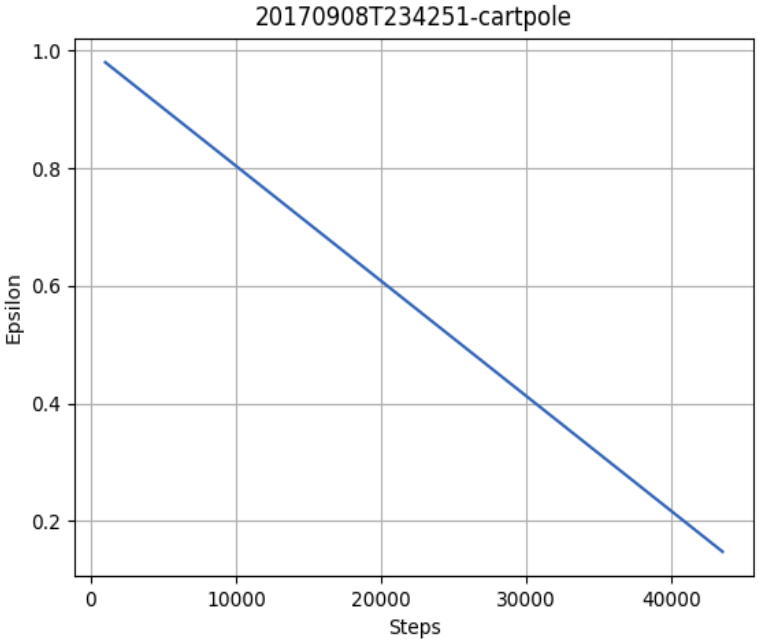


Figura 6-23: Epsilon lineal

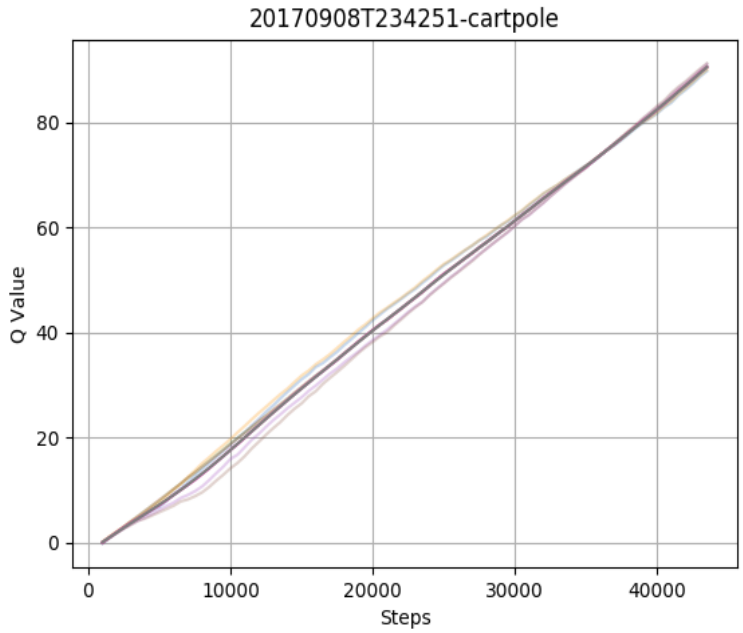


Figura 6-24: Q values ascendentes

Durante el desarrollo de este experimento se pudo cotejar la importancia y cuidado que requieren los datos de entrada de la red.

Se realizó una regularización para asemejar la escala de los valores, sin tener en cuenta que los límites inferior y superior de la velocidad eran infinitos.

Al aplicar la regularización con dichos valores, la velocidad del carrito vista por la red era nula, quedando la red ciega en este sentido e incapacitada para aprender.

El Cartpole con un learning rate de $5 \cdot 10^{-4}$ converge al valor final en 800 episodios. En la gráfica aparece en color azul la recompensa media a lo largo de 100 episodios, y en verde el número de episodios durante los que se ha explorado, de cada 100, es decir, se ha realizado una acción aleatoria, que finaliza a los 300 episodios.

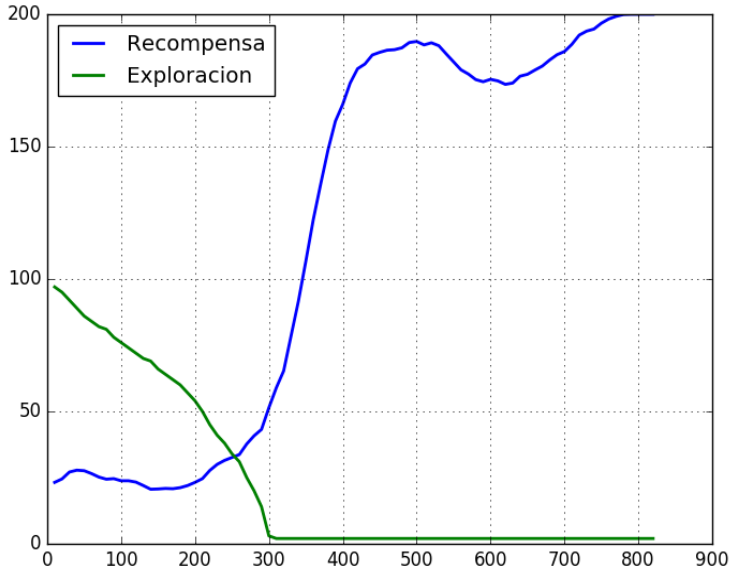


Figura 6-25: CartPole Learning Rate bajo

Incrementar el learning rate a $1.25 \cdot 10^{-2}$ aumenta en gran medida las oscilaciones y la inestabilidad. Los grandes cambios que se realizan al refrescar el valor de los pesos afectan al descenso del gradiente, que salta de un lado al otro de la curva de coste, llegando, en casos extremos, a ascender por ella y subir el error.

Sin embargo, subir el learning rate acelera la velocidad para aumentar la puntuación y acercarse a la solución final. Debe encontrarse un compromiso entre la velocidad de convergencia y la estabilidad relativa.

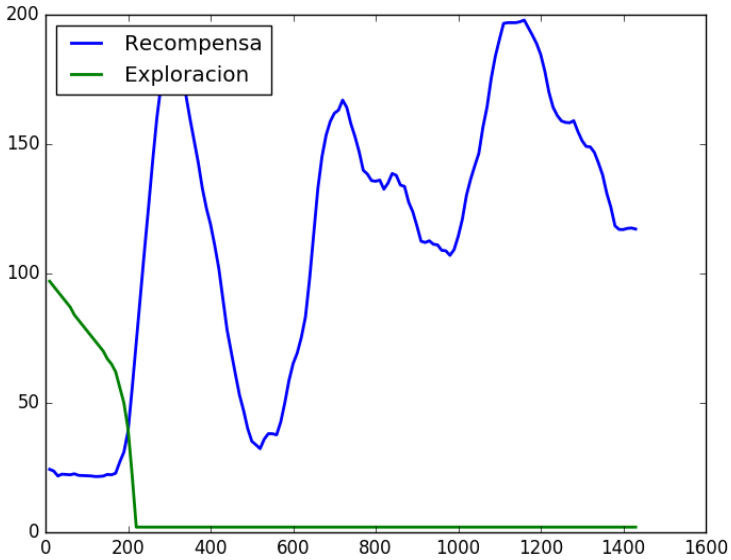


Figura 6-26: CartPole Learning Rate alto

6.3 EXPERIMENTO 5: MOUNTAIN CAR

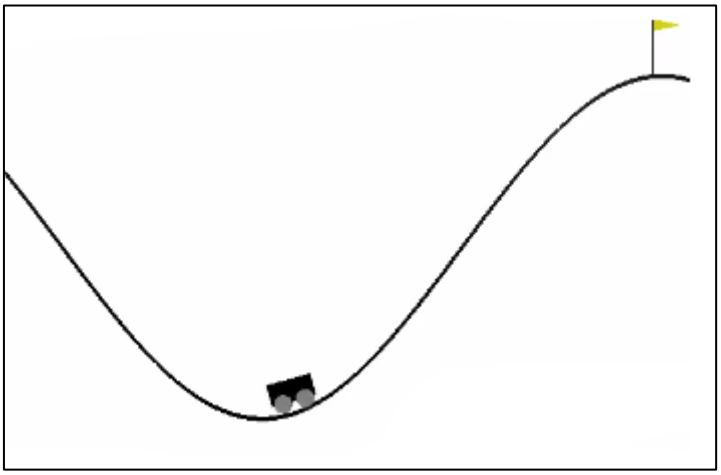


Figura 6-27: Montain car

También hemos utilizado el mountain car un entorno en el que un coche se encuentra entre dos montañas. El objetivo del coche es ascender la montaña de la derecha. Sin embargo, la fuerza del motor no es suficiente, por lo que debe retroceder por la montaña de la izquierda, para ganar momento que le ayude a ascender el obstáculo. El agente tiene 3 acciones, avanzar, retroceder o no hacer nada.

Cada paso que realiza el coche recibe un castigo de -1, que solo finaliza al llegar a la cima. De esta forma, el agente intenta escalar la montaña lo más rápido posible. Si no lo ha conseguido en 200 pasos, el entorno se reinicia.

La observación son dos variables continuas, que representan la velocidad y la posición del coche.

Al utilizar q-learning, ha sido necesario discretizar los valores de la observación, aproximándolos con una red neuronal.

Para resolver el MountainCar se han usado 10 mil pasos de exploración, aproximadamente 50 episodios de un total de 1800, por lo que la exploración es casi nula.

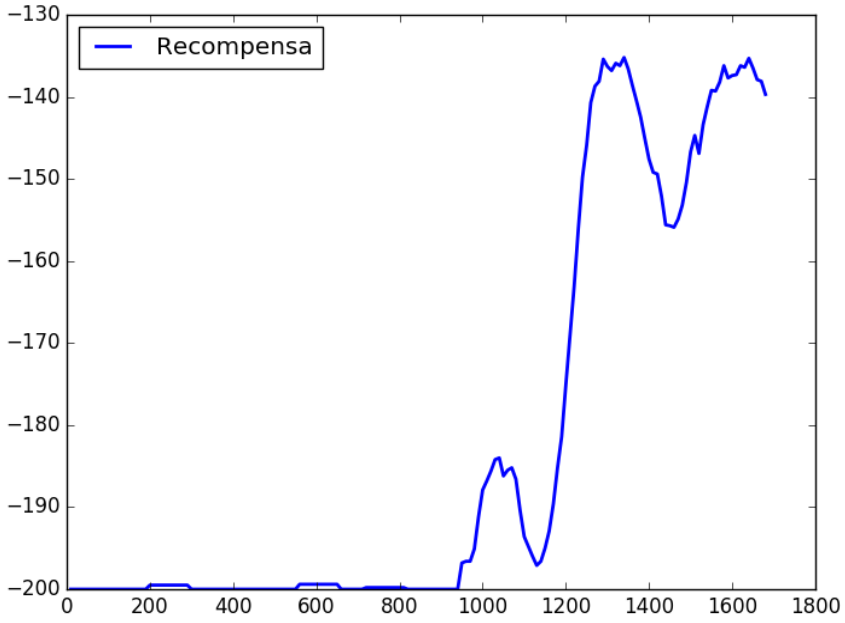


Figura 6-28: Recompensa mountain car (1)

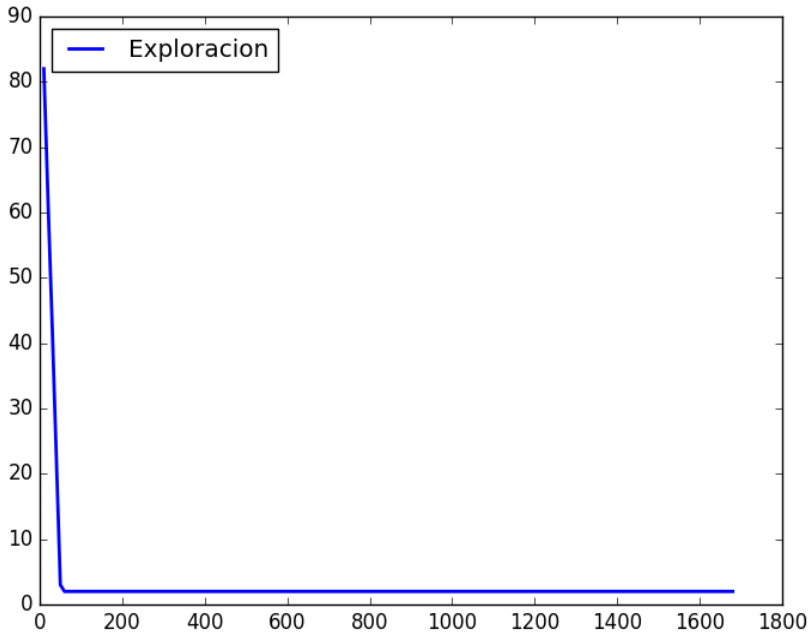


Figura 6-29: Exploración mountain car (1)

Este entorno tiene una dificultad importante, la exploración. El agente sólo recibe una respuesta de que ha llevado una política correcta cuando asciende la cima. Los primeros intentos se realizan con acciones aleatorias, ya que no se ha propagado ninguna recompensa aún. Sin embargo, ascender la cima con acciones aleatorias es estadísticamente muy poco probable.

Sin una estrategia adecuada, el coche terminará todos los episodios sin haber alcanzado su objetivo.

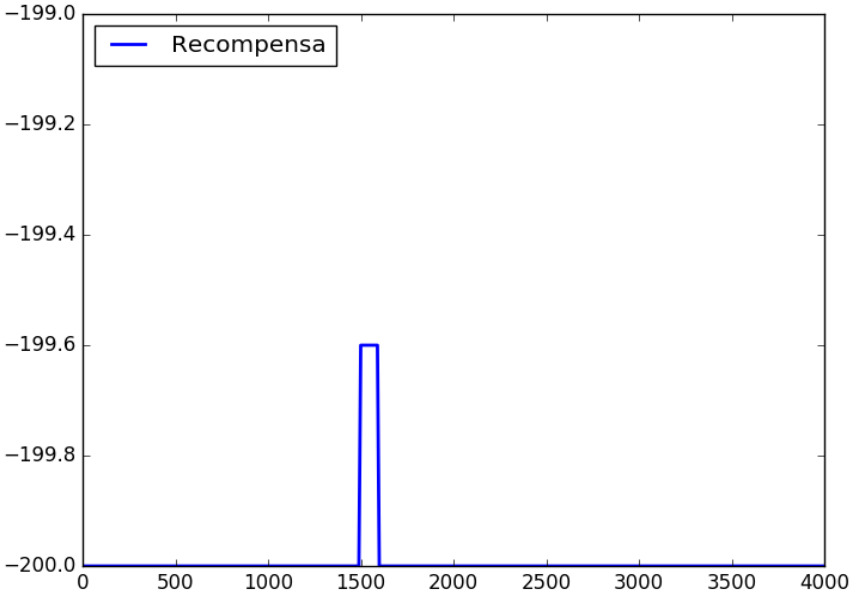


Figura 6-30: Recompensa mountain car (2)

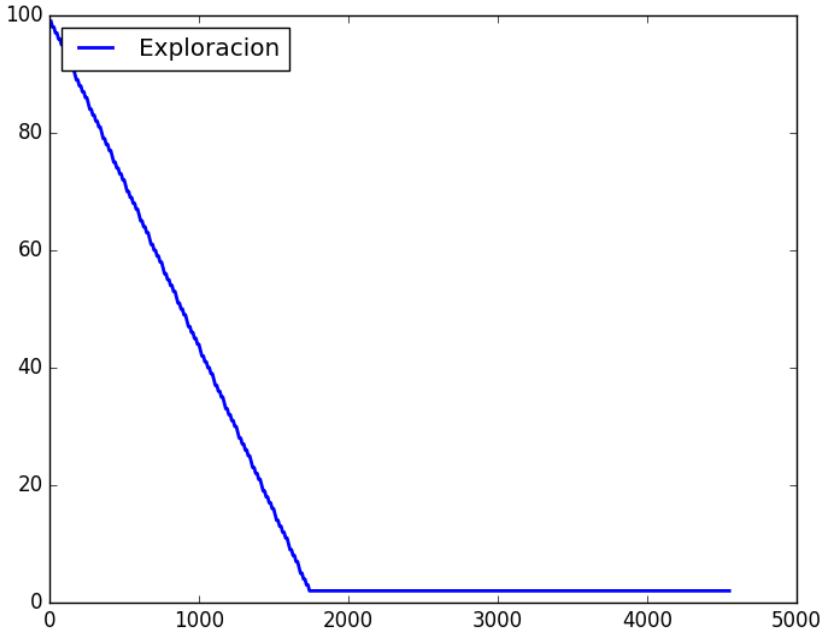


Figura 6-31: Exploración mountain car (1)

Un cambio en la exploración altera radicalmente la salida, que no llega a converger. Para hacer frente a este problema, las estrategias que se realizaron fueron:

El coche no tiene ninguna información de la altura que está ascendiendo, ni de si se está acercando a la meta. Se realizaron pruebas con pequeñas recompensas a lo largo del camino. Inicialmente, convertía en alguno de estos mínimos locales, de los que no salía en el resto de la simulación. Al reducir los valores tuvieron el comportamiento indicado, guiando al coche hasta la cima en mucho menos tiempo.

Reducir el número de decisiones que el agente debe tomar simplifica el problema. Aumentando la velocidad del carrito para que avance un mayor número de pasos, y anulando la acción '*no hacer nada'*, que no aporta ningún valor, le permite llegar antes de que se active el límite de 200 pasos marcado.

Insertando una función que le guíe durante la exploración. Se elaboró una función que movía el coche primero hacia atrás un número de pasos. Ese número de pasos crecía con cada nuevo episodio, hasta que se llegaba a la altura que ofrecía suficiente momento para atacar la cima. Una vez conseguidos algunos éxitos, se anula la función para que el agente aprendiera por sí mismo.

7. ATARI

La compañía Atari además de trabajar en microcontroladores fue pionera en juegos arcade. [25]

Para el aprendizaje del funcionamiento y desarrollo de redes neuronales y machine learning se ha utilizado este tipo de juego.

Las imágenes de Atari tienen una dimensión de 210 x 160 píxeles y 128 colores siendo computacionalmente muy exigente. Por tanto con el fin de simplificar los datos de entrada a la red neuronal ha sido necesario preprocesar la imagen, convirtiéndola a escala de grises y reduciendo la dimensionalidad. El tamaño final de la imagen es de 84 x 84 píxeles, captando exactamente el área de juego.

La profundidad del filtro va a ser siempre la profundidad de la entrada, en escala de grises será 1 mientras que en RGB, 3.

Al haber varios elementos en movimiento en el videojuego el estado no puede quedar definido por una sola imagen, ya que no aporta ni la dirección ni la velocidad de movimiento.

Por tanto, las imágenes se guardan en un vector de 4 en 4, siendo éstos datos la entrada que se le mete a la red neuronal. [26]

Las primeras capas son convolucionales seguidas de un rectificador no lineal. Posteriormente hay una capa fully-connected de 256 unidades. La capa final es una lineal con una sola salida para cada acción tomada.

El número de acciones en estos juegos varía entre 4 y 18.

Para probar esta clase de entorno se ha utilizado el Pong.

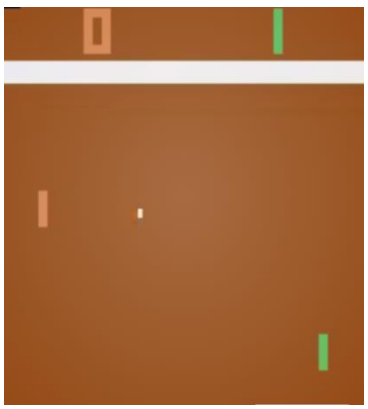


Figura 7-1: Entorno Pong

El agente tiene que evitar que la pelota cruce su lado, recibiendo un castigo de -1 si esto ocurre, mientras intenta marcar a la pala contraria, recibiendo un premio de +1 al conseguir el objetivo.

Los episodios terminan cuando algún jugador llega a 21 puntos.

El entorno tiene dos opciones posibles a la hora de devolver la observación de la pantalla: una opción estocástica, donde el número de salto de los frames varía entre 1 y 4, o una opción determinista, en la que el entorno devuelve una observación cada 4 frames.

8. DDQN

Este algoritmo no sólo produce estimaciones más precisas que el DQN original sino que conduce a puntuaciones mucho más altas en varios juegos.

Esto demuestra que las sobreestimaciones de DQN se ocasionan debido a que utiliza unas políticas demasiado pobres. [27]

La idea de DDQN es reducir la sobreestimación descomponiendo la operación del máximo, del target en dos redes, en la selección y evaluación de la acción.

La red del target en la arquitectura DQN se corresponde con la función de valor, sin introducir redes adicionales. Por lo tanto propone evaluar la política codiciosa de acuerdo a la red neuronal, haciendo uso de la red target para estimar su valor. Para conseguir reducir el problema de las sobreestimaciones DDQN utiliza el siguiente target:

$$y_i^{DDQN} = r + \gamma Q\left(s', \arg\max_{a'} Q\left(s', a'; \theta_i\right); \theta^-\right) \quad (1)$$

DDQN tiene como objetivo conseguir los beneficios de Double Q-learning mientras el resto del algoritmo DQN permanece intacto, y con una sobrecarga computacional mínima.

Además DDQN comparte los mismos hiperparámetros que el DQN original.

8.1 EXPERIMENTO 6: PONG

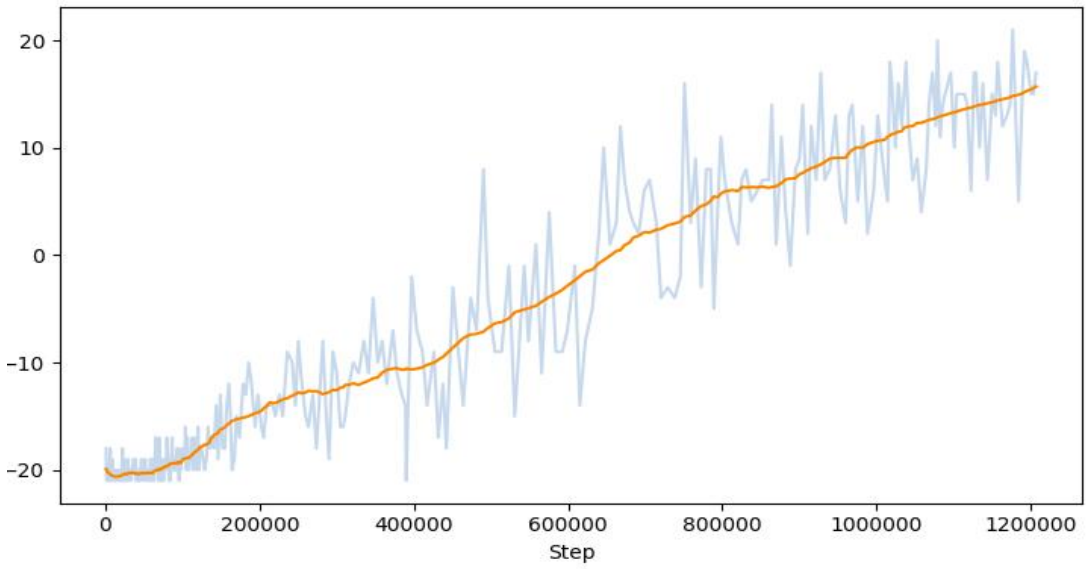


Figura 8-1: Recompensa Pong

Solo se requiere de exploración una sexta parte de la partida, momento del que se dispone de datos suficientemente variados para sacar conclusiones durante el entrenamiento. Esto supone el inicio en la estabilización de los q-values de la red.

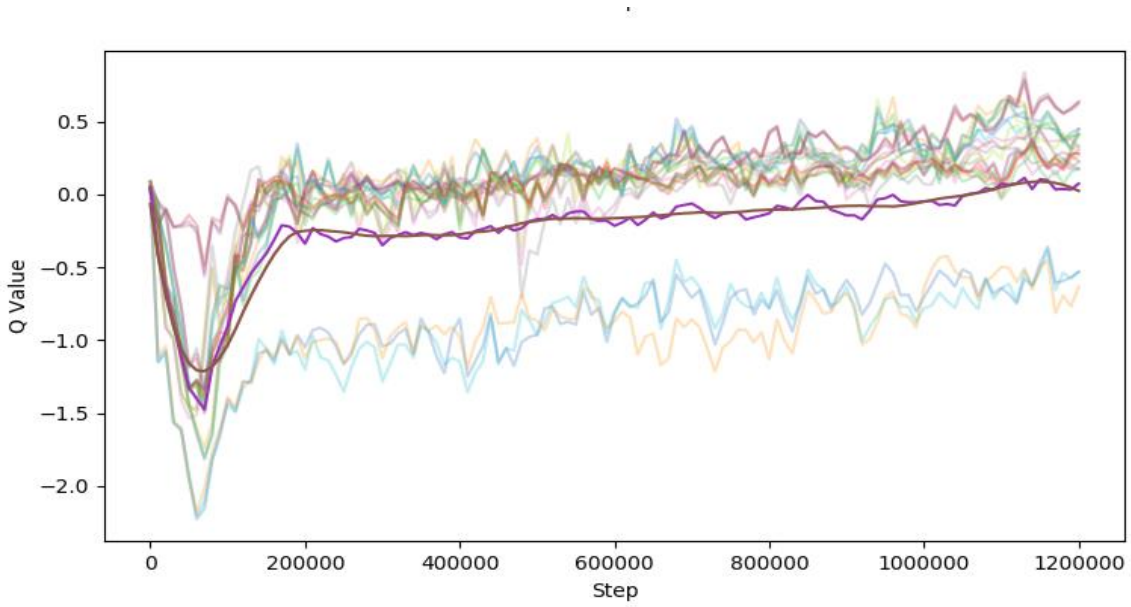


Figura 8-2: Estabilidad de Q values

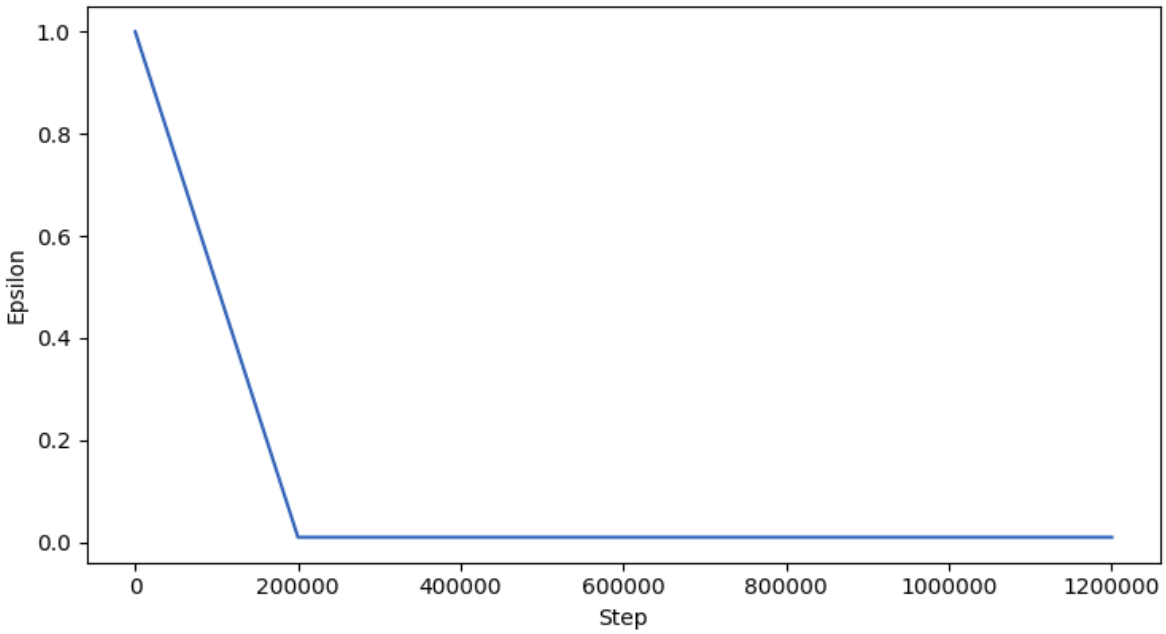


Figura 8-3: Épsilon descendente linealmente

El estado, como se señaló antes, queda definido por 4 imágenes en blanco y negro, recortadas y escaladas.

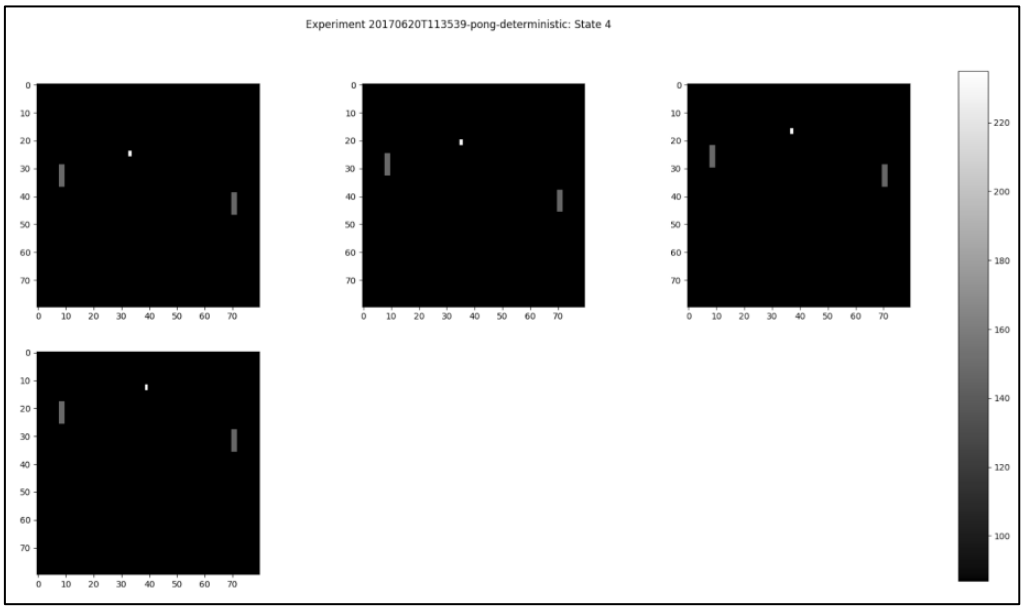


Figura 8-4: Pong escala de grises

Nos encontramos con el problema que al reducir y comprimir la imagen al menor número de píxeles posibles, para acelerar el entrenamiento, la pelota, por su tamaño reducido (4 píxeles), llegó a desaparecer de la imagen, haciendo imposible el aprendizaje. Los valores de la imagen están comprendidos entre 0 y 255.

Para los mismos hiperparámetros del ejemplo anterior, el agente consiguió notables mejoras cambiando únicamente el estado de entrada.

Dos frames consecutivos se binarizan y restan. De esta forma, en una sola imagen se puede distinguir la dirección y velocidad de movimiento.

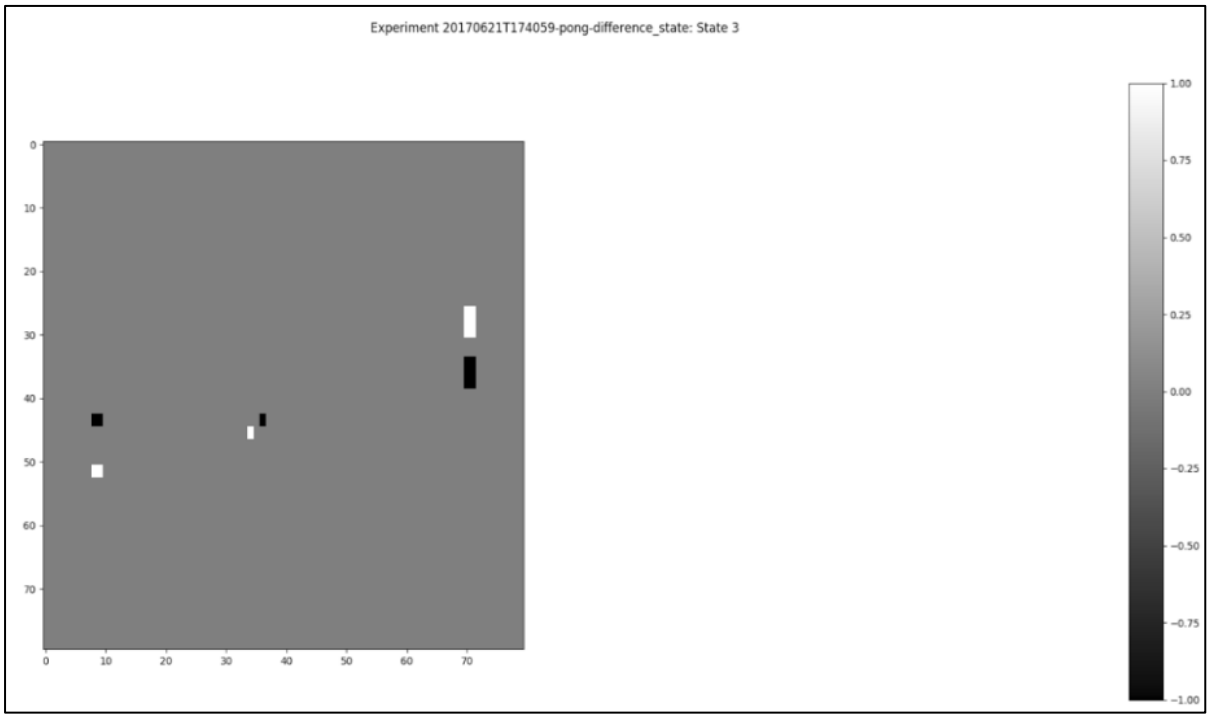


Figura 8-5: Binarización de dos frames consecutivos

Con esto se reducen los datos de entrada, acelerando el algoritmo un 800%. Además, esta representación es más fácil de entender por el agente, que convergía en muchos menos pasos a la solución final.

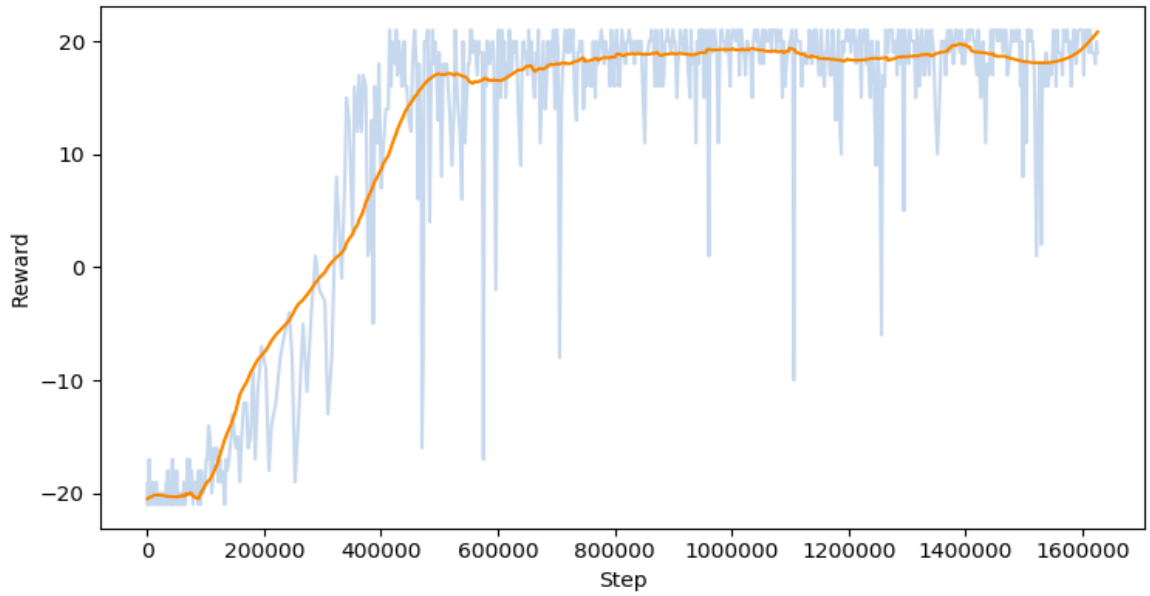


Figura 8-6: Recompensa con la imagen procesada

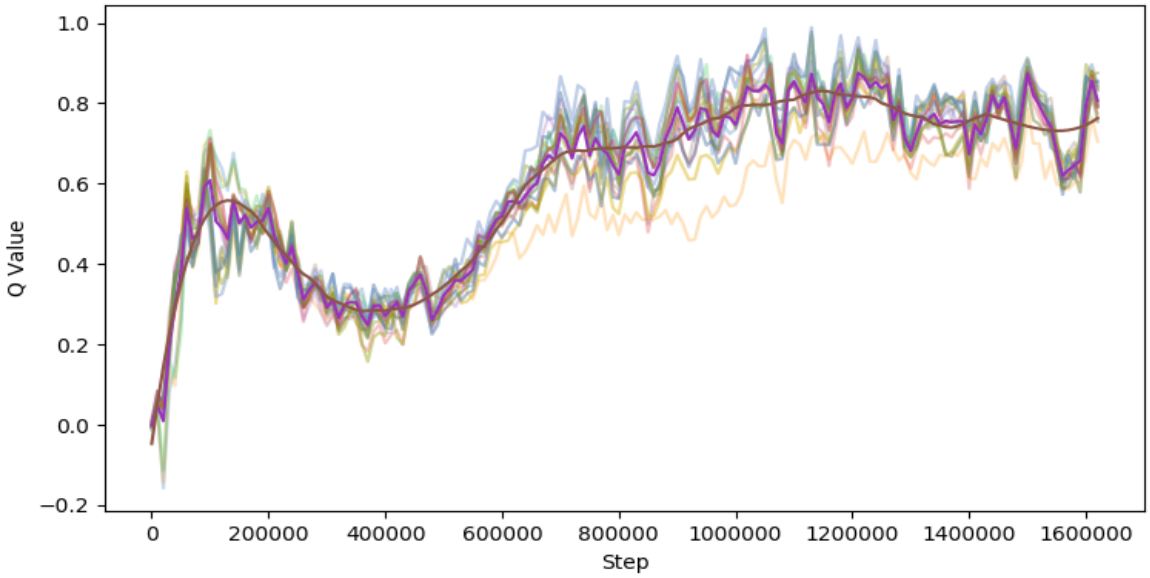


Figura 8-7: Estabilidad de Q values Pong

Para un entorno con un salto de frame estocástico, el aprendizaje es mucho más complejo, no habiéndose conseguido la puntuación máxima con ningún algoritmo implementado, presentando mayor ruido.

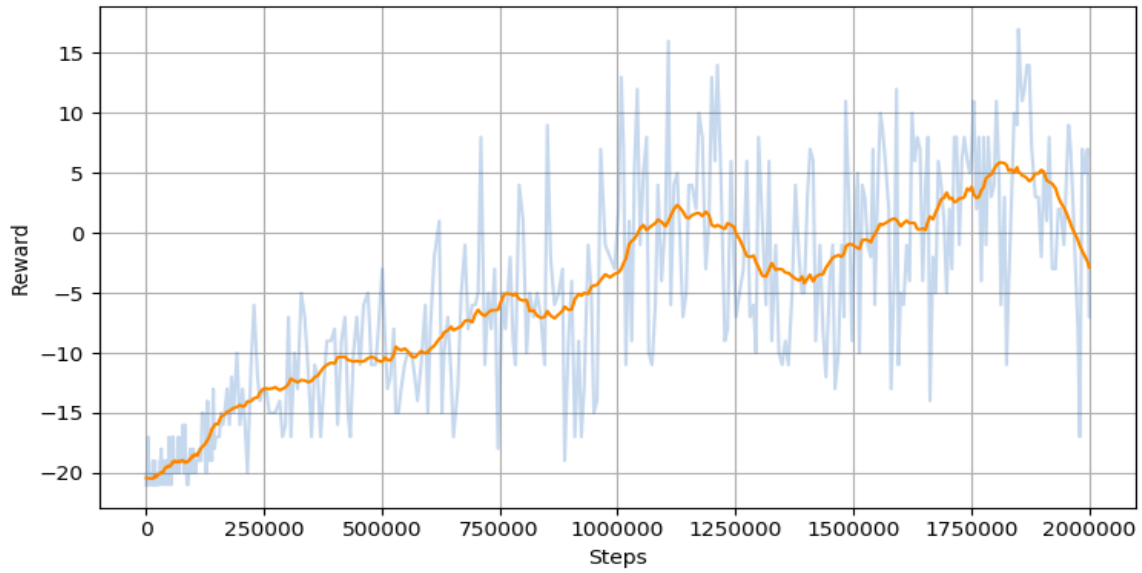


Figura 8-8: Recompensa con entorno estocástico

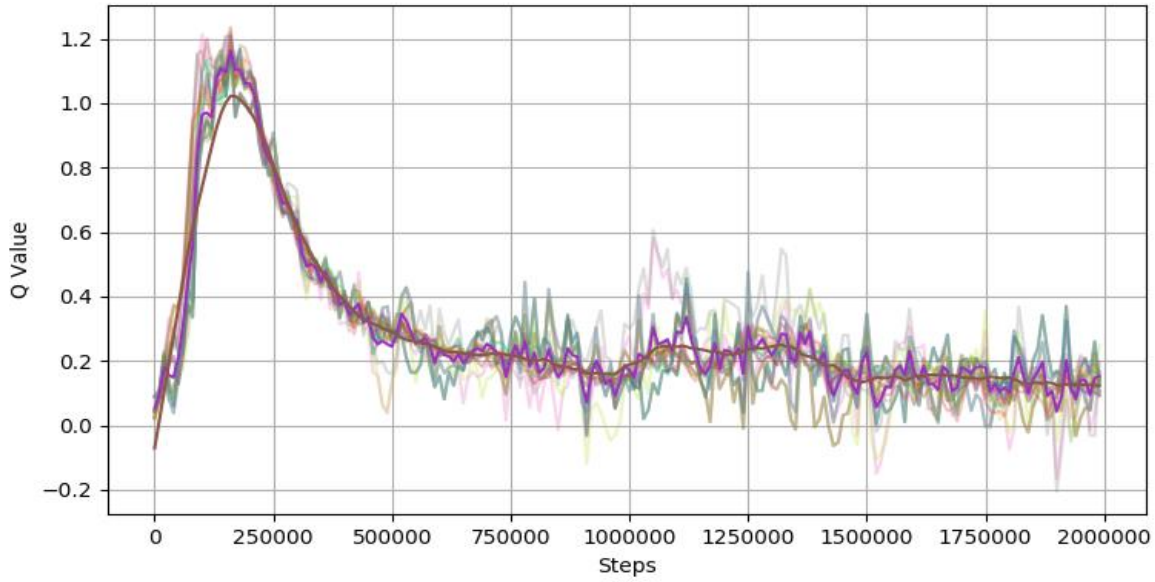


Figura 8-9: Q values en entorno estocástico

Se observa gran diferencia en la velocidad de convergencia entre la observación estocástica y determinista. [25]

Tras renderizar y simular el entorno, se pudo apreciar que, en un entorno determinista, el agente aprendía a golpear de la forma adecuada la pelota al inicio de la partida.

La IA del oponente no es capaz de adaptarse a las jugadas, de forma que al ser marcada, el agente vuelve a tener la opción de saque, y ejecutando la misma estrategia, repitiéndose esta acción en bucle.

En el entorno estocástico, una acción concreta en un estado conocido pueden acabar en varios estados posibles, evitando la opción de repetir la estrategia, y obligándole a generalizar y aprender distintos movimientos, algo mucho más complejo.

9. PRIORITIZED

En principio, se descartan los datos de entrada de inmediato, después de una sola actualización de los pesos de la red neuronal. Esto provoca dos problemas:

- Fuerte correlación en las modificaciones de los parámetros que rompen las variables aleatorias independientes e idénticamente distribuidas (i.i.d) de muchos algoritmos populares basados en gradientes estocásticos (stochastic gradient-based algorithms).
- El rápido olvido de transiciones posiblemente raras que podrían ser útiles más adelante.

La “Experience Replay” aborda ambas cuestiones. [29]

Las transiciones almacenadas en una replay memory consiguen romper las correlaciones temporales mezclando más o menos las transiciones recientes, de forma que no se suele dar el caso en el que una transición se utilice para más de una actualización. Esto se demostró en el algoritmo de DQN (Mnih et al., 2013; 2015) [32]. Con prioritized, las transiciones con mayor error, se consideran más valiosas, ya que se puede aprender más de ellas, y se entrena un mayor número de veces con ellas que con otras con menor error.

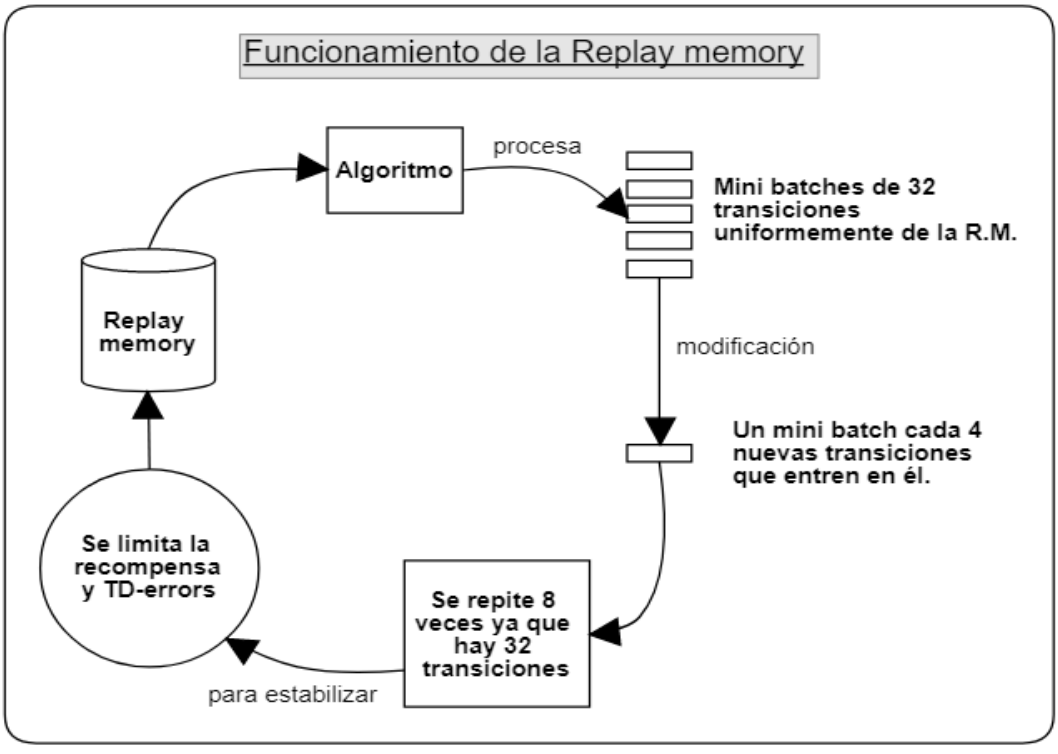


Figura 9-1: Funcionamiento de la replay memory

Se persigue conseguir un uso más eficaz de la replay memory para el aprendizaje, asumiendo que los contenidos en su interior están fuera de nuestro control. [25]

Específicamente DQN utiliza una replay memory muy grande que muestrea de una forma uniforme al azar y revisa dicha transición una media de ocho veces para un “mini-batch” de 32 transiciones. Experience Replay puede reducir la cantidad de transiciones necesarias para aprender.

Se trata de repetir las transiciones más importantes con mayor frecuencia, por lo que se consigue un aprendizaje más eficiente.

Se repiten con mayor frecuencia las transiciones con un temporal difference error (TD-error) mayor, porque son las que tienen la expectativa más alta durante el proceso de aprendizaje, es decir, las más valiosas. Se basa en una selección del estado que se va a modificar posteriormente, priorizando de acuerdo al cambio del valor del estado.

No siempre priorizar según el TD-error es la mejor opción, ya que existen circunstancias en las que la recompensa es muy ruidosa y ésta estimación es muy pobre.

Además, para evitar barridos costosos a lo largo de toda la replay memory, los TD-errors sólo se actualizan para transiciones que se repiten.

Como consecuencia, las transiciones que tengan un TD-error bajo al principio, no saldrán de la replay memory hasta un largo periodo de tiempo, perdiendo parte de la diversidad del entrenamiento.

Para entender los beneficios de la priorización se toma como ejemplo un entorno “cuerda floja” en el que para llegar al destino se debe realizar en todos los pasos la acción correcta.

Como no se obtiene una recompensa hasta llegar al final, requiere de un número exponencial de pasos aleatorios, para ser más exactos, la probabilidad de que una secuencia aleatoria de acciones obtenga una recompensa positiva es de 2^{-n} .

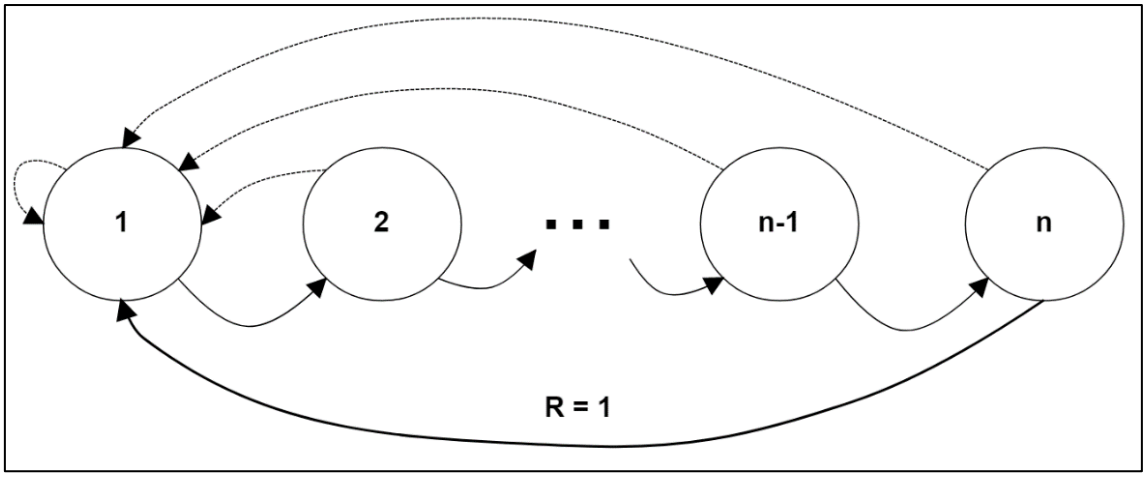


Figura 9-2: Entorno de "cuerda floja"

Con la repetición de las transiciones utilizando Prioritized se ha comprobado que existe una mejoría exponencial respecto a la repetición uniforme que se ejecuta en DQN. [24]

La falta de diversidad en las transiciones hace que se produzcan a veces ajustes excesivos (overfitting). Para suplir este problema se introduce un método de muestreo estocástico, es decir, no determinista, que interpole entre la priorización puramente codiciosa y la aleatoria.

Esta priorización puede llevar a una pérdida de diversidad, la cual es solventada con una priorización estocástica e introduciendo el bias. Se define la probabilidad de muestreo de una transición como:

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}$$

Donde p_i es la prioridad de la transición 'i' siempre mayor de 0 y α determina el grado de priorización al muestrear las transiciones.

Si α es 0 el muestreo es totalmente uniforme, mientras que si α toma el valor de 1, el muestreo es codicioso.

La primera variante que se considera es la priorización directa proporcional, $p_i = |\delta_i| + \varepsilon$, donde ε es una pequeña constante de carácter positivo, que previene que no haya transiciones sin revisar una vez el error es cero.

La segunda variante es una priorización indirecta basada en el rango de las transiciones, $p_i = \frac{1}{rank(i)}$. El problema de esta segunda priorización es que es más robusta porque no es sensible a valores atípicos.

Para corregir el bias introducido se utiliza importance-sampling (IS) en los pesos de la red.

$$\omega_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta}$$

Por razones de estabilidad conviene normalizar los pesos por $1/\max_i \omega_i$.

9.1 EXPERIMENTO 7: PONG PRIORITIZED

El algoritmo permite un aumento de velocidad muy importante respecto a un DQN simple. Los primeros 250000 pasos son utilizados para explorar y obtener una experiencia variada. Tras esto el algoritmo llega a la puntuación máxima rápidamente.

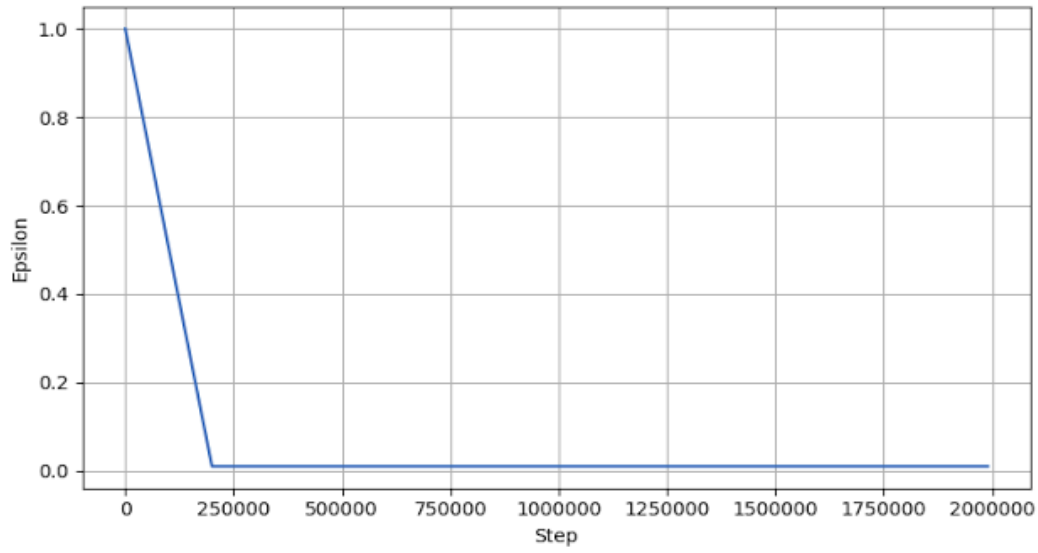


Figura 9-3: Epsilon prioritized

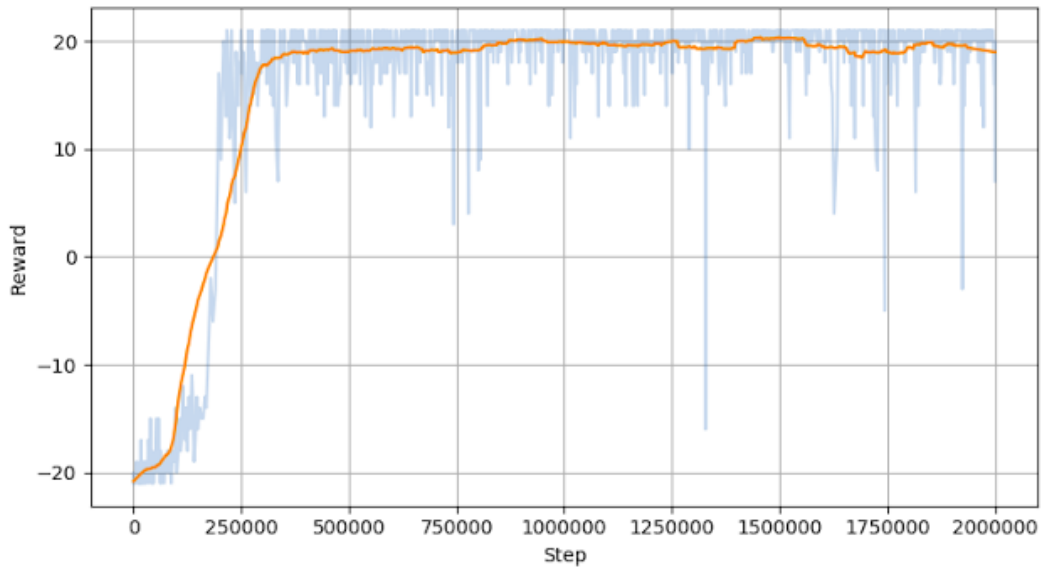


Figura 6.2: Recompensa prioritized.

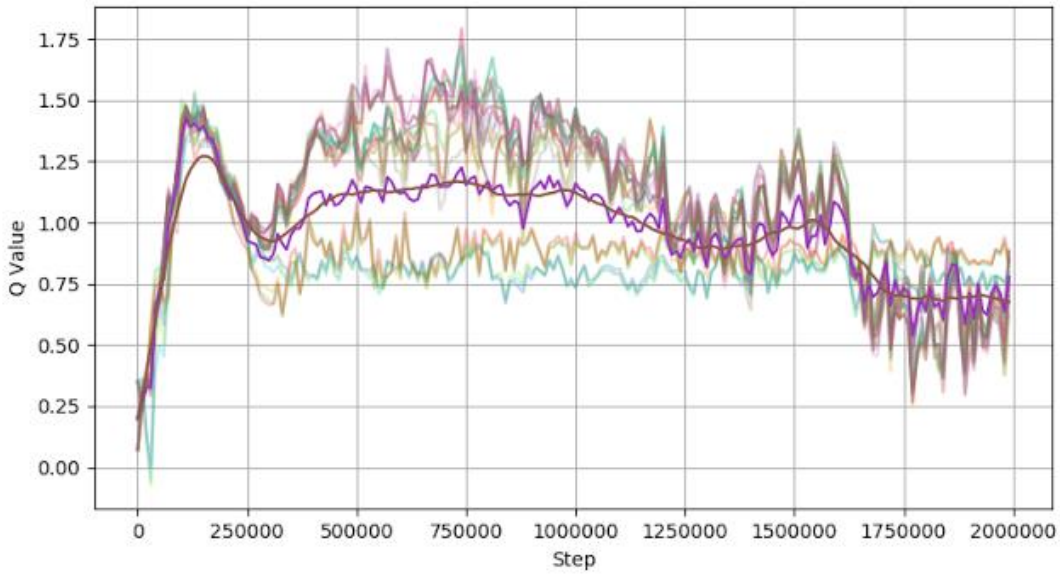


Figura 6.3: Q values prioritized.

Como se puede observar, el agente presenta las dificultades citadas antes, a la hora de converger con un muestreo estocástico, a pesar de disponer de 10 millones de pasos. [29]

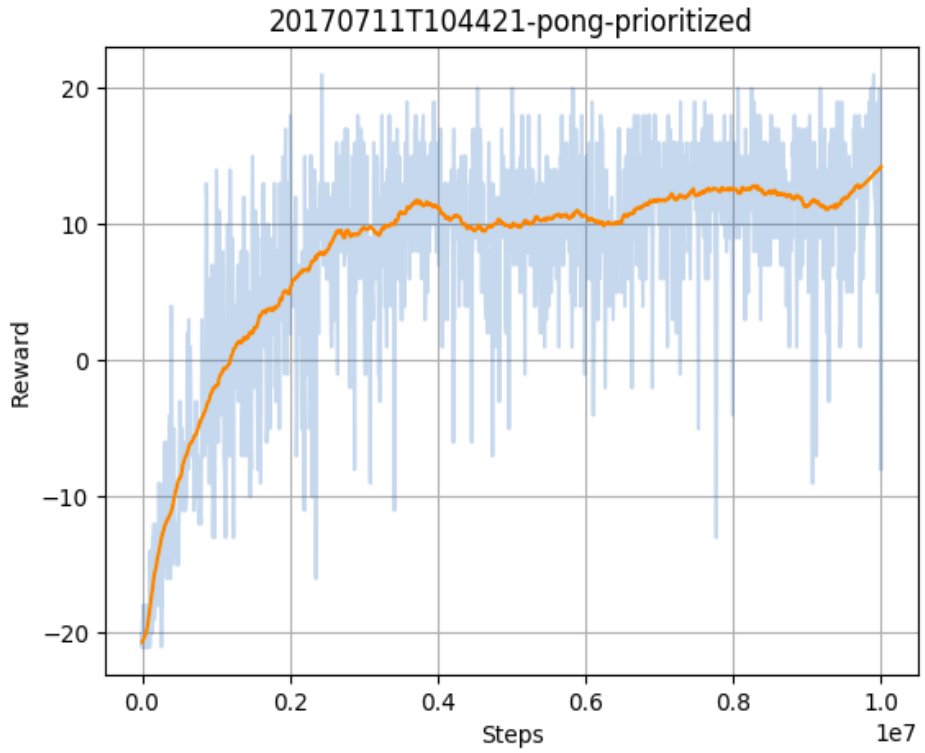


Figura 6.4: Recompensa prioritized estocástico.

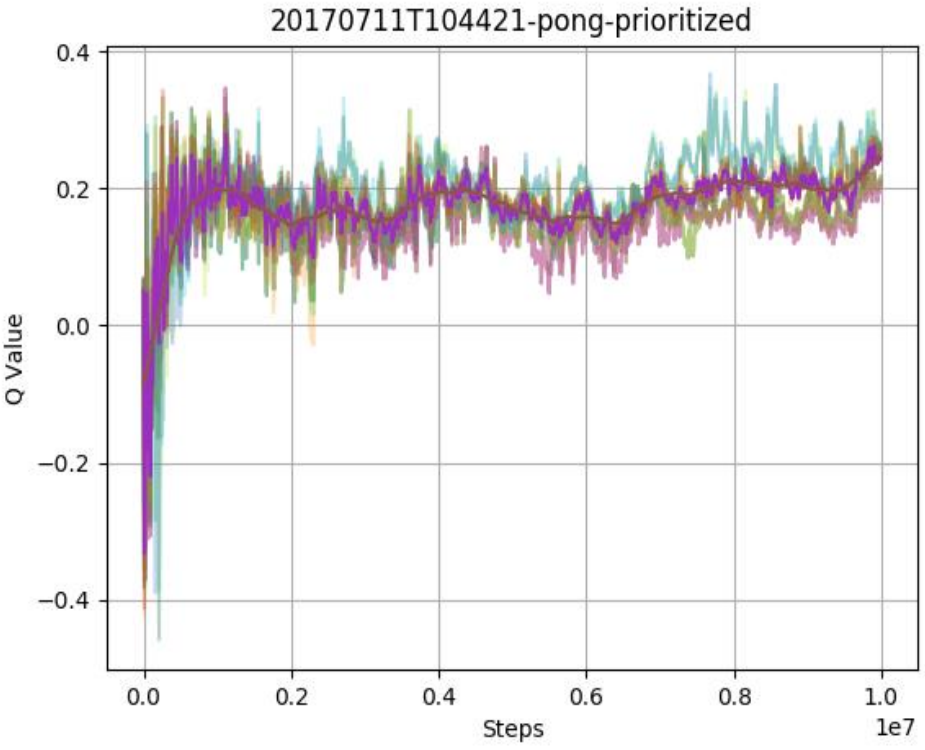


Figura 6.5: Q values prioritized estocástico.

10. DUELING

Ésta arquitectura de red separa explícitamente la representación de valores de estado V(s), que manifiesta cómo de bueno es un estado, y ventajas de acción “state-dependent action advantages” A(s, a), que contiene el vector de acciones óptimas para dicho estado. [28]

Separadas en dos redes, compartiendo ambas el módulo de aprendizaje convolucional como en el DQN original.

Las dos redes se combinan a través de una capa especial de agregación para producir una estimación de la función de valor estado-acción Q(s, a) “state-action value function”.

El principal beneficio de usar esta factorización es el de generalizar el aprendizaje de las acciones sin imponer ningún cambio en el algoritmo, sólo en la estructura de la red.

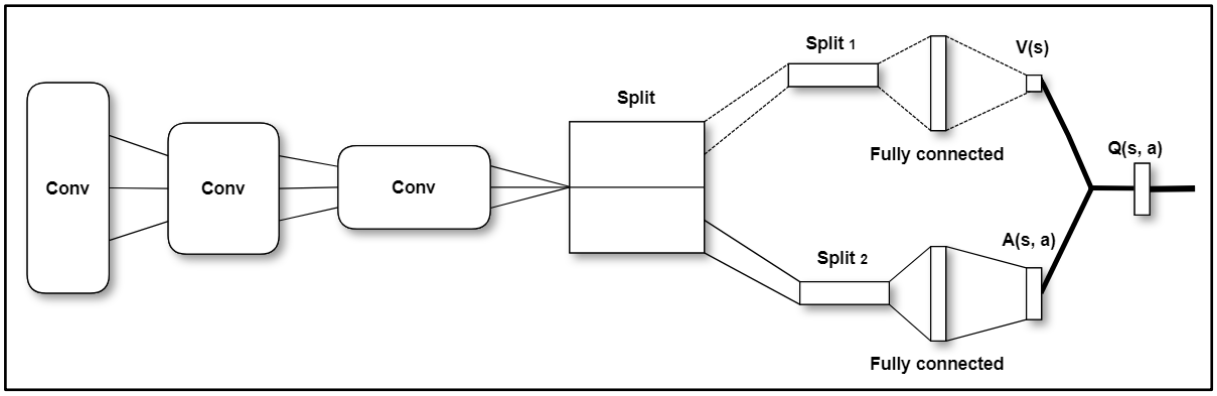


Figura 9.1: Red Dueling.

Intuitivamente la arquitectura del Dueling puede aprender qué estados son o no valiosos, sin tener que aprender el efecto de esa acción para dicho estado. Esto es particularmente útil en estados donde sus acciones no afectan al entorno de una forma relevante.

Para entender mejor el comportamiento de esta arquitectura se va a tomar como ejemplo el clásico juego de Atari Enduro, el cual consiste en recorrer con un vehículo una carretera sin salirse y esquivando los obstáculos que van apareciendo.

En los entrenamientos con la arquitectura Dueling se ha observado que la red de los valores del estado V(s) presta atención al horizonte de la carretera y la red de la ventaja de la acción A(s, a) aprende a fijarse sólo cuando hay objetos inmediatamente en el frente, con el fin de evitar colisiones.

En los experimentos que se han realizado hasta la fecha se ha demostrado que la arquitectura del Dueling puede identificar más rápido la acción correcta durante la evaluación de la estrategia, haciendo acciones parecidas durante el aprendizaje.

Para un agente que se comporta de acuerdo a una política π, los valores del estado V(s) y la función Q(s, a) se definen de la siguiente forma:

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t \mid s_t = s, a_t = a, \pi \right]$$

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^{\pi}(s,a) \right]$$

Siendo 's_t' el número de imágenes por el tiempo que conlleva un paso (step). El agente busca maximizar el retorno esperado descontado, el cual se define como R_t.

La función anterior para calcular la Q(s, a) se puede calcular de forma recursiva con la dinámica de programación:

$$Q^{\pi}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \mathbb{E}_{a'\sim\pi(s')}\left[Q^{\pi}(s',a') \right] \mid s,a,\pi \right]$$

A continuación definimos la Q(s, a) óptima $Q^{*}(s,a) = \max_{\pi}(s,a)$.

Bajo una política determinista $a = \arg\max_{a'\in\Lambda} Q^{*}(s,a) = \max_{\pi}(s,a)$, siendo Λ el conjunto discreto de acciones.

Por tanto la función para obtener los valores del estado es $\mathrm{V}^{*}(\mathrm{s}) = \max_{a} Q^{*}(s,a)$.

Como resultado obtenemos que la función óptima de Q satisface la ecuación de Bellman.

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

Se define otra variable importante como es la función de ventaja (Advantage function), relacionando las funciones del valor V(s) y de Q:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s).$$

La función V(s) mide la bueno que es estar en un estado particular ‘s’.

La función Q, sin embargo, mide el valor de la elección de una acción en particular cuando se encuentra en ese estado.

La función ventaja resta el valor del estado de la función Q para obtener una medida relativa de la importancia de cada acción.

Las funciones V(s) como se describen anteriormente son objetos con dimensiones altas. Para aproximarlos podemos usar una red profunda $Q(s,a;\theta)$ con parámetros θ.

Para estimar esta red o cadena, optimizamos la siguiente secuencia de funciones de pérdida en ‘i’ iteraciones:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right]$$

Siendo $y_i^{DQN} = r + \gamma \max_{a'} Q(s',a';\theta^-)$, donde θ^- representa los parámetros de una red fija y separada de destino “target network”.

Para poder estimar los valores es necesario congelar los parámetros de la red de destino $Q(s',a';\theta)$ para un número fijo de iteraciones mientras se actualiza la red en línea $Q(s,a;\theta_i)$ utilizando el descenso del gradiente. Con esto mejoramos de forma sustancial la estabilidad del algoritmo.

Durante el aprendizaje el agente acumula un conjunto de datos ‘D’ de transiciones $e_t = (\mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, \mathrm{s}_{t+1})$ durante varios episodios.

Al entrenar la Q-network, en lugar de usar sólo la transición actual según lo prescrito por el aprendizaje estándar, la red se entrena mediante el muestreo de mini-batches de experiencias de ‘D’ uniformemente al azar.

La secuencia de las pérdidas queda de la siguiente forma:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right]$$

Hasta ahora se han descrito los componentes principales de DQN.

Se introduce una mejora en el algoritmo denominada doble DQN “DDQN”. En Q-learning y DQN, el operador ‘max’ utiliza los mismos valores tanto para seleccionar como para evaluar una acción. Por tanto, esto puede conducir a sobreestimaciones de los valores. Para conseguir mitigar este problema DDQN utiliza el siguiente objetivo (ecuación (1)).

Hasta el momento se define de la misma manera que en DQN, reemplazando únicamente y_i^{DQN} por y_i^{DDQN}.

La arquitectura del Dueling se basa en que en algunos estados es de suma importancia saber que acción tomar, pero en muchos otros la elección de la acción no tiene repercusión en lo que sucede. En cambio en el DQN original la estimación de los valores de estado es de gran importancia para todos los estados.

Ésta arquitectura puede ser entrenada con muchos algoritmos existentes como DDQN o SARSA y aprovechar las mejoras de los mismos, como mejores políticas de exploración o replay memories más desarrolladas.

Considerando la red de Dueling representada en la figura (9.1), donde se parte en dos corrientes, la del valor del estado $V(s;\theta,\beta)$ y la función de ventaja $A(s,a;\theta,\alpha)$; donde θ son los parámetros de las capas convolucionales y α y β los parámetros de las capas Fully connected; y haciendo uso de la función de ventaja se llega al siguiente resultado:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha)$$

Pero hay que tener en cuenta que $Q(s,a;\theta,\alpha,\beta)$ es sólo una estimación de la verdadera parametrización de la función Q.

Por lo que sería erróneo concluir que $V(s;\theta,\beta)$ es un buen estimador de la función de valor del estado debido a que existe una falta de visibilidad que se refleja en un pobre rendimiento cuanto se utiliza directamente esta función.

Para solucionar este problema de visibilidad, se fuerza la función que estima la ventaja a no tener ninguna en la acción elegida. Con ello se asegura que por un lado una corriente aprende los V(s) y por el otro, otra corriente los A(s, a).

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |A|} A(s,a';\theta,\alpha) \right)$$

A la hora de programar las ecuaciones se sustituye el máximo por un promedio. Con ello aumenta la estabilidad de la optimización.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|} \sum_{a'} A(s,a';\theta,\alpha) \right)$$

Con Dueling es común utilizar rectificadores no lineales, para ayudar a la convergencia de modelos profundos, entre todas las capas adyacentes. Por lo general se utilizan las Relus, dando más flexibilidad para explorar arquitecturas de red más potentes.

10.1 EXPERIMENTO 8: PONG DUELING

Aplicando el algoritmo de Dueling con una política determinista se logran mejores resultados puesto que este método aprende muy bien cuando hay acciones que no afectan de manera relevante al entorno.

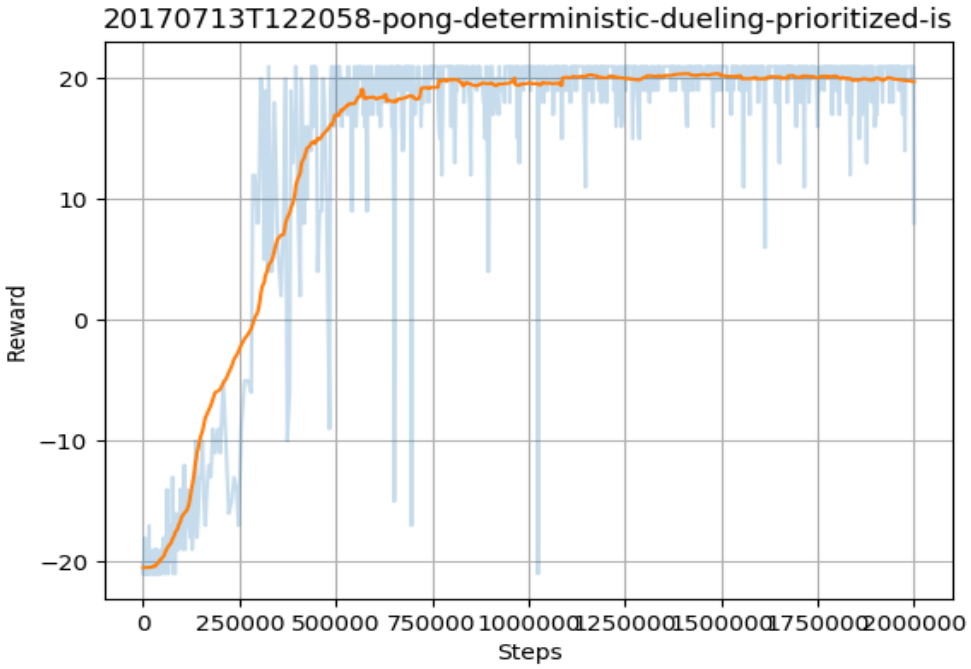


Figura 5.1: Recompensa Pong con Dueling

En tan sólo 250000 pasos ya comienza a obtener resultados de 21-0. Los hiperparámetros utilizados son los mismos que en los anteriores algoritmos. [28]

11. HIPERPARÁMETROS

Los agentes de Deep Reinforcement Learning permiten ajustar el valor de las variables de las que dependen los sistemas de control, pero son en sí mismos sistemas de control muy complejos. No solo por tener un gran número de hiperparámetros, sino por constar de dos partes a ajustar: el agente propiamente dicho, y la red neuronal que ajusta el valor de los pesos.

Esta combinación suele ser inestable, necesitando de técnicas para combatirlo, y para conseguir la convergencia a una solución en un tiempo aceptable, evitando los mínimos locales.

Los algoritmos desarrollados constan de multitud de hiperparámetros. Para definirlos los dividimos en grupos:

- Parámetros del QNetwork:
  - Learning rate, el cual varía entre 0,025 y 0,000025 en los experimentos realizados.
  - Learning rate decay, que es una función de disminución exponencial, ya que a menudo se recomienda reducir el learning rate a lo largo del entrenamiento.
- Parámetros del experimento:
  - Load path, carga un experimento desde la ruta especificada.
  - Load replay memory, guarda el contenido de la memoria durante el experimento.
  - Load q network, cuya función es la de cargar las redes Q.
  - Sabe base path, para guardar los resultados.
  - Populating replay memory, el número de pasos necesarios para llenar la replay memory.
  - Learning steps, número de pasos que se utilizarán durante el aprendizaje. Normalmente varía entre 100.000 y 2.000.000.
  - Max episode steps, para terminar los episodios al llegar a una serie de pasos.
- Parámetros simples del experimento:
  - Initial epsilon, valor inicial, normalmente 1.
  - Final epsilon, valor final, suele estar comprendido entre 0 y 0,6.
  - Epsilon decay rate, política de reducción exponencial. Generalmente comprendido entre 0,01 y 0,00001.

  - Super action update frequency, la acción que se devuelve para cada etapa del aprendizaje, si el valor es 1, se encuentra deshabilitada.
  - Replay memory max size, es el tamaño máximo de la memoria. Estos valores pueden variar sustancialmente dependiendo de la diversidad de datos que se necesiten para evitar la correlación. En las pruebas realizadas se ha estimado entre 1.000.000 y 10.000.000.
  - Replay memory batch size, determina el tamaño de los batches, los cuales almacenan las transiciones que se van a entrar en la red.
  - Discount factor, la tasa de descuento durante el entrenamiento. Suele estar comprendida entre 0.9 y 1, a medida que aumenta tiene en cuenta la recompensa obtenida en un plazo mayor.
- Parámetros del Target:
  - Replay memory priority offset, desplazamiento utilizado al calcular la prioridad. Evita tener entradas con prioridad 0.
  - Replay memory priority exponent, exponente utilizado al calcular la prioridad.
- Parámetros del análisis:
  - Analysis states, lista de los estados que se utilizarán en el análisis.
  - Analysis steps, realiza un análisis cada número de pasos fijado.
- Parámetros de la red neuronal: El número y tipo de capas utilizado, la cantidad de neuronas por capa, la clase del optimizador o la función de activación son algunos de los muchos parámetros que se pueden encontrar a la hora de diseñar la red.

11.1 CHECK LIST

Debido a la gran cantidad de acciones y de hiperparámetros a tener en cuenta para realizar una prueba, ha sido necesario crear una check list como herramienta de ayuda para verificar que se siguen los pasos necesarios para la ejecución de un experimento correctamente.

La estructura de la lista se divide en dos partes:

Una primera, previa a la ejecución del experimento, en la que se revisan los valores de hiperparámetros, se comprueba que los ficheros son los correctos y que la red neuronal se encuentre bien definida.

Y una segunda parte, en la que se obtienen los datos, se realiza un análisis y finalmente se representa el resultado por pantalla para llegar a las conclusiones.

Esta lista se ha ido modificando a medida que han ido apareciendo nuevos fallos.

Antes de ejecutar el experimento:

☐ Comprobar que es el fichero que de verdad queremos cambiar

☐ Comprobar que se cargan los archivos: QNetworks, LoadPath… y Max episodes step.

☐ DQN

- ☐ Learning Rate
- ☐ Learning Rate decay
- ☐ Learning steps
- ☐ Initial epsilon
- ☐ Final epsilon
- ☐ Epsilon decay rate
- ☐ Superaction update frequency
- ☐ State frame number
- ☐ State group size
- ☐ Replay memory max size
- ☐ Replay memory batch size
- ☐ Discount factor
- ☐ Target update frequency
- ☐ Replay memory priority offset
- ☐ Replay memory priority exponent
- ☐ Analysis steps
- ☐ Training frecuency

☐ A3C

- ☐ Agents
- ☐ Optimizers
- ☐ Learning steps
- ☐ Discount factor
- ☐ n_step return

☐ Learning rate

- ☐ Loss value
- ☐ Loss entropy
- ☐ Min batch

☐ Initial epsilon

- ☐ Final epsilon
- ☐ Epsilon decay

☐ Step group size

- ☐ Analysis steps

☐ Plotear el epsilon antes del experimento. Lineal vs exp

☐ Analysis states

☐ Entorno

- ☐ Posición 1 y Posición 2

☐ Comprobar Original vs Postprocesado por si se pierde información

☐ Ver estados que le entran a la red: columnas, filas, dimensiones...

☐ Red neuronal

- ☐ Estructura y número
- ☐ Activaciones e inicializaciones
- ☐ Data format y padding
- ☐ Número de entradas y salidas
- ☐ Optimizador

☐ Simple, target, double, prioritize

- ☐ Posición 1 y Posición 2

☐ Función de exploración o selector de acciones

☐ Cualquier nueva acción añadida

☐ Entra en TMUX

Después del experimento:

☐ ANALYSIS

- ☐ Rewards
- ☐ Epsilon
- ☐ Qvalues
- ☐ State Values

☐ Los estados y acciones que estamos ploteando

☐ RENDER

- ☐ Eliminar el límite de pasos
- ☐ Comprobar los rewards que devuelve el entorno
- ☐ Environment
- ☐ Modelo de red

11.2 ESTRATEGIAS DE SELECCIÓN DE HIPERPARÁMETROS

Al tener que tunear en el sistema de control muchos hiperparámetros, existen distintas estrategias a seguir.

La primera estrategia que se ha utilizado para encontrar los valores óptimos de los hiperparámetros ha sido Grid Search, la cual funcionaba bien en los primeros experimentos realizados, puesto que los entornos eran simples.

Posteriormente se ha buscado otra estrategia de selección de hiperparámetros, Random Search, obteniendo resultados generalmente algo peores que en la anterior, por lo que se ha desechado para el resto de experimentos.

La última estrategia que se ha probado es Bayesian, obteniendo unos resultados sustancialmente mejores. Se ha comprobado con los juegos Cartpole y Pong.

11.2.1 Grid search

Se escogen los parámetros que se desean tunear y el rango de los valores de los mismos. Ésta estrategia se basa en realizar una cuadrícula para evaluar las distintas combinaciones que existen entre los parámetros seleccionados.

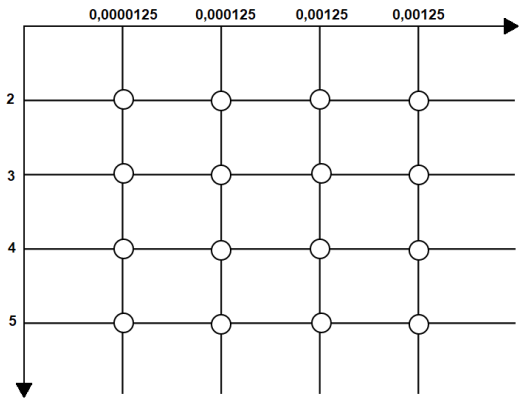


Figura 11.1: Grid Search.

11.2.2 Random search

Es un proceso de búsqueda aleatoria. Es muy posible que esta estrategia no encuentre un resultado tan bueno como la anterior en el mismo tiempo. Este método trabaja muy bien con parámetros continuos, ya que la búsqueda en este caso es más fina.

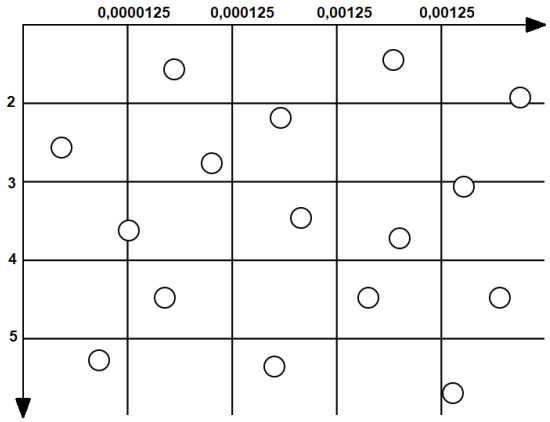


Figura 11.2: Random Search.

11.2.3 Bayesian

Se trata del proceso más novedoso. Comienza siendo prácticamente aleatorio, normalmente durante el 10% del entrenamiento. A partir de los resultados obtenidos en este tramo, hace énfasis en las zonas donde los Q values han sido mejores, de forma cada vez más greedy. Tiene una parte denominada Gaussian process en la que estudia la incertidumbre de distintos puntos, y otra parte llamada Acquisition function que se encarga de seleccionar los puntos con más o menos incertidumbre para conseguir llegar a la función final. [30]

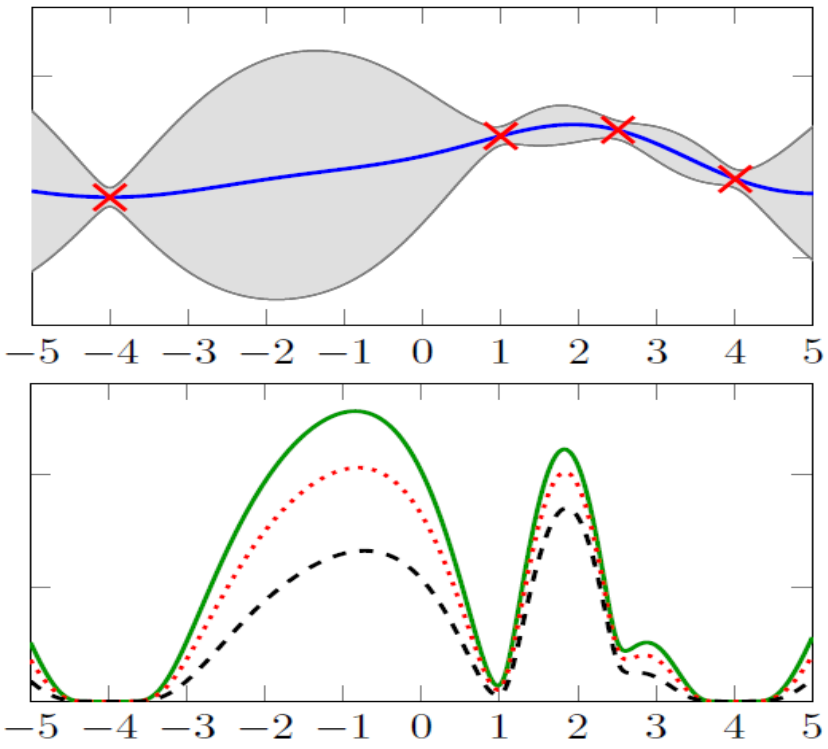


Figura 11.3: Optimización Bayesiana.

12. META-LEARNING

Se basa en la selección de hiperparámetros, investigando algoritmos para poder adaptarlos automáticamente. En este caso se ha estudiado "Learning to learn by gradient descent" [31], llegando a la conclusión de que existen mejores modificaciones de las estrategias de búsqueda que el descenso del gradiente, ya que esta regla de aprendizaje generaliza entrenando la red y el algoritmo conjuntamente, lo cual dificulta los entrenamientos.

El meta-learning considera que las redes neuronales son capaces de modificar sus propios pesos entrenando directamente un base-learner en lugar de un algoritmo de entrenamiento.

El objetivo es encontrar un buen optimizador para la función de minimización de coste $f(\theta)$.

Se pueden utilizar varios métodos para minimizar esta función pero el enfoque estándar para funciones diferenciales es un tipo de gradient descent que sigue el siguiente comportamiento:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$$

Con ello lo que se propone es reemplazar las reglas de actualización diseñadas a mano con una regla de actualización aprendida denominada optimizador 'g' que contiene una serie de parámetros ϕ .

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$

Para una red neuronal recurrente, la regla de actualización 'g' hace que se mantenga el estado y se actualice en función de sus iteraciones.

Una visión general del proceso sería la siguiente:

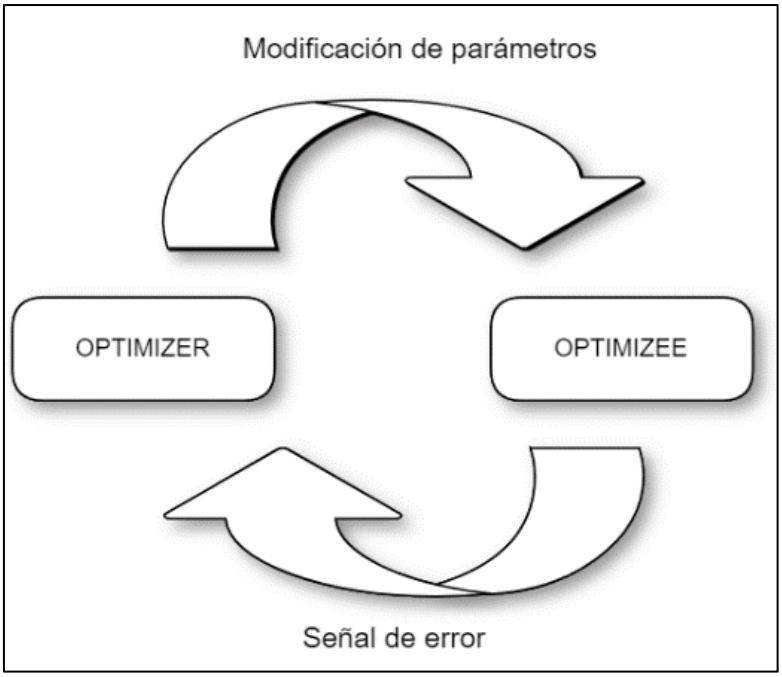


Figura 12.1: Proceso de actualización del optimizador.

Para parametrizar el optimizador se definen los parámetros finales del optimizee como $\theta^*(f,\phi)$ en función de los parámetros ϕ del optimizador.

Dada una distribución de funciones f se representa la pérdida esperada como:

$$\mathfrak{L}(\phi) = \mathbb{E}_f\left[f\left(\theta^*(f,\phi)\right)\right]$$

Se toman las modificaciones de los pasos 'g_t' como la salida de la red neuronal recurrente 'm', parametrizada por ϕ, cuyo estado denotaremos con h_t.

A continuación, mientras que la función anterior depende sólo del valor del parámetro final, para la formación del optimizador será conveniente tener un objetivo que dependa de toda la trayectoria de optimización, sobre un horizonte 'T'.

$$\mathfrak{L}(\phi) = \mathbb{E}_f\left[\sum_{t=1}^{T}\omega_t f\left(\theta_t\right)\right]$$

Dónde: $\theta_{t+1} = \theta_t + g_t$ $\quad \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m\left(\nabla_t, h_t, \phi\right)$ 'ω_t' representa los pesos arbitrarios.

En el caso específico en el que $\omega_t = 1[t = T]$, la primera ecuación de la función de pérdida y la segunda son iguales, por lo que el valor de los gradientes será 0 y sólo el paso de optimización final proporcionará información para entrenar al optimizador.

Para resolver este problema hace falta crear la restricción $\omega_t > 0$ en los puntos intermedios a lo largo de la trayectoria.

El cálculo del gradiente del optimizador permite entrenar el optimizador en trayectorias parciales.

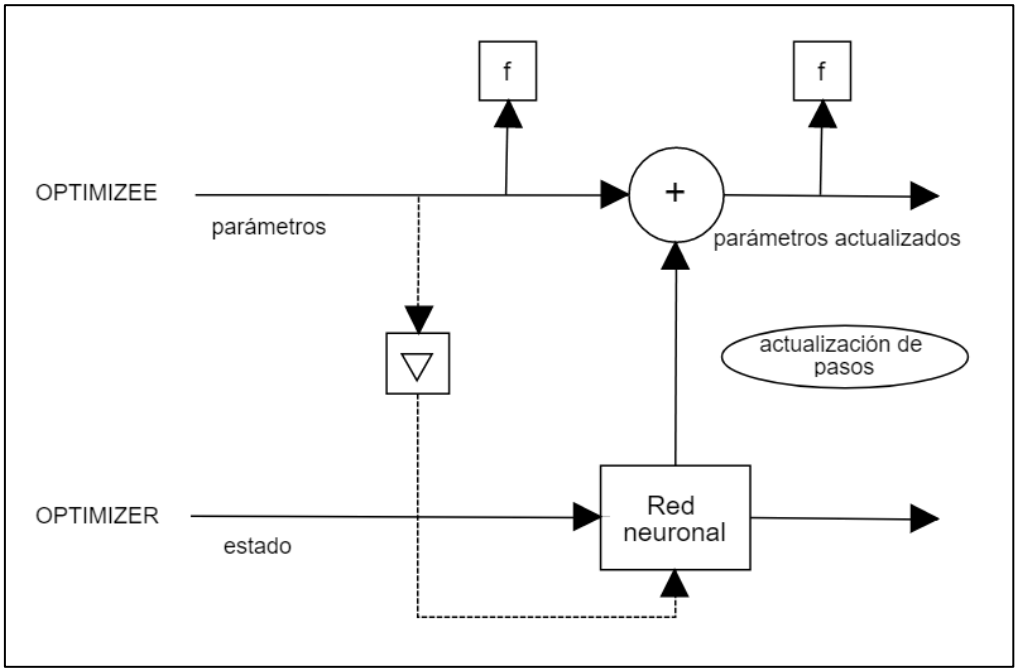


Figura 12.2: Cálculo del gradiente del optimizador.

Los gradientes que circulan por los bordes discontinuos de la imagen son eliminados para que los parámetros del optimizee no dependan de los del optimizador.

El valor estimado del gradiente $\frac{d\mathfrak{L}(\phi)}{d\phi}$ se puede calcular mediante el muestreo de una función aleatoria y aplicando posteriormente backpropagation, un algoritmo de aprendizaje supervisado que se usa para entrenar redes neuronales artificiales.

El problema de este sistema apareció cuando los experimentos realizados mostraron que esta optimización con una red neuronal fully connected requería un enorme número de parámetros y por tanto no era factible.

Para evitar esto se utiliza un learned optimizer similar al RMSprop o Adam, además de redes neuronales recurrentes LSTM. Esto permite utilizar una red pequeña que sólo mira una coordenada para definir el optimizador y compartir parámetros del mismo con otros del optimizee. Ya que de esta forma las salidas y entradas a la red neuronal LSTM tienen diferentes magnitudes, es necesario un preprocesado.

Para probar el rendimiento de los optimizadores entrenados se utiliza un modelo con tres capas convolucionales seguidas de una fully connected con activación Relu y una única capa LSTM.

El experimento se realizó con el entorno CIFAR-10, consistente en seleccionar imágenes de diez grupos distintos e ir agrupándolas.

El resultado del experimento resultó nefasto porque los pesos compartidos entre la fully connected y las capas convolucionales no se computaban correctamente, al encontrarse diferencias entre las arquitecturas de ambas capas.

Introduciendo una nueva capa LSTM, por un lado una capa LSTM modifica los parámetros de la fully connected, y por el otro, la segunda LSTM trabaja con los parámetros de las convolucionales.

Utilizando este tipo de optimizador el rendimiento mejora respecto a otros como ADAM, RMSprop o SGD, ya que la pérdida es menor.

Para evitar que los gradientes demasiado altos alteren la estabilidad del sistema es necesario generar celdas para permitir a las LSTM coordinarse entre sí. Por lo general se utilizan las GACs, un conjunto de celdas en cada capa LSTM para poder limitar el gradiente.

Añadiendo una memoria externa se consigue que se trabaje como dos procesos separados, la operación “read” y la “write”, que se comunican a través de una aproximación Hessiana, un método Quasi-Newtoniano que se utiliza para localizar la minimización de una función diferenciable guardada en la memoria.

La operación “read” se representa de la siguiente manera:

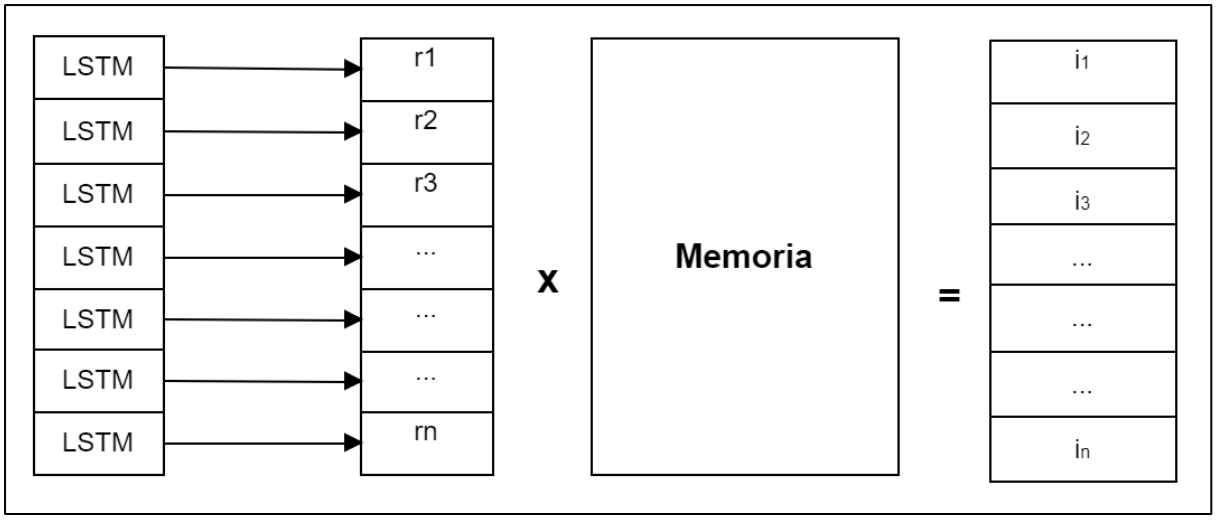


Figura 12.3: Operación "read".

$$g_t = read\left(M_t, \theta_t\right)$$

La actualización de pasos a la salida de la red neuronal LSTM entra en el vector "read" para combinarse con la memoria. Obteniendo una aproximación similar a la Hessiana M_t. El resultado entra en el controlador en el siguiente paso de tiempo:

$$\theta_{t+1} = \theta_t + g_t$$

La operación "write" trata de modificarla aproximación Hessiana M_t.

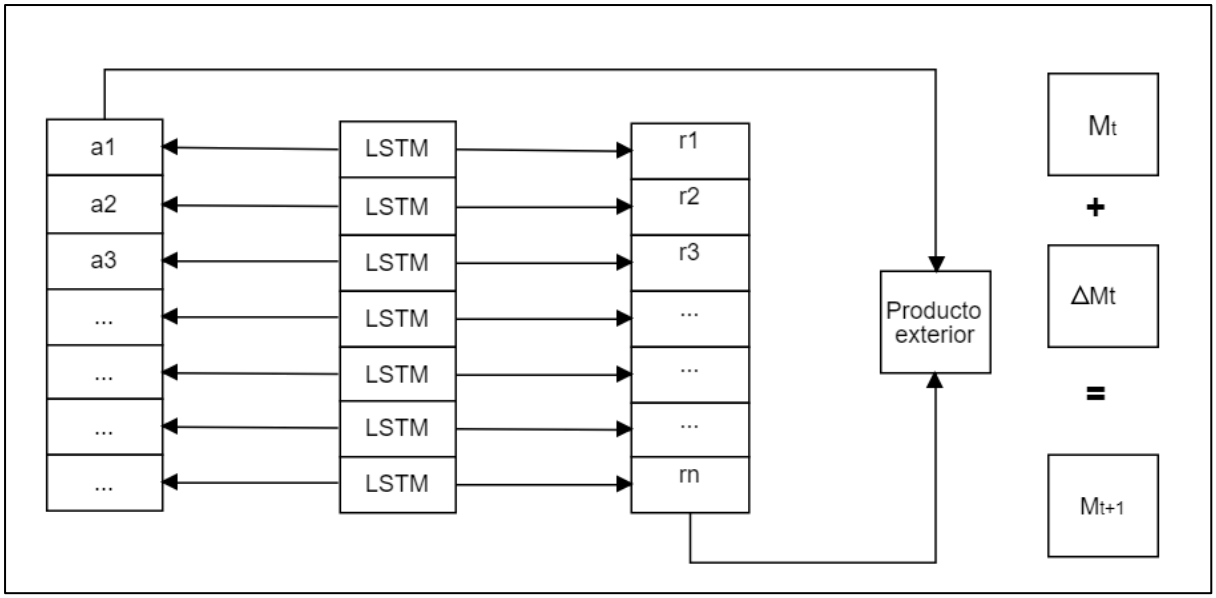


Figura 12.4: Operción "write"

Los vectores de salida "a" y "r" se utilizan para actualizar el estado de la memoria acumulando su producto exterior.

$$M_{t+1} = write\left(M_t, \theta_t, g_t\right)$$

13. MODELO DINÁMICO DEL DRON

13.1 NEWTON-EULER

El objetivo es crear un modelo matemático que se asemeje lo máximo posible al funcionamiento real del dron.

La representación dinámica del dron debe resolverse antes de utilizar las estrategias de control, como los métodos de estabilización y control de la trayectoria.

El reto en el control de un dron viene determinado por los seis grados de libertad que tiene, ya que sólo hay cuatro entradas de control. [32]

Tres de los grados de libertad son los ángulos de navegación (pitch, yaw and roll) y los otros se refieren al movimiento en un espacio 3D (x, y, z).

En este caso el dron se controla a partir del empuje total producido por los rotores y los momentos angulares del roll, pitch y yaw.

El modelo que se va a desarrollar se basa en el método “Newton-Euler”, con el que se calculan tanto los pares y fuerzas de cada rotor, como las ecuaciones de velocidad y aceleración, lineal y angular.

Figura 13.1: Quadrotor.

Para el correcto desarrollo del modelo se debe tener en cuenta los momentos, velocidades angulares y fuerzas a los que estará expuesto el dron.

La posición se define como:

$$\xi^i = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

El superíndice 'i' se refiere a que pertenece al sistema inercial, del que se hablará posteriormente.

La posición angular se obtiene a partir de los ángulos de navegación:

$$\eta^i = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

Figura 13.2: Orientación pitch.

Siendo Pitch el grado de inclinación hacia delante y hacia atrás.

Se representa con el ángulo θ que especifica la rotación del dron sobre el eje y.

Figura 13.3: Orientación roll.

Roll es el ángulo ϕ que determina la rotación sobre el eje x. Es la inclinación de los lados del dron.

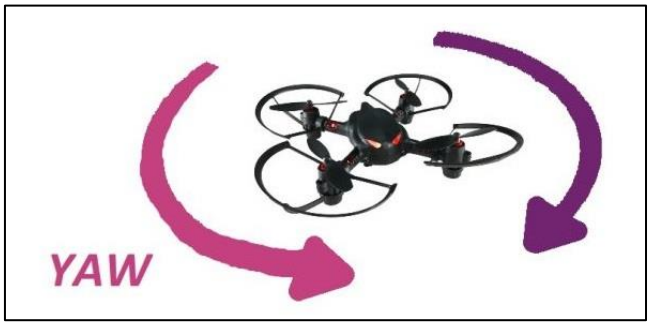


Figura 13.4: Orientación yaw.

Yaw es el ángulo ψ define la rotación respecto el eje z, es decir, la rotación izquierda y derecha del dron.

Tanto la posición lineal como la angular están definidas en el siguiente sistema inercial:

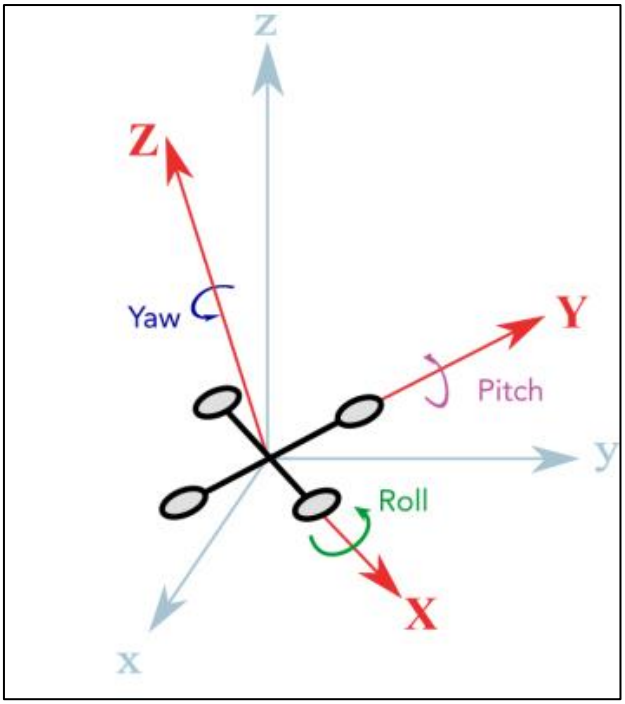


Figura 13.5: Sistema inercial.

Una vez obtenida la posición del dron se calcula su velocidad lineal V_L y su velocidad angular ν en su centro de masa:

$$V_L^{CM} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \qquad \nu^{CM} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

El superíndice 'CM' hace referencia al centro de masa del dron.

La orientación del dron desde el centro de masa con respecto al sistema inercial se define a partir de la matriz de rotación, la cual se va a utilizar posteriormente en las ecuaciones de Newton-Euler para determinar el modelo dinámico del dron.

El cálculo de la matriz de rotación total se realiza a partir del producto de tres matrices de rotación básicas, calculadas de la siguiente manera:

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = R \cdot \begin{pmatrix} x_{CM} \\ y_{CM} \\ z_{CM} \end{pmatrix}$$

La primera matriz de rotación básica es la rotación Yaw, sobre el eje z. Al hacer el giro respecto el sistema de referencia inercial, se obtiene un nuevo eje de coordenadas que hemos representado con el subíndice ‘1’.

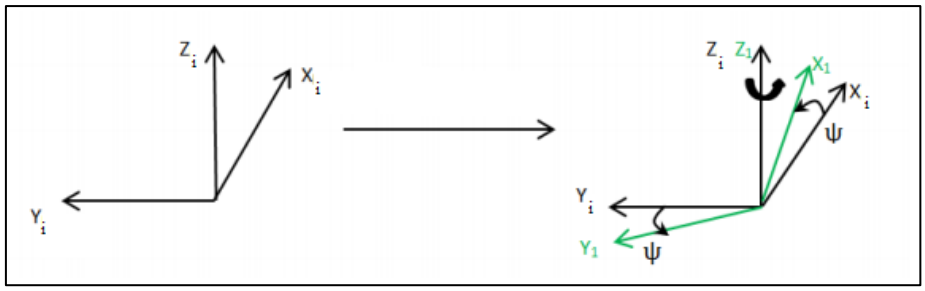


Figura 13.6: Giro sobre eje “z”.

$$R_i^1 = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La matriz de rotación Pitch se halla a partir del sistema ‘1’ calculado anteriormente.

Al hacer el giro del Pitch se adquiere un nuevo sistema de coordenadas nombrado con el subíndice ‘2’.

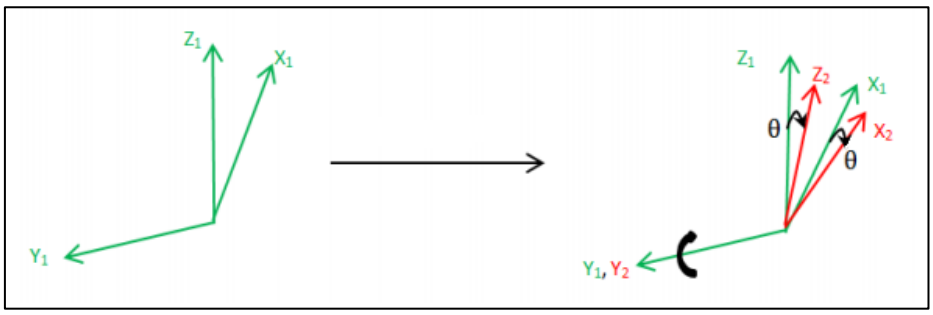


Figura 13.7: Giro del pitch.

$$R_1^2 = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

Con la matriz de rotación básica Roll, se calcula el último giro que realiza el dron, por lo que el nuevo eje de coordenadas es el centro de masa del mismo.

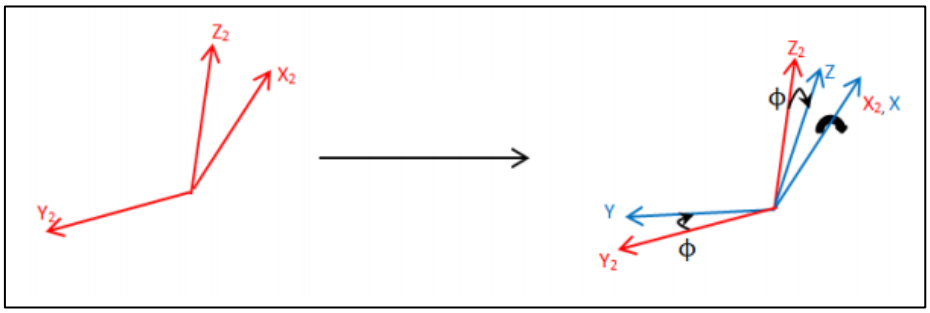


Figura 13.8: Giro del roll.

$$R_2^{CM} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

La matriz de rotación total necesaria para obtener la posición y la velocidad respecto del sistema inercial es:

$$R = R_2^{CM} \cdot R_1^2 \cdot R_i^1$$

$$R = \begin{bmatrix} \cos(\psi)\cdot\cos(\theta) & \cos(\psi)\cdot\sin(\theta)\cdot\sin(\phi) - \sin(\psi)\cdot\cos(\phi) & \cos(\psi)\cdot\sin(\theta)\cdot\cos(\phi) + \sin(\psi)\cdot\sin(\phi) \\ \sin(\psi)\cdot\cos(\theta) & \sin(\psi)\cdot\sin(\theta)\cdot\sin(\phi) + \cos(\psi)\cdot\cos(\phi) & \sin(\psi)\cdot\sin(\theta)\cdot\cos(\phi) - \cos(\psi)\cdot\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\cdot\sin(\phi) & \cos(\theta)\cdot\cos(\phi) \end{bmatrix}$$

La matriz de rotación es ortogonal, es decir, su matriz inversa coincide con su matriz traspuesta, $R^{-1} = R^T$.

Posteriormente se calcula la matriz de transformación también llamada matriz de translación, necesaria para hallar las velocidades angulares desde el sistema inercial hasta el centro de masas del vehículo. Se representa como W_η.

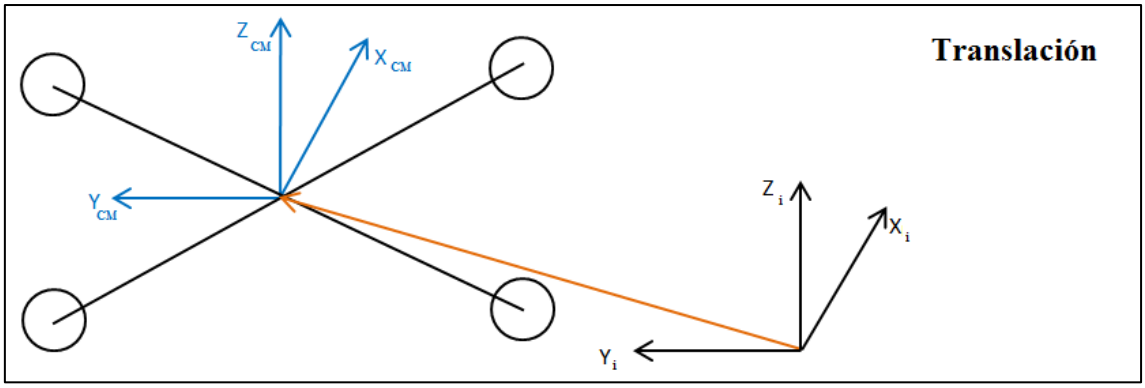


Figura 13.9: Traslación.

Ya que se pretende conseguir la matriz de transformación desde el centro de masa hasta el sistema inercial, la calculamos como W_η^{-1}.

Para calcular la matriz W_η es necesario tener los ángulos de navegación respecto del eje de coordenadas, del centro de masa, debido a que en el cálculo de las matrices básicas de rotación los ejes se han ido modificando con cada giro.

De esta manera, el ángulo Roll no hay que modificarlo puesto que ha sido el último giro realizado y ya está situado sobre el eje del centro de masa.

Al ángulo Pitch le falta la última rotación, por tanto se multiplica por la matriz de rotación R_2^{CM}.

Y el ángulo Yaw, al ser el primero calculado, se debe multiplicar por las matrices de rotación básicas posteriores.

$$\nu^{CM} = W_\eta \cdot \dot{\eta}^i \rightarrow \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_2^{CM} \cdot \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_2^{CM} \cdot R_1^2 \cdot \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

$$\nu^{CM} = W_\eta \cdot \dot{\eta}^i \rightarrow \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin(\phi) \\ 0 & \cos(\phi) & \cos(\theta)\cdot\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cdot\cos(\phi) \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

$$\dot{\eta}^i = W_\eta^{-1} \cdot \nu^{CM} \rightarrow \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\cdot\tan(\theta) & \cos(\phi)\cdot\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

La matriz de transformación es:

$$W_\eta^{-1} = \begin{bmatrix} 1 & \sin(\phi)\cdot\tan(\theta) & \cos(\phi)\cdot\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{bmatrix}$$

Asumiendo que el dron tiene una estructura simétrica con respecto al ‘eje x’ y al ‘eje y’, la matriz de inercia es una matriz diagonal I en la cual $I_{xx} = I_{yy}$.

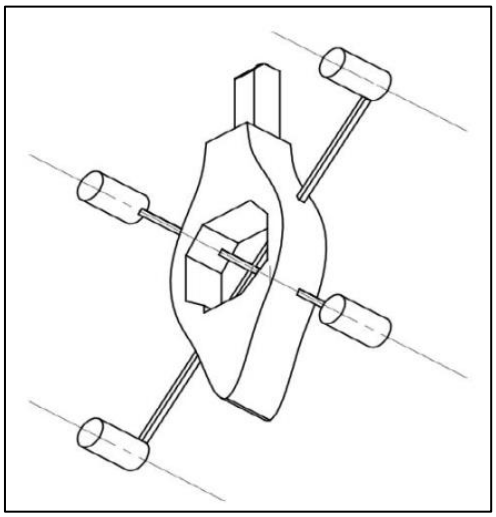


Figura 13.10: Simetría del dron.

Las ecuaciones dinámicas para un cuerpo rígido expuesto a fuerzas externas que se aplican al centro de masa se calculan mediante Newton-Euler de la siguiente forma:

$$\begin{bmatrix} m\cdot I & 0 \\ 0 & J \end{bmatrix} \cdot \begin{bmatrix} \dot{V} \\ \dot{\nu} \end{bmatrix} + \begin{bmatrix} \nu \times m \cdot V \\ \nu \times J \cdot \nu \end{bmatrix} = \begin{bmatrix} F + F_d \\ \tau + \tau_d \end{bmatrix}$$

Donde m es la masa total del dron, I es la matriz identidad, J es la matriz de inercia que se pretende calcular. F es el vector de fuerzas de traslación aplicadas al modelo y F_d fuerzas producidas por las perturbaciones que afectan al sistema.

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

La velocidad angular del rotor i que se define como ϖ_i, crea una fuerza f_i en la dirección perpendicular a los ejes del rotor.

La velocidad angular y la aceleración del rotor también crean esfuerzos de torsión (torques o par motor) τ_{Mi} alrededor de los ejes del rotor.

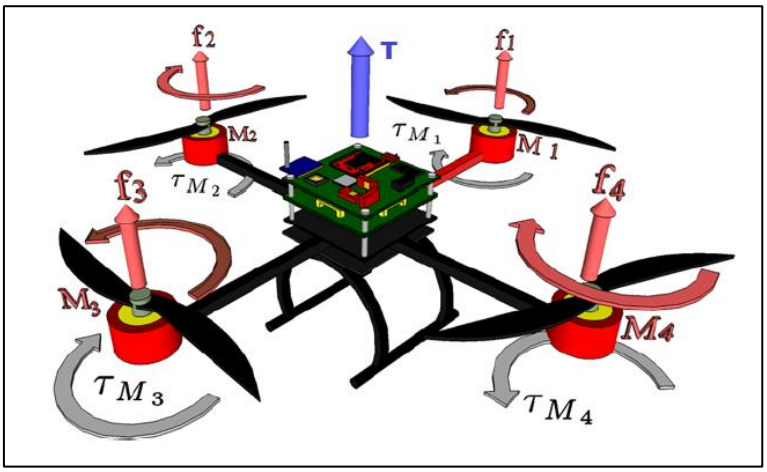


Figura 13.11: Fuerzas, momentos y velocidades angulares.

$$f_i = k \cdot \varpi_i^{\,2}$$

$$\tau_{Mi} = b \cdot \varpi_i^{\,2} + I_M \cdot \dot{\varpi}_i$$

El subíndice 'i' denota el número de motor, de 1 a 4.

Siendo k (lift) la constante que permite al dron ascender, b (drag) la constante de arrastre que se produce por la resistencia del aire e I_M el momento de inercia del rotor.

Normalmente el efecto de $\dot{w}_i$ es considerado muy pequeño y por tanto se suele omitir.

Combinando las fuerzas obtenidas de los cuatro motores, y al ser simétrico, se crea un empuje T en el centro de masa del dron en dirección del eje z, como se representa en la figura 13.11.

$$T = \sum_{i=1}^{4} f_i = k \cdot \sum_{i=1}^{4} \varpi_i^{\,2}$$

$$T^{CM} = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

Siendo l es la distancia entre el centro del rotor y el centro de masa del dron. τ^{CM} es el momento creado por el rotor correspondiente.

$$\tau^{CM} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} l \cdot k \cdot (-\varpi_2^2 + \varpi_4^2) \\ l \cdot k \cdot (-\varpi_1^2 + \varpi_3^2) \\ \sum_{i=1}^{4} \tau_{Mi} \end{bmatrix}$$

Como se puede ver, los momentos generados en el pitch y roll (θ y ϕ) se deben a la diferencia de velocidades angulares entre motores del mismo eje, es decir, en la figura anterior, el momento del roll se obtendría a partir de los motores 2 y 4, y el del pitch a partir de los motores 1 y 3.

Ya que el dron es un cuerpo rígido, las ecuaciones de Newton-Euler pueden utilizarse para describir su dinámica, partiendo del equilibrio de fuerzas y pares.

$$\sum F = m \cdot a$$

$$\sum T = I \cdot w + w \times (I \cdot w)$$

En el centro de masa del dron, la fuerza gravitacional que se requiere para acelerar esa masa, ($m \cdot \dot{V}^{CM}$), y la fuerza centrífuga, $\nu \times (\mathrm{m} \cdot \mathrm{V}^{CM})$, son iguales a la suma de la gravedad $R^T \cdot G$ y el empuje total de los motores T^{CM}.

$$m \cdot \dot{V}^{CM} + \nu \times (\mathrm{m} \cdot \mathrm{V}^{CM}) = \mathrm{R}^T \cdot G + T^{CM}$$

En un sistema no inercial no se cumplen las leyes de Newton para las fuerzas reales que aparecen, por tanto para poder aplicar estas leyes es necesario introducir las llamadas fuerzas ficticias, en este caso la fuerza centrífuga.

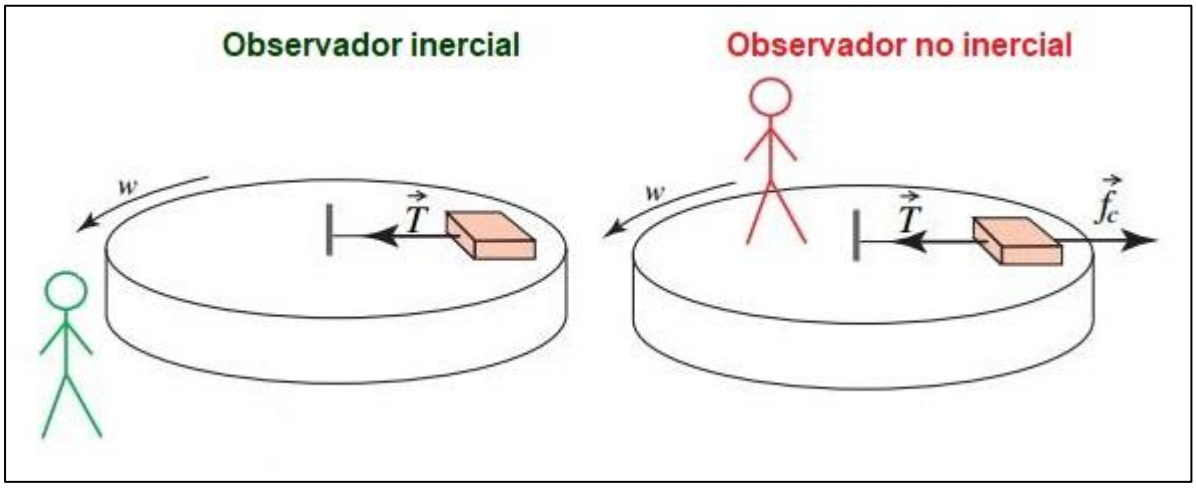


Figura 13.12: Sistema inercial y no inercial.

En el sistema inercial las fuerzas ficticias son nulas.

Solo la fuerza gravitacional, la magnitud y la dirección contribuyen en la aceleración del dron.

Aplicando el equilibrio de fuerzas, obtenemos que el producto de la masa del dron y la aceleración sea igual a la suma de la gravedad con el producto de la matriz de rotación y el empuje total de los motores en el centro de masas.

$$m \cdot \ddot{\xi} = G + R \cdot T^{CM}$$

A partir de la anterior ecuación y colocando los términos, podemos calcular las aceleraciones producidas en el sistema inercial:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \cdot \begin{bmatrix} \cos(\psi)\cdot\sin(\theta)\cdot\cos(\phi) + \sin(\psi)\cdot\sin(\phi) \\ sin(\psi)\cdot\sin(\theta)\cdot\cos(\phi) - \cos(\psi)\cdot\sin(\phi) \\ \cos(\theta)\cdot\cos(\phi) \end{bmatrix} \qquad (1)$$

En el centro de masa, la aceleración angular producida por la inercia ($I \cdot \dot{\nu}^{CM}$), las fuerzas centrípetas ($\nu^{CM} \times (\mathrm{I} \cdot \nu^{CM})$) y las fuerzas giroscópicas (Γ) son iguales al torque o par motor externo (τ).

$$I \cdot \dot{\nu}^{CM} + \nu^{CM} \times (\mathrm{I} \cdot \nu^{CM}) + \Gamma = \tau$$

A partir de esta ecuación podemos calcular la aceleración angular del centro de masa del dron:

$$\dot{\nu}^{CM} = I^{-1} \cdot \left(-\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx} \cdot p \\ I_{yy} \cdot q \\ I_{zz} \cdot r \end{bmatrix} - I_r \cdot \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot w_\Gamma + \tau \right)$$

Ordenando los términos obtenemos la siguiente ecuación:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{(\mathrm{I}_{yy} - \mathrm{I}_{zz}) \cdot \mathrm{q} \cdot \mathrm{r}}{\mathrm{I}_{xx}} \\ \frac{(\mathrm{I}_{zz} - \mathrm{I}_{xx}) \cdot p \cdot \mathrm{r}}{\mathrm{I}_{yy}} \\ \frac{(\mathrm{I}_{xx} - \mathrm{I}_{yy}) \cdot p \cdot q}{\mathrm{I}_{zz}} \end{bmatrix} - I_r \cdot \begin{bmatrix} \frac{q}{\mathrm{I}_{xx}} \\ \frac{-p}{\mathrm{I}_{yy}} \\ 0 \end{bmatrix} \cdot w_\Gamma + \begin{bmatrix} \frac{\tau_\phi}{\mathrm{I}_{xx}} \\ \frac{\tau_\theta}{\mathrm{I}_{yy}} \\ \frac{\tau_\psi}{\mathrm{I}_{zz}} \end{bmatrix}$$

Donde $w_\Gamma = w_1 - w_2 + w_3 - w_4$.

La aceleración angular en el sistema inercial se obtiene a partir de la aceleración del centro de masa y la matriz de transformación W_η^{-1} .

$$\ddot{\eta} = \frac{d}{dt}(W_\eta^{-1} \cdot \nu) = \frac{d}{dt}(W_\eta^{-1}) \cdot \nu + W_\eta^{-1} \cdot \dot{\nu} =$$

$$= \begin{bmatrix} 0 & \dot{\phi} \cdot \cos(\phi) \cdot \tan(\theta) + \dot{\theta} \cdot \sin(\phi) / \cos^2(\theta) & -\dot{\phi} \cdot \sin(\phi) \cdot \cos(\theta) + \dot{\theta} \cdot \cos(\phi) / \cos^2(\theta) \\ 0 & -\dot{\phi} \cdot \sin(\phi) & -\dot{\phi} \cdot \cos(\phi) \\ 0 & (\dot{\phi} \cdot \cos(\phi) / \cos(\theta)) + \dot{\phi} \cdot \sin(\phi) \cdot \tan(\theta) / \cos(\theta) & (-\dot{\phi} \cdot \sin(\phi) / \cos(\theta)) + \dot{\theta} \cdot \cos(\phi) \cdot \tan(\theta) / \cos(\theta) \end{bmatrix} \cdot \nu + W_\eta^{-1} \cdot \dot{\nu}$$

El modelo matemático que hemos creado con las anteriores ecuaciones es una simplificación de interacciones dinámicas complejas.

Para generar un modelo más realista del dron se incluye la fuerza de arrastre generada por la resistencia del aire.

Por tanto se lo debemos añadir a la ecuación (1).

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \cdot \begin{bmatrix} \cos(\psi) \cdot \sin(\theta) \cdot \cos(\phi) + \sin(\psi) \cdot \sin(\phi) \\ sin(\psi) \cdot \sin(\theta) \cdot \cos(\phi) - \cos(\psi) \cdot \sin(\phi) \\ \cos(\theta) \cdot \cos(\phi) \end{bmatrix} - \frac{1}{m} \cdot \begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix} \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

Siendo $A_x, A_y \, y \, A_z$ los coeficientes de las fuerzas de arrastre para las velocidades en la dirección correcta del sistema de inercia.

Los efectos aerodinámicos que se producen son difíciles de modelar y algunos sólo tienen repercusión cuando el dron se encuentra a altas velocidades. Estos efectos se excluyen del modelo con el fin de simplificarlo.

Los cuatro motores eléctricos que tiene el dron no están modelados.

Los valores utilizados para la simulación son los siguientes:

Parámetro	Valor	Unidad
g	9.81	m/s^2
m	0.468	Kg
l	0.225	m
k	$2.98 \cdot 10^{-6}$	$N \cdot s^2$
b	$1.14 \cdot 10^{-7}$	$N \cdot m \cdot s^2$
I_M	$3.357 \cdot 10^{-5}$	$Kg \cdot m^2$

Parámetro	Valor	Unidad
I_{xx}	$4.856 \cdot 10^{-3}$	$Kg \cdot m^2$
I_{yy}	$4.856 \cdot 10^{-3}$	$Kg \cdot m^2$
I_{zz}	$8.801 \cdot 10^{-3}$	$Kg \cdot m^2$
A_x	0.25	Kg/s
A_y	0.25	Kg/s
A_z	0.25	Kg/s

Se representa el modelo del dron dibujando un diagrama con sus entradas y salidas:

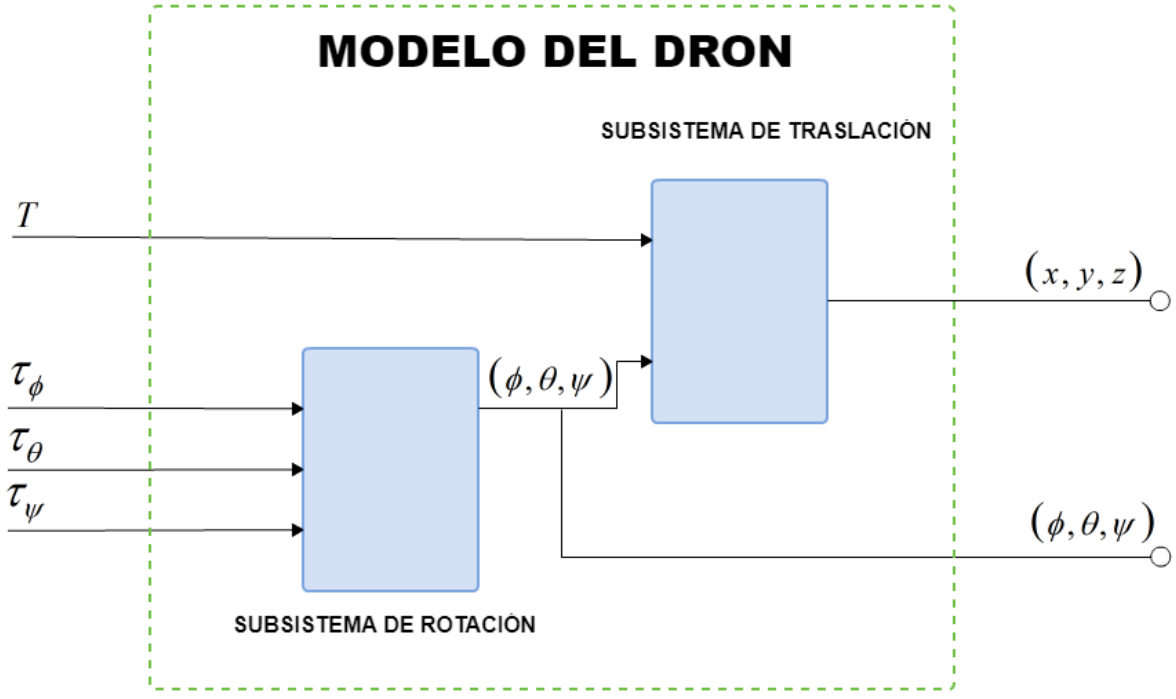


Figura 13.13: Modelo dinámico del dron.

En el diseño del controlador se han redactado las ecuaciones como variables de estado, las cuales se dividen en dos subsistemas, el de rotación angular y el de traslación lineal.

El subsistema de rotación cuyas entradas son los pares que permiten el giro del dron, se encarga de su orientación mediante el control de los ángulos. Se pretende con ello estabilizar el dron en la posición deseada.

Para poder controlar tanto el movimiento en el plano xy como la altura, se utiliza el subsistema de traslación, que tiene como entradas los ángulos de navegación y el empuje total formado por el sumatorio de las fuerzas producidas en cada rotor.

13.2 EULER-LAGRANGE

La función de Lagrange $\mathfrak{L}$ es la suma de las energías cinéticas de translación E_{trans} y de rotación E_{rot} menos la energía potencial E_{pot}.

$$\begin{aligned}\mathfrak{L}(q,\dot{q}) = \quad E_{trans} \quad + \quad E_{rot} \quad - E_{pot} = \\ = (m/2)\dot{\xi}^{\mathrm{T}}\dot{\xi} + (1/2)\nu^{\mathrm{T}} I\nu - mgz.\end{aligned}$$

Donde z representa la altitud del dron.

Se aplica las ecuaciones de Euler-Lagrange para las fuerzas externas y los pares:

$$\begin{bmatrix} f \\ \tau \end{bmatrix} = \frac{d}{dt}\left(\frac{\partial \mathfrak{L}}{\partial \dot{q}}\right) - \frac{\partial \mathfrak{L}}{\partial q}.$$

Los componentes lineales y angulares no dependen el uno del otro ya que el lagrangiano no tiene términos cruzados entre la energía cinética de rotación la de translación, por lo tanto se pueden estudiar por separado.

La fuerza externa lineal es el empuje total de los rotores.

El modelo dinámico se obtiene a partir de las ecuaciones de Euler-Lagrange con fuerzas y momentos generalizados.

Las ecuaciones lineales de Euler-Lagrange son:

$$f = R\mathrm{T}_B = m\ddot{\xi} + mg\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Como se puede apreciar, la forma de calcular la fuerza externa lineal o empuje total es la misma que en el método de Newton-Euler.

La matriz jacobiana de ν a $\dot{\eta}$ se define de la siguiente manera:

$$J(\eta) = J = W_{\eta}^{\mathrm{T}} I W_{\eta},$$
$$= \begin{bmatrix} I_{xx} & 0 & -I_{xx}\sin\theta \\ 0 & I_{yy}\cos^2\phi + I_{zz}\sin^2\phi & \left(I_{yy} - I_{zz}\right)\cos\phi\sin\phi\cos\theta \\ -I_{xx}\sin\theta & \left(I_{yy} - I_{zz}\right)\cos\phi\sin\phi\cos\theta & I_{xx}\sin^2\phi\cos^2\theta + I_{zz}\cos^2\phi\cos^2\theta \end{bmatrix}$$

Por lo tanto, la energía cinética de rotación E_{rot} se puede expresar como:

$$E_{rot} = (1/2)\nu^{\mathrm{T}} I \nu = (1/2)\ddot{\eta}^{\mathrm{T}} J \ddot{\eta}$$

La fuerza angular externa se corresponde con los pares de los rotores.

$$\tau = \tau_B = J\ddot{\eta} + \frac{d}{dt}(J)\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial\eta}\left(\dot{\eta}^{\mathrm{T}} J \dot{\eta}\right) = J\ddot{\eta} + C(\eta,\dot{\eta})\dot{\eta}$$

Siendo la matriz $C(\eta,\dot{\eta})$ el término de Coriolis, el cual contiene las fuerzas giroscópicas y centrípetas.

La matriz $C(\eta,\dot{\eta})$ tiene la siguiente forma:

$$C(\eta,\dot{\eta}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix},$$

Siendo:

$$C_{11} = 0$$

$$C_{12} = \left(I_{yy} - I_{zz}\right)\left(\dot{\theta}\cos\phi\sin\phi + \dot{\psi}\sin^2\phi\cos\theta\right) + \left(I_{zz} - I_{yy}\right)\dot{\psi}\cos^2\phi\cos\theta - I_{xx}\dot{\psi}\cos\theta$$

$$C_{13} = \left(I_{zz} - I_{yy}\right)\dot{\psi}\cos\phi\sin\phi\cos^2\theta$$

$$C_{21} = \left(I_{zz} - I_{yy}\right)\left(\dot{\theta}\cos\phi\sin\phi + \dot{\psi}\sin\phi\cos\theta\right) + \left(I_{yy} - I_{zz}\right)\dot{\psi}\cos^2\phi\cos\theta + I_{xx}\dot{\psi}\cos\theta$$

$$C_{22} = \left(I_{zz} - I_{yy}\right)\dot{\phi}\cos\phi\sin\phi$$

$$C_{23} = -I_{xx}\dot{\psi}\sin\theta\cos\theta + I_{yy}\dot{\psi}\sin^2\phi\, sen\theta\cos\theta + I_{zz}\dot{\psi}\cos^2\phi\sin\theta\cos\theta$$

$$C_{31} = \left(I_{yy} - I_{zz}\right)\dot{\psi}\cos^2\theta\sin\phi\cos\phi - I_{xx}\dot{\theta}\cos\theta$$

$$C_{32} = \left(I_{zz} - I_{yy}\right)\left(\dot{\theta}\cos\phi\sin\phi\sin\theta + \dot{\phi}\sin^2\phi\cos\theta\right) + \left(I_{yy} - I_{zz}\right)\dot{\phi}\cos^2\phi\cos\theta +$$
$$+ I_{xx}\dot{\psi}\sin\theta\cos\theta - I_{yy}\dot{\psi}\sin^2\phi\sin\theta\cos\theta - I_{zz}\dot{\psi}\cos^2\phi\sin\theta\cos\theta$$

$$C_{33} = \left(I_{yy} - I_{zz}\right)\dot{\phi}\cos\phi\sin\phi\cos^2\theta - I_{yy}\dot{\theta}\sin^2\phi\cos\theta\sin\theta -$$
$$- I_{zz}\dot{\theta}\cos^2\phi\cos\theta\sin\theta + I_{xx}\dot{\theta}\cos\theta\sin\theta$$

A partir de las ecuaciones diferenciales para las aceleraciones angulares e igualándolas a los pares de los rotores calculados, se obtiene:

$$\ddot{\eta} = J^{-1}\left(\tau_B - C\left(\eta,\dot{\eta}\right)\dot{\eta}\right)$$

13.3 CONCLUSIONES

El método de Newton-Euler relaciona fuerzas, momentos y pares. Las ecuaciones resultantes incluyen fuerzas de restricción que deben ser eliminadas para conseguir las ecuaciones dinámicas de forma cerrada.

Las ecuaciones no se representan en términos de variables independientes, ya que sería necesario el uso de operaciones aritméticas para derivar las ecuaciones dinámicas.

La alternativa encontrada a este método es la formulación de Euler-Lagrange explicada anteriormente, que describe el comportamiento de un sistema dinámico en función del trabajo y la energía almacenada en el sistema, es decir, en el balance de energía.

Para el desarrollo del proyecto se ha seleccionado el método de Newton-Euler. Obteniéndose con ambos algoritmos unos resultados similares, la dificultad matemática y comprensiva del desarrollo de Euler-Lagrange es muy superior al método elegido.

El modelo del dron ha sido programado en Python y se ha comprobado el correcto funcionamiento del mismo. Se ha implementado incluyendo los principales efectos físicos que actúan sobre él, como efectos aerodinámicos, giroscópicos y de la gravedad.

También se ha tenido en cuenta la resistencia que impone el aire para conseguir que el dron ascienda.

Durante el desarrollo del modelo surgieron varios problemas con el tipo de dato que se manejaba, ya que los resultados se guardaban como vectores de 3x1, debido a que se trata de un modelo tridimensional y se necesitan los datos de cada una de esas dimensiones, en distintas listas. Al combinar el uso de listas con las tuplas, creadas con el fin de proteger los datos que no debían ser modificados hasta que se propusiese una nueva orden, por ejemplo en el momento de elegir una acción para impulsar el dron, generó fallos en el cómputo del programa tras un mal uso de las mismas.

Poniendo todos los momentos y empujes con valor inicial cero, el dron tiende a caer ya que la fuerza de la gravedad lo empuja hacia abajo. La posición inicial del dron es (0, 0, 1).

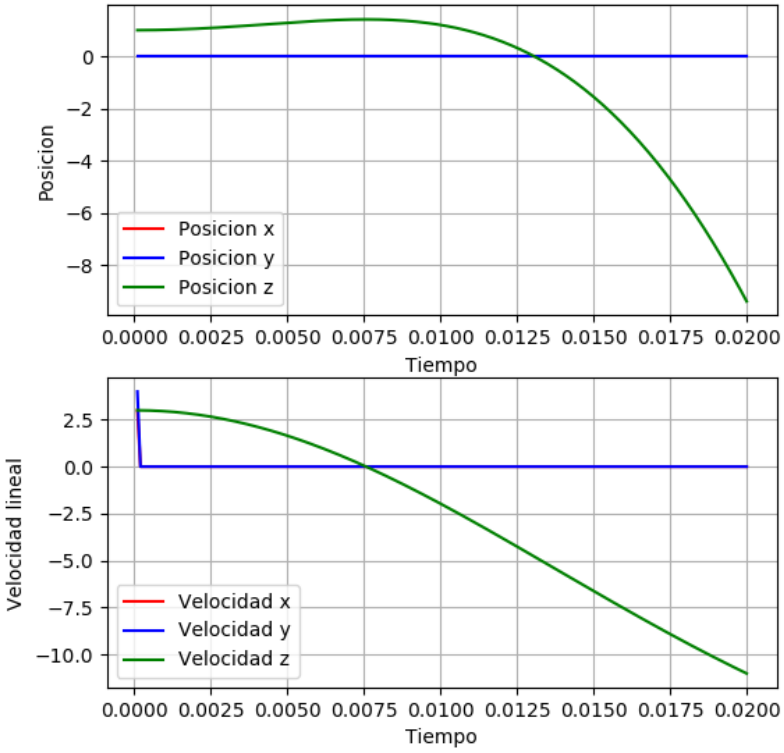


Figura 13.14: Situación dinámica lineal del dron.

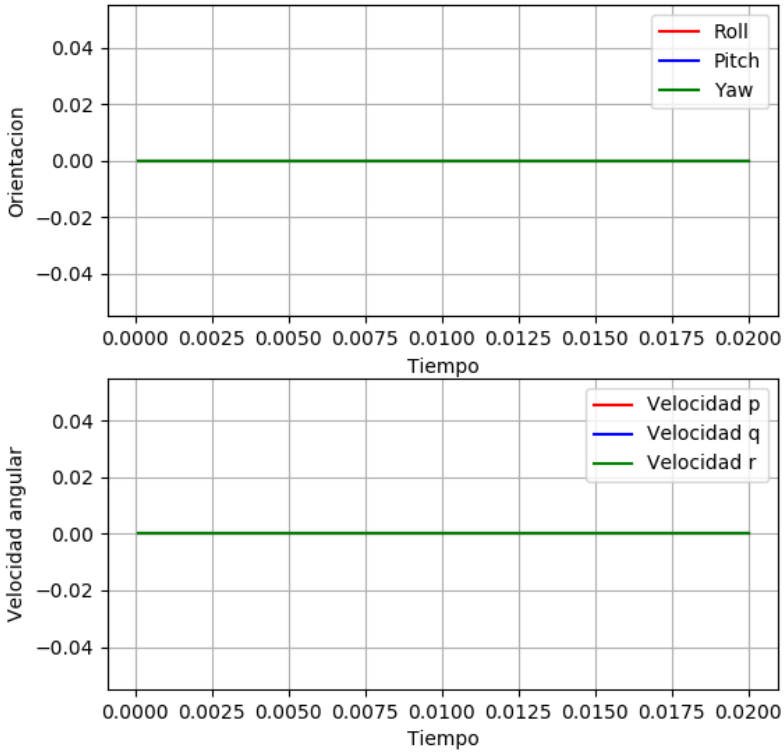


Figura 13.14: Situación dinámica lineal del dron.

Tras incrementar el empuje y el momento del roll se obtiene el siguiente resultado:

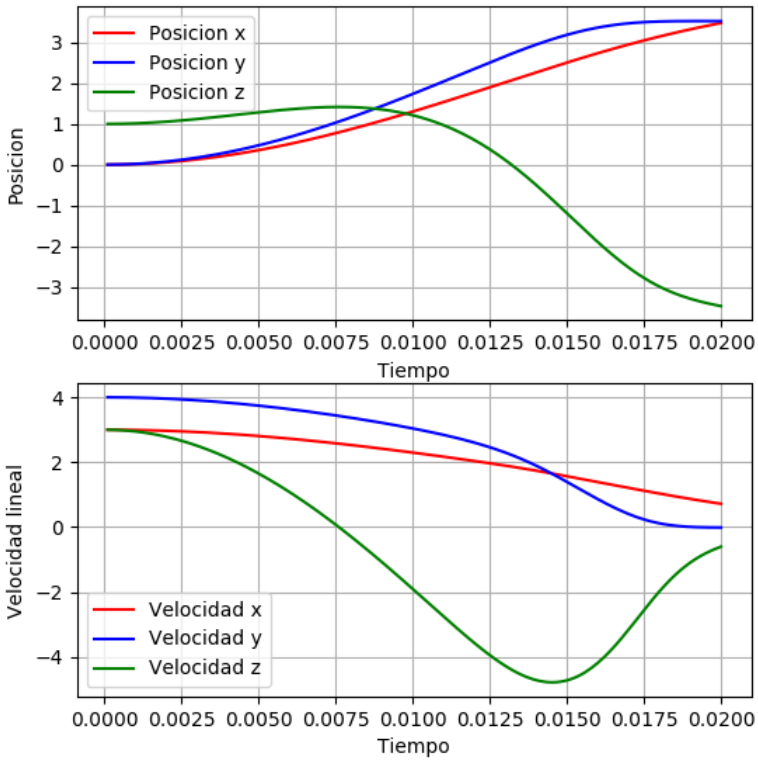


Figura 13.14: Situación dinámica lineal del dron.

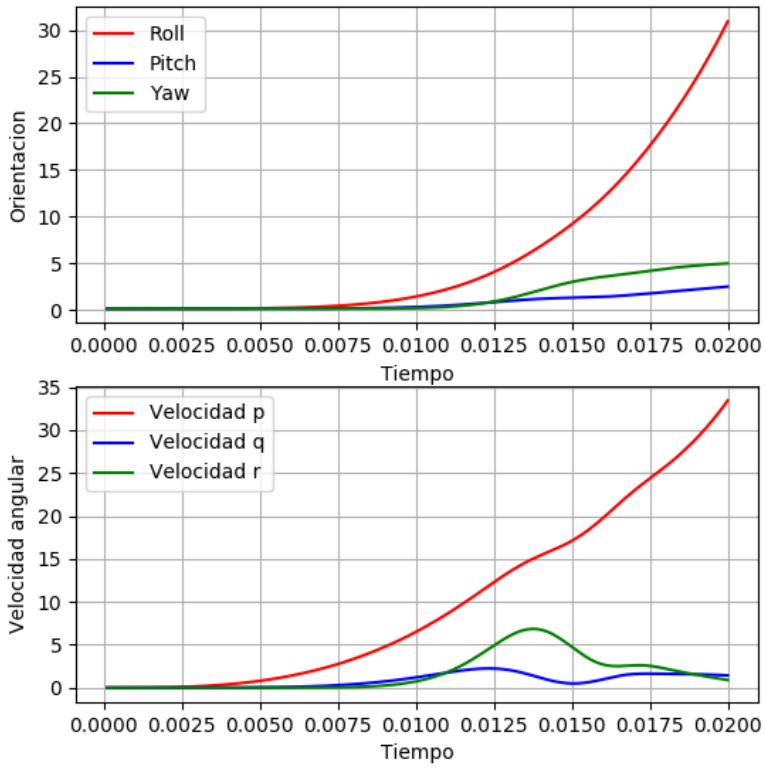


Figura 13.14: Situación dinámica del dron.

Se puede observar que todavía no se le ha dado al dron el empuje suficiente para que consiga contrarrestar la gravedad por lo que la posición z cae hasta un valor de -3 metros. La velocidad lineal cae en la primera zona, pero al aumentar el empuje en cada paso llega a la zona en la que el dron empieza a compensar la fuerza de la gravedad por lo que cada vez es menos negativa.

Para resolver este problema se ha fijado un valor inicial del empuje de 4.2 $N \cdot s$ con el fin de llegar a la zona en el que se estabiliza lo antes posible.

Añadiendo momento del pitch y empuje se consigue lo siguiente:

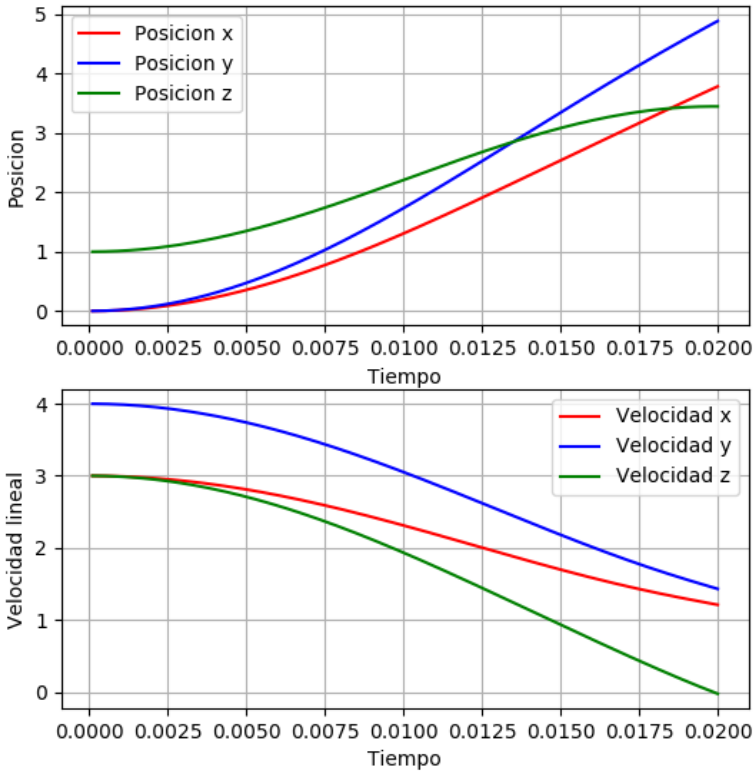


Figura 13.15: Zona lineal a partir de nuevas entradas.

Ahora el dron ha conseguido coger altura y llegar a los 3 metros y medio. La orientación cambia de modo que el dron se inclina hacia delante, rota sobre el eje z en sentido antihorario y lateralmente hacia la izquierda. La velocidad lineal ahora es positiva y al haber modificado el momento del pitch, tanto su orientación como su velocidad angular también aumentan.

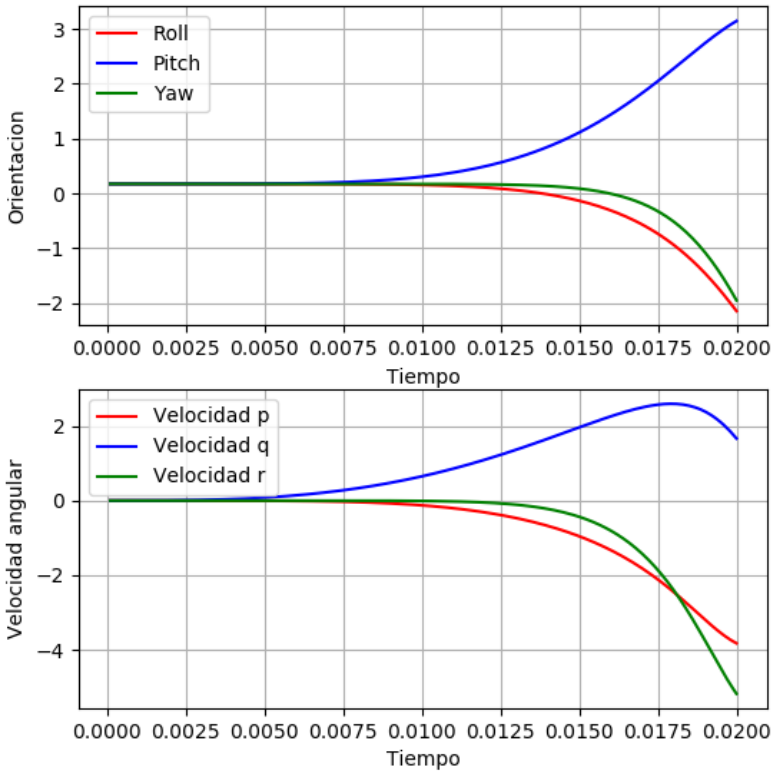


Figura 13.15: Zona angular a partir de nuevas entradas.

Si se utiliza el mismo empuje que en los casos anteriores y se realiza un decremento del momento de yaw, se produce una fuerte rotación antihoraria, afectando a la orientación, y la velocidad angular decae fuertemente. Se llega a la conclusión de que tanto las modificaciones en el roll como en el yaw realizan cambios más fuertes que el pitch.

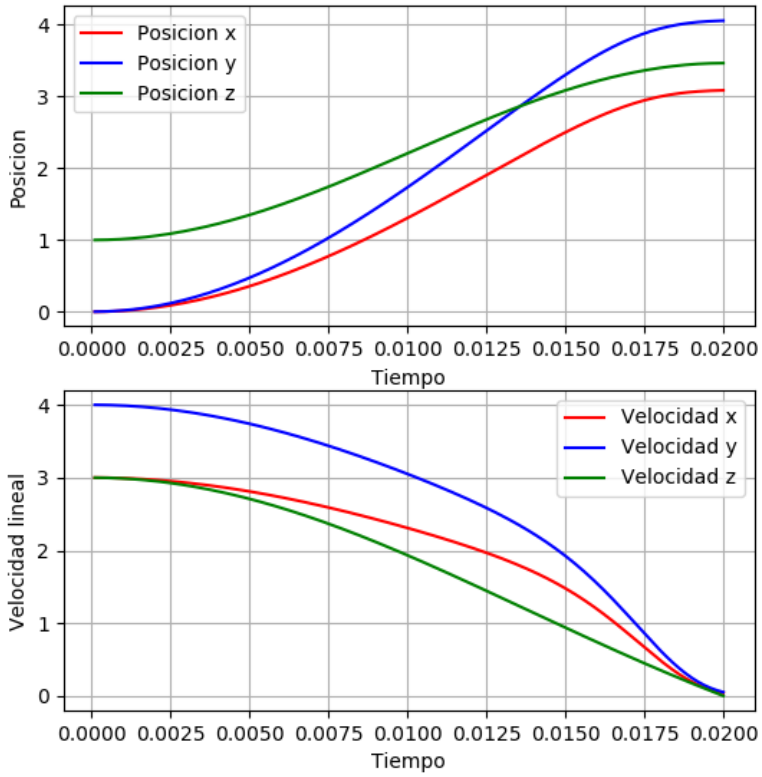


Figura 13.15: Zona angular a partir de nuevas entradas

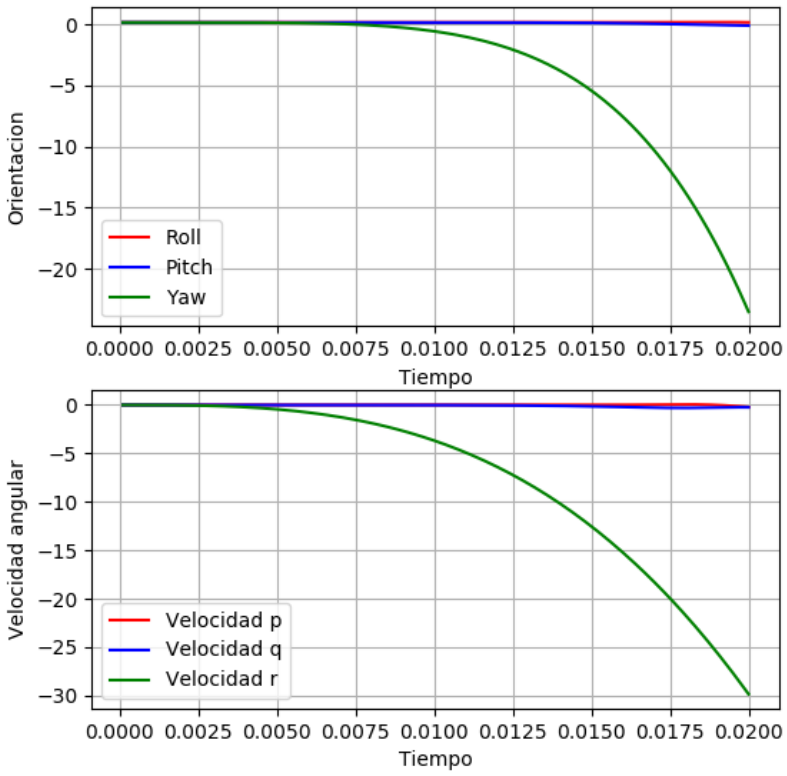


Figura 13.15: Zona angular a partir de nuevas entradas

Las posiciones, orientaciones, velocidades lineales, velocidades angulares e incremento de tiempo, se han guardado a partir de una lista vacía en la que iban entrando los datos calculados para cada iteración. A continuación se muestra la solución obtenida en la ventana de resultados de Python de las posiciones, incremento de tiempo y velocidades lineales:

```
Posición x
[0.0, 0.00029998655904145557, 0.0008999012958930962, 0.0017996385918467924, 0.0029990456161374996, 0.004497922363976499, 0
 .006296021707190476, 0.008839039457456298, 0.01078866444211859, 0.013482478592574527, 0.016474057045207467, 0.019762918254848303,
 0.023348534120740658, 0.027230330124983153, 0.031407685483419254, 0.035879933308942215, 0.040646360787179984, 0.045706209364520 7,
 0.051058674948440574, 0.0567029081200846, 0.06263801435905769, 0.06886305428037232, 0.07537704388349944, 0.08217895481346484, 0
 .08926771463392942, 0.09664220711218827, 0.1043012725160188, 0.11224370792230455, 0.12046826753735579, 0.1289736630288437, 0
 .13775856386925908, 0.14682159769080083, 0.15616135065159276, 0.1657763678131208, 0.1756651535287744, 0.18582617184336778, 0
 .1962578469035073, 0.20695856337866145, 0.2179266668927777, 0.2291604644662791 6, 0.2406582249682595, 0.2524181795786799, 0
 .2644385222603548, 0.2767174102404957, 0.2892529645015606, 0.3020432702811368, 0.3150863775805574, 0.3283803016819289, 0
 .3419230236732143, 0.3557124909809878, 0.36974661791043884, 0.3840232861921688, 0.39854034553527984, 0.4132956141862111, 0
 .4282868794927298, 0.4435118984724311, 0.45896839838504355, 0.47465407730777615, 0.4905666047128765, 0.5067036220464973, 0
 .5230627433078936, 0.5396415556278885, 0.5564376198454591, 0.5734484710811999, 0.5906716193063173, 0.6081045499057 07, 0.6257447242335451,
 0.643589580159709, 0.661636532605208, 0.6798829740646731, 0.6983262751138057, 0.7169637848995349, 0.7357928316104682, 0
 .7548107229250522, 0.7740147464346799, 0.7934021700387898, 0.8129702423088064, 0.8327161928175633, 0.8526372324306288, 0
 .8727305535557324, 0.8929933303462451, 0.9134227188544265, 0.9340158571298876, 0.9547698652584518, 0.9756818453363 22, 0.9967488813741686,
 1.0179680391254 6, 1.0393363658330466, 1.0608508898876994, 1.0825086203919 72, 1.1043065466224316, 1.1262416373829556, 1.148310840241458,
1.1705110806420427, 1.192839260884242, 1.215292258960631, 1.2378669272437544, 1.2605600910129444, 1.2833685468112583, 1.3062890606224098,
 1.3293183658572367, 1.352453161138918, 1.3756901078758434, 1.3990258276107523, 1.422456899134491, 1.4459798553525094, 1
 .4695911798920243, 1.4932873034376246, 1.5170645997830006, 1.5409193815864466, 1.5648478958178236, 1.5888463188847919, 1
 .6129107514263474, 1.6370372127620207, 1.661221634985555, 1.6854598566924845, 1.7097476163317902, 1.7340805451727617, 1.7584541598793402,
 1.7828638546856095, 1.8073048931677325, 1.831772399609572 5, 1.8562613499614802, 1.8807665623943384, 1.9052826874539477, 1
 .9298041978242697, 1.9543253777119391, 1.9788403118688798, 2.0033428742748316, 2.0278267165071946, 2.0522852558318 39, 2.0767116630554923,
 2.1010988501880328, 2.1254394579715403, 2.149725843342336, 2.1739500669025 15, 2.1981038804886914, 2.2221787149378374, 2
 .2461656681632762, 2.2700554936680324, 2.293838589637895, 2.3175049887726673, 2.3410443490310966, 2.3644459454828 61, 2.3876986634795903,
2.410790993376104, 2.4337110270526385, 2.4564464565086044, 2.478984574818031, 2.501312279755983, 2.5234160804234262, 2.5452821072147804,
2.566896125487103, 2.5882435533018597, 2.6093094836187216, 2.6300787113249577, 2.650535765482755, 2.67066494716915, 2.6904503732680416,
2.7098760265497233, 2.728925813399277, 2.747583622029654, 2.7658334036370453, 2.7836592395239252, 2.8010454313277733, 2.8179765920299253,
 2.8344377449784823, 2.8504144295412814, 2.865892812909545, 2.8808598074004608, 2.895303192418753, 2.9092117400351 28, 2.9225753429261 12,
2.9353851431989586, 2.9476336604018125, 2.9593149167991797, 2.970424557783049, 2.980959965099148, 2.990920360405129, 3.0003068965533704,
3.0091227339165387, 3.017373099060483, 3.0250653231277123, 3.0322088574361814, 3.0388152640316477, 3.044898179264543, 3.050473248898281,
3.0555580337954673, 3.060171885867292, 3.0643357946995744, 3.068072206070832, 3.071404814431125, 3.0743583322870043, 3.0769582403033855,
3.0792305227485826, 3.0812013936317197, 3.082897019468719, 3.0843432450218553, 3.085565328550634, 3.086587693058192]

Posición y
[0.0, 0.00039998175970914205, 0.0011998655235007758, 0.0023995066638900078, 0.003998695591410596, 0.005997158056669258, 0
 .008394554969919383, 0.011190482612787418, 0.014384472717345284, 0.017975992587436405, 0.02196444523723957, 0.026349169547040 99,
 0.031129440436180 85, 0.036304469053136904, 0.041873402982703506, 0.047835326470220 55, 0.054189260662802 71, 0.060934163867515 19,
 0.06806893182643796, 0.07559239800855633, 0.08350333391841097, 0.09180044942143636, 0.10048239308591154, 0.10954775254144238, 0
 .11899505485388957, 0.1288227669166509, 0.13902929585820084, 0.14961298946578477, 0.16057213662515812, 0.17190496777625 43, 0
 .18360965538465795, 0.19568431442875148, 0.2081270029023946, 0.2209357223329872, 0.2341084183147545, 0.24764298105708363, 0
 .2615372459477263, 0.2757889941306708, 0.29039595309846883, 0.3053557972987 88, 0.3206661487549420 4, 0.3363245777001292 5, 0
 .3523286032250895, 0.36867569393886407, 0.38536326864231624, 0.40238869701404123, 0.41974930030826135, 0.437442352064267, 0
 .455465078826925, 0.47381466087773416, 0.49248823297586086, 0.5114828851085375, 0.5307956632501521, 0.5504235701292987, 0
 .5703635660029901, 0.5906125694371721, 0.6111674580925947, 0.6320250695150202, 0.6531822019286 6, 0.6746356150316343, 0.6963820307921533,
0.7184181342440054, 0.7407405742798245, 0.7633459644404856, 0.7862308836988419, 0.8093918772358801, 0.8328254572072189, 0
 .8565281034977144, 0.8804962644617719, 0.9047263576467807, 0.9292147704968999, 0.9539578610342253, 0.9789519585141538, 1
 .0041933640515424, 1.0296783512140208, 1.0554031665785792, 1.0813640302247284, 1.1075571363177144, 1.1339786533034286, 1.1606247244994685,
 1.1874914682876132, 1.214574978375765, 1.2418713239655246, 1.2693765498416656, 1.2970866763768616, 1.3249976994446 51, 1.353105590233247,
 1.381406294952402, 1.4098957344251437, 1.438569803555779, 1.4674243706651457, 1.4964552766836623, 1.5256583341922 92, 1.5550293263010933,
 1.5845640053545875, 1.6142580914527274, 1.6441072707758062, 1.6741071937012038, 1.7042534726994325, 1.7345416799965212, 1
.7649673449893635, 1.7955259514002653, 1.8262129341565587, 1.8570236759800 09, 1.8879535036768424, 1.9189976840965 63, 1.9501514197723 22,
1.9814098441995 57, 2.0127680167535544, 2.0442209172269936, 2.075763439969451, 2.1073903876172624, 2.1390964643968986, 2.1708762689883088,
 2.2027242869345587, 2.2346348825850466, 2.2666022905606593, 2.2986206067305424, 2.330683778691751, 2.3627855957449 33, 2.3949196783614 34,
 2.4270794671398086, 2.459258211252761, 2.491448956389004, 2.5236445321985 39, 2.5558375392543904, 2.5880203355549009, 2.6201850225493 77,
2.6523234308414043, 2.684427105401542, 2.7164872905417 32, 2.7484949145828 85, 2.7804405743221 58, 2.8123145193703 7, 2.8441066364480836,
2.8758064337421607, 2.9074030254391117, 2.9388851165672323, 2.9702409882964713, 3.0014584838631273, 3.0325249953058 89, 3.0634274512203 03,
 3.094152305760455, 3.1246855291393634, 3.1550125999031375, 3.185118499278205, 3.2149877079155305, 3.2446042053804867, 3.2739514727614 58,
 3.3030124987938914, 3.3317697899187935, 3.3602053847149502, 3.3883008731616 19, 3.4160374212022 33, 3.4433958010887737, 3.4703564279897 6,
3.4968994033410 11, 3.5230045654061 5, 3.5486515474917035, 3.5738198442280 74, 3.5984888862810283, 3.6226381237969516, 3.6462471188073597,
3.6692956467224582, 3.6917638069284533, 3.7136321423676 37, 3.7348817678231 02, 3.7554945064508174, 3.7754530339007553, 3.7947410291466 65,
3.8133433309024007, 3.8312460982441543, 3.8484369737860 75, 3.8649052474766723, 3.8806420188013213, 3.8956403548999 69, 3.9098954418480445,
 3.923404726113 3105, 3.9361686043004073, 3.9481877287779 55, 3.9594687129999393, 3.9700185877324636, 3.9798476502477342, 3.9889689161206 52,
3.9973980999158885, 4.005153561095407, 4.012256213362451, 4.018729396358612, 4.024598709460276, 4.029891808360457, 4.0346381661513 28,
4.038868801712492, 4.042615979323137, 4.045912884508644, 4.048793282153616, 4.051291163809949, 4.053440391845163, 4.0552743485599 93,
4.056825598608 2935, 4.0581255729398755]
```

Posición z

[1.0, 1.0002999822433438, 1.0008998624558572, 1.0017994832388877, 1.002998614107349, 1.004496951547284, 1.0062941190929238, 1
.0083896674232224, 1.010783074477826, 1.0134737455924399, 1.0164610136535441, 1.0197441392723998, 1.023322310978285, 1.0271946454308898,
1.0313601876517926, 1.035817911274934, 1.0405667188159953, 1.0456054419605862, 1.0509328418711328, 1.0565476095123543, 1
.0624483659952124, 1.0686336629392028, 1.0751019828528587, 1.0818517395323264, 1.0888812784778654, 1.0961888773281243, 1
.1037727463120264, 1.1116310287181084, 1.1197618013811312, 1.128163075185793, 1.1368327955873518, 1.1457688431489763, 1.1549690340956207,
1.164431120884225, 1.1741527927900337, 1.1841316765088148, 1.1943653367747653, 1.2048512769938735, 1.2155869398925099, 1
.2265697081810119, 1.2377969052320204, 1.2492657957733213, 1.2609735865949443, 1.272917427270 26, 1.2850944108908178, 1.2975015748146577,
1.3101359014278278, 1.322994318918833, 1.3360737020657365, 1.349370873035636, 1.3628826021962204, 1.3766056089391274, 1.3905365625148003,
1.4046720828785506, 1.4190087415475268, 1.4335430624682828, 1.4482715228946412, 1.4631905542755408, 1.4782965431525563, 1
.4935858320667739, 1.5090547204747058, 1.5246994656729222, 1.5405162837310795, 1.5565013504330172, 1.572650802225601, 1.5889607371749808,
1.6054272159299325, 1.6220462626919547, 1.6388138666191784, 1.6557259806719977, 1.6727785268753672, 1.689967393038623, 1
.7072884358912994, 1.7247374816593173, 1.7423103270729676, 1.7600027403789582, 1.7778104623561821, 1.795729207334874, 1.81375466421881,
1.8318824975102168, 1.8501083483370497, 1.8684278354823047, 1.8868365564150236, 1.9053300883226612, 1.9239039891444762, 1
.9425537986056138, 1.9612750392515457, 1.9800632174825383, 1.9989138245878193, 2.017822337779111, 2.0367842212232117, 2.0557949270732925,
2.074849896498592, 2.0939445607121896, 2.113074341996536, 2.1322346547264286, 2.151420906389116, 2.1706284986012268, 2.1898528281222056,
2.209089287863963, 2.228333267896428, 2.24758015644871, 2.2668253409055716, 2.2860642087989214, 2.305292148794036, 2.324504551670228,
2.3436968112956733, 2.3628643255961226, 2.38200249751722, 2.401106735980154, 2.4201724568303824, 2.4391950837791496, 2.4581700493375545,
2.4770927957428963, 2.4959587758770554, 2.514763454176651, 2.5335023075347376, 2.552170826193794, 2.5707645146297646, 2.5892788924269228,
2.6077094951433235, 2.6260518751666164, 2.64430160256, 2.662454265898092, 2.680505473092501, 2.6984508522068933, 2.7162860522613363,
2.734006744025716, 2.7516086208020316, 2.769087399195361, 2.786438819873307, 2.8036586483137307, 2.8207426755405853, 2.837686718847661,
2.8544866225100 66, 2.871138258483259, 2.8876375270894696, 2.9039803576913212, 2.9201627093525047, 2.936180571485333, 2.952029964485021,
2.967706940350541, 2.9832075832919025, 2.9985280103237186, 3.013664371844924, 3.028612852204517, 3.0433696702532043, 3.0579310798808415,
3.0722933705395663, 3.086452867752537, 3.1004059336081937, 3.1141489672399834, 3.127678405291496, 3.140990722366981, 3.1540824314672258,
3.1669500844108014, 3.1795902722406986, 3.1919996256164023, 3.2041748151914766, 3.216112551976751, 3.227809587689232, 3.239262715086894,
3.2504687682895 12, 3.26142462308576, 3.272127197226781, 3.2825734507065083, 3.2927603860290033, 3.3026850484631174, 3.3123445262847975,
3.3217359510073616, 3.3308564976000907, 3.3397033846954693, 3.348273874785407, 3.3565652744067505, 3.364574934316372, 3.3723002496560763,
3.379738660107512, 3.3868876500372185, 3.3937447486318564, 3.4003075300235697, 3.406573613405349, 3.412540663136132, 3.418206388835279,
3.4235685454659 28, 3.4286249334066263, 3.433373398510525, 3.4378118321513207, 3.4419381712550603, 3.4457503983168856, 3.4492465414017754,
3.4524246741283955, 3.4552829156352565, 3.4578194305285312, 3.460032428811115, 3.4619201657927765, 3.4634809419816173, 3
.4647131029574427, 3.4656150392281027, 3.4661851860703146, 3.4664220233569565]

Tiempo

[0.0001, 0.0002, 0.00030000000000000003, 0.0004, 0.0005, 0.0006000000000000001, 0.0007000000000000001, 0.0008000000000000001, 0
.0009000000000000002, 0.0010000000000000002, 0.0011000000000000003, 0.0012000000000000003, 0.0013000000000000004, 0.0014000000000000004,
0.0015000000000000005, 0.0016000000000000005, 0.0017000000000000006, 0.0018000000000000006, 0.0019000000000000006, 0.0020000000000000005,
0.0021000000000000003, 0.0022, 0.0023, 0.0024, 0.0024999999999999996, 0.0025999999999999994, 0.0026999999999999993, 0
.0027999999999999999, 0.0028999999999999999, 0.0029999999999999988, 0.0030999999999999986, 0.0031999999999999984, 0.0032999999999999982,
0.0033999999999999998, 0.0034999999999999998, 0.0035999999999999977, 0.0036999999999999976, 0.0037999999999999974, 0.0038999999999999972,
0.0039999999999999975, 0.004099999999999998, 0.004199999999999998, 0.004299999999999998, 0.004399999999999985, 0.004499999999999999,
0.004599999999999999, 0.004699999999999999, 0.0048, 0.0049, 0.005, 0.0051, 0.005200000000000001, 0.005300000000000001, 0
.005400000000000001, 0.005500000000000001, 0.005600000000000002, 0.005700000000000002, 0.005800000000000002, 0.005900000000000025,
0.006000000000000003, 0.006100000000000003, 0.006200000000000003, 0.006300000000000035, 0.006400000000000004, 0.006500000000000004,
0.006600000000000004, 0.006700000000000046, 0.006800000000000005, 0.006900000000000005, 0.007000000000000005, 0.007100000000000006,
0.007200000000000006, 0.007300000000000006, 0.007400000000000006, 0.007500000000000007, 0.007600000000000007, 0.007700000000000007,
0.007800000000000074, 0.007900000000000008, 0.008000000000000007, 0.008100000000000007, 0.008200000000000006, 0.008300000000000005,
0.008400000000000005, 0.008500000000000004, 0.008600000000000003, 0.008700000000000003, 0.008800000000000002, 0.008900000000000002,
0.009000000000000001, 0.0091, 0.0092, 0.0093, 0.009399999999999999, 0.009499999999999998, 0.009599999999999997, 0.009699999999999997,
0.009799999999999996, 0.009899999999999996, 0.009999999999999995, 0.010099999999999994, 0.010199999999999994, 0.010299999999999993,
0.010399999999999993, 0.010499999999999992, 0.010599999999999991, 0.010699999999999999, 0.010799999999999999, 0.010899999999999999,
0.010999999999999989, 0.011099999999999988, 0.011199999999999988, 0.011299999999999987, 0.011399999999999987, 0.011499999999999986,
0.011599999999999985, 0.011699999999999985, 0.011799999999999984, 0.011899999999999984, 0.011999999999999983, 0.012099999999999982,
0.012199999999999982, 0.012299999999999981, 0.012399999999999998, 0.012499999999999998, 0.012599999999999998, 0.012699999999999979,
0.012799999999999978, 0.012899999999999977, 0.012999999999999977, 0.013099999999999976, 0.013199999999999976, 0.013299999999999975,
0.013399999999999974, 0.013499999999999974, 0.013599999999999973, 0.013699999999999973, 0.013799999999999972, 0.013899999999999971,
0.013999999999999997, 0.014099999999999997, 0.014199999999999997, 0.014299999999999969, 0.014399999999999968, 0.014499999999999968,
0.014599999999999967, 0.014699999999999967, 0.014799999999999966, 0.014899999999999965, 0.014999999999999965, 0.015099999999999964,
0.015199999999999964, 0.015299999999999963, 0.015399999999999962, 0.015499999999999962, 0.015599999999999961, 0.015699999999999996,
0.015799999999999996, 0.015899999999999996, 0.015999999999999996, 0.016099999999999958, 0.016199999999999957, 0.016299999999999957,
0.016399999999999956, 0.016499999999999956, 0.016599999999999955, 0.016699999999999954, 0.016799999999999954, 0.016899999999999953,
0.016999999999999953, 0.017099999999999952, 0.017199999999999995, 0.017299999999999995, 0.017399999999999995, 0.017499999999999995,
0.017599999999999995, 0.017699999999999995, 0.017799999999999948, 0.017899999999999947, 0.017999999999999947, 0.018099999999999946,
0.018199999999999945, 0.018299999999999945, 0.018399999999999944, 0.018499999999999944, 0.018599999999999943, 0.018699999999999942,
0.018799999999999942, 0.018899999999999994, 0.018999999999999994, 0.019099999999999994, 0.019199999999999994, 0.019299999999999994, 0
.019399999999999938, 0.019499999999999938, 0.019599999999999937, 0.019699999999999936, 0.019799999999999936, 0.019899999999999935,
0.019999999999999934]

Velocidad x

[2.9999443463706668, 2.9997311847835904, 2.9993605409347084, 2.9988324657605557, 2.998147035433078, 2.997304351350581, 2.9963045401248354,
2.9951477535642814, 2.9938341686533403, 2.9923639875278036, 2.990737437446285, 2.9889547707577067, 2.9870162648647907, 2.9849222223183521,
2.9826729700985455, 2.9802688609144643, 2.9777102718029584, 2.974997604745694, 2.9721312864729343, 2.969111768397768, 2.965939526545861,
2.962615061480611, 2.9591388982235616, 2.955511586169932 6, 2.951733698999058, 2.9478058345795306, 2.9437286148687933, 2
.9395026858068976, 2.9351287172040887, 2.9306074026218463, 2.9259394592469463, 2.9211256277580673, 2.9161666721843713, 2.911063379755469,
2.9058165607420516, 2.9004270482864256, 2.8948956982220984, 2.8892233888814425, 2.8834110208903927, 2.8774595169490147, 2
.8713698215966454, 2.865142900960229, 2.8587797424842902, 2.852281354640877, 2.8456487666176598, 2.8388830279821957, 2.8319852083202384,
2.8249563968457587, 2.8177977019802083, 2.8105102508983295, 2.8030951890376614, 2.795553679568638, 2.787886902822012, 2.7800960556700813,
2.772182350857996, 2.764147016281163, 2.755991294204547, 2.7477164404194028, 2.739323723332724, 2.7308144229844307, 2.7221898299870513,
2.713451244382385, 2.7045999744093394, 2.6956373351768717, 2.6865646472356617, 2.677383235041873, 2.668094425306035, 2.65869954521982,
2.6491999205531624, 2.6395968736138964, 2.6298917210617683, 2.6200857715684136, 2.610180323314567, 2.600176661315512, 2.590076054565487,
2.579879752991481, 2.569588984206604, 2.559204950052947, 2.5487288229236094, 2.538161741853353, 2.52750480836711 37, 2.516759082075429,
2.5059255760056 69, 2.495005251657812, 2.4839990137734147, 2.472907704806338, 2.461732099083771, 2.4504728966461093, 2.4391307167543013,
2.427706091053424, 2.4161994563814 21, 2.4046111472122345, 2.392941387722 89, 2.3811902834745857, 2.369357812698 36, 2.357443817176643,
2.345447992712795, 2.3333698791817286, 2.321208850155878, 2.308964102102119, 2.2966346431468208, 2.2842192814080415, 2.2717166128959505,
2.25912500898497, 2.246442603463863, 2.233667279173089, 2.22079665424230 17, 2.2078280679448183, 2.194758566190429, 2.1815848866829253,
2.1683034437744566, 2.154910313055142, 2.141401215723485, 2.1277715027910564, 2.1140161391837022, 2.1001296878113003, 2.0861062936889097,
2.071939668204074, 2.057623073638198, 2.043149308064361, 2.0285106907597505, 2.0136990482882253, 1.9987057014273335, 1.9835214531346286,
1.9681365777702677, 1.9525408118168255, 1.9367233463629585, 1.9206728216450684, 1.904377323970448, 1.887824385376454, 1
.8710009864130142, 1.8538935624700639, 1.8364880140107181, 1.8187697218794285, 1.800723566190943, 1.7823339527456517, 1.7635848442021334,
1.744459798676384, 1.7249420157712958, 1.705014390844063, 1.6846595782512745, 1.663860064334891, 1.6425982509289734, 1.6208565501752001,
1.5986174914328226, 1.575863841053437, 1.5525787357602487, 1.5287458303225328, 1.5043494601456118, 1.4793748193015848, 1
.4538081544026453, 1.4276369745633979, 1.4008502775073275, 1.3734387916415975, 1.3453952336499837, 1.3167145808324683, 1
.2873943570488244, 1.2574349307000308, 1.2268398227041286, 1.1956160218919551, 1.163774304664431, 1.1313295551200555, 1.0983010811847154,
1.0647129215643574, 1.030594137606402, 0.9959790834135346, 0.9609076468234998, 0.9254254531750142, 0.8895840231519672, 0
.8534408754696494, 0.8170595647765041, 0.7805096449359489, 0.7438665478718 67, 0.7072113684577069, 0.6706305465529359, 0.6342154382908665,
0.5980617701437246, 0.5622689711721764, 0.5269393812343223, 0.49217733579504785, 0.4580881313322993, 0.42477687914935175, 0
.392347259609256, 0.3609001933139118, 0.3305324504239939, 0.301335223988098, 0.2733926976141704, 0.2467806418347236, 0.22156507682468268,
0.1978010414468145, 0.17553150964375894, 0.15478649470468664, 0.13558237968496406, 0.11792150808865391, 0.10179206276380585, 0
.08716825284779318, 0.07401081870292509, 0.062267853412030866, 0.05187592701547002, 0.04276148685640739]

Velocidad y

[3.99992579516089, 3.99963519957384, 3.999128248300085, 3.9984050110772467, 3.997465592311963, 3.996310131067087, 3.9949388010434306,
3.9933518105560437, 3.991549402505009, 3.9895318543407132, 3.987299478023592, 3.9848526199782928, 3.982191661042236, 3.97931701640852,
3.9762291355631274, 3.9729285022163787, 3.9694156342285516, 3.9656910835296, 3.9617554360328713, 3.9576093115427056, 3.953253363655795,
3.948688279656141, 3.9439147804034285, 3.9389336202146064, 3.9337455867384272, 3.928351500822 65, 3.9227522163735853, 3.9169486202077593,
3.910941631894093, 3.9047322035895897, 3.8983213198621414, 3.891709997505621, 3.884899285343049, 3.8778902640181707, 3.8706840457743614,
3.863281774219832, 3.8556846240780014, 3.8478938009217503, 3.839910540890184, 3.831736110386337, 3.8233718057541 36, 3.814818952932759,
3.806078907086357, 3.797153052206936, 3.7880428006879767, 3.77874959286619, 3.76927489652858, 3.759620206381773, 3.7497870434803127,
3.739776954610416, 3.7295915116253737, 3.719232310728582, 3.7087009716998414, 3.697999137060339, 3.687128471171392, 3.6760906592617517,
3.664887406377929, 3.6535203436251709 3, 3.641991490078677, 3.6303023252012308, 3.6184547136892395, 3.606450440811131, 3.594291303387845,
3.5819791080217316, 3.5695156691921017, 3.5569028072087776, 3.544142346014603, 3.5312361108275288, 3.5181859256124888, 3
.5049936103729284, 3.491660978251475, 3.4781898324288 78, 3.464581962809972, 3.450839142485091, 3.4369631239550014, 3.4229556351070856,
3.408818374930217, 3.394553008955432, 3.3801611644092455, 3.365644425066195, 3.3510043257869473, 3.3362423467281217, 3.3213599072097844,
3.306358359226459, 3.2912389805874045, 3.2760029676718547, 3.2606514277849583, 3.245185371100222, 3.2296057021744096, 3.2139132110211133,
3.1981085637294897, 3.1821922926151434, 3.166164785890639, 3.150026276843825, 3.1337768325129782, 3.117416341848747, 3.100944503354085,
3.0843608121946957, 3.067664546774179, 3.0508547547698 95, 3.0339302386277516, 3.0168895405165936, 2.999730926745715, 2.982452371652252,
2.9650515409689158, 2.9475257746866577, 2.9298720694316325, 2.9120870603810842, 2.8941670027488002, 2.876107752877463, 2.857904748982747,
2.83955299160238 6, 2.821047023812784, 2.802380911286131, 2.783548222272517, 2.7645420076043004, 2.7453547808340804, 2.725978498633175,
2.706404541594587, 2.6866236956031573, 2.6666261339561337, 2.646401400439708, 2.6259383935914067, 2.6052253524045765, 2.5842498437597015,
2.562998751897941, 2.5414582702852244, 2.5196138962503087, 2.497450428817611, 2.474951970195069, 2.4521019314188193, 2.4288830426997827,
2.405277369062195, 2.3812663319101977, 2.3568307372055135, 2.3319508109862355, 2.306606243003288, 2.2807762392961304, 2.254439584571817,
2.2275747152902685, 2.200159804392052, 2.1721728586313427, 2.1435918294940666, 2.1143947386870856, 2.084559819176133, 2
.0540656727250153, 2.0228914448430793, 1.9910170179784719, 1.9584232236973258, 1.9250920744594482, 1.8910070154348457, 1
.8561531965978308, 1.8205177650816 8, 1.7840901774721303, 1.7468625313578128, 1.7088299150357824, 1.6699907737870 48, 1.6303472905877183,
1.5899057785045845, 1.5486770813398607, 1.506676978340602, 1.4639265879788608, 1.4204527649468373, 1.3762884836086877, 1.33147320022313,
1.2860531853190982, 1.2400818166964338, 1.1936198226665553, 1.1467354643819674, 1.0995046454720394, 1.052010936755452, 1.0043455035920 13,
0.9566069235276137, 0.9089008823381322, 0.8613397374540771, 0.8140419391086532, 0.7671313014531238, 0.720736118370528, 0
.6749881218229957, 0.6300212842999986, 0.585970471279803, 0.5429699545271 29, 0.5011518024419229, 0.4606441694200995, 0.4215695121140281,
0.3840427663680373 6, 0.3481695241861363, 0.3140442550586473 6, 0.2817486199934208, 0.25134992931054384, 0.22289979631379842, 0
.19643303802027284, 0.17196687093791682, 0.14950044424001227, 0.12901474452112632, 0.1104728957005409, 0.09382086480290373, 0
.07898856970973635, 0.06589136914022198, 0.05443189885501005]

Velocidad z

[2.999944346370668, 2.999644872432873, 2.999101614704262, 2.9983146487689956, 2.997284089260871, 2.9960100898360205, 2.9944928431351845,
2.9927325807355794, 2.9907295730923757, 2.988484129469801, 2.9859965978618996, 2.983267364902978, 2.9802968557677594, 2.9770855340612874,
2.9736339016986264, 2.96994249877439, 2.9660119034221517, 2.9618427316637876, 2.9574356372488078, 2.952791311483728, 2.9479104830515555,
2.9427939178214424, 2.9374424186485837, 2.931856825164434, 2.926038013557311, 2.9199868963434765, 2.913704422128773, 2.9071915753608986,
2.9004493760724253, 2.893478879614635, 2.886281176382286, 2.8788573915294062, 2.8712086846762106, 2.863336249607259, 2.85524131396096,
2.84692513891053, 2.8383890188365384, 2.8296342809911352, 2.820662285154106, 2.8114744232808646, 2.802072119142521, 2.792456827958144 3,
2.7826300360193663, 2.7725932603074575, 2.7623480481030076, 2.751895976588363, 2.7412386524429566, 2.7303777114316836, 2.719314817986463,
2.708051664781145, 2.696589972299913, 2.684931488399333, 2.673077987864217, 2.6610312719574467, 2.648793167963932, 2.63636552872886,
2.623750232190403, 2.6109491809070504, 2.597964301579734, 2.584797544568915, 2.571450883406815, 2.557926314304942, 2.5442258556571153,
2.530351547538132, 2.5163054511982867, 2.502089648553892, 2.4877062416740015, 2.473157352263498, 2.458445121142741, 2.4435717077239487,
2.4285392894844997, 2.41335006143733, 2.398006235598622, 2.3825100404529542, 2.36686372041611, 2.351069535295715, 2.335129759749906,
2.319046682744192, 2.3028226070067177, 2.286459848482092, 2.269960735783975, 2.253327609646611, 2.236562822375481, 2.219668737297259,
2.202647728209264, 2.1855021788285716, 2.168234482240981, 2.150847040350002, 2.133342263326053, 2.115722569056031, 2.09799038259345,
2.0801481356092917, 2.0621982658437643, 2.044143216559132, 2.0259854359937775, 2.0077273768176642, 1.9893714955893682, 1
.9709202522148332, 1.9523761094080079, 1.9337415321535243, 1.9150189871715644, 1.8962109423850688, 1.8773198663894291, 1.8583482279248094,
1.8392984953512348, 1.82017313612658, 1.800974616287592, 1.7817053999340684, 1.7623679487163226, 1.7429647213260393, 1.7234981729906478,
1.7039707549713112, 1.6843849140646374, 1.6647430921082071, 1.6450477254900167, 1.6253012446619117, 1.6055060736570996, 1
.585664629611815, 1.5657793222912029, 1.545852553619482, 1.5258867172144526, 1.5058841979263873, 1.485847371381362, 1.4657786035290 57,
1.4456802501950683, 1.4255546566377553, 1.4054041571096532, 1.3852310744234642, 1.3650377195226526, 1.344826391056656, 1
.3245993749607288, 1.3043589440404255, 1.284107357560752, 1.2638468608399882, 1.2435796848482197, 1.2233080458105994, 1.2030341448153816,
1.1827601674267814, 1.1624882833027224, 1.1422206458175592, 1.1219593916898787, 1.1017066406155103, 1.0814644949059027, 1
.0612350391320589, 1.0410203397742588, 1.020822444877837, 1.0006433837153283, 0.98048516645535, 0.9603497838386342, 0.9402392068616839,
0.9201553864685906, 0.9001002532516061, 0.8800757171611305, 0.8600836672258364, 0.8401259712837175, 0.8202044757249033, 0
.8003210052471353, 0.780477362624846, 0.760675328492816, 0.7409166611454014, 0.7212030963523295, 0.701536347192034, 0.6819181039034609,
0.6623500337571947, 0.6428337809466451, 0.6233709664998811, 0.6039631882125078, 0.5846120206017384, 0.565319014881524, 0
.5460856989582621, 0.5269135774462155, 0.5078041317013269, 0.48875881987163666, 0.4697790769619804, 0.4508663149101022, 0
.4320219226707513, 0.41324726630377884, 0.39454368906172516, 0.3759125114719194, 0.35735503140773467, 0.33887252414338775, 0
.3204662423865821, 0.30213741628341323, 0.28388725339031406, 0.26571693860847695, 0.24762763407715757, 0.22962047902359337, 0
.2116965895689515 7, 0.1938570584917669, 0.17610295495270356, 0.1584353241871286, 0.14085518717482848, 0.12336354029911123, 0
.1059613550103553 8, 0.08864957751159001, 0.07142912848568057, 0.05430090288488636 4, 0.03726576980366505 5, 0.02032457245434246 2,
0.003478128262385332]

14. ENTORNO

Para representar el entorno se ha utilizado la librería matplotlib, especializada en gráficos a partir de datos contenidos en listas o arrays. Utiliza como extensión matemática Numpy.

Se ha implementado el modelo del dron en el entorno, ya que se pretende que su funcionamiento se base en las ecuaciones dinámicas y realice las acciones propias del modelo.

En el entorno se define el dron, los obstáculos y la zona final como distintas clases.

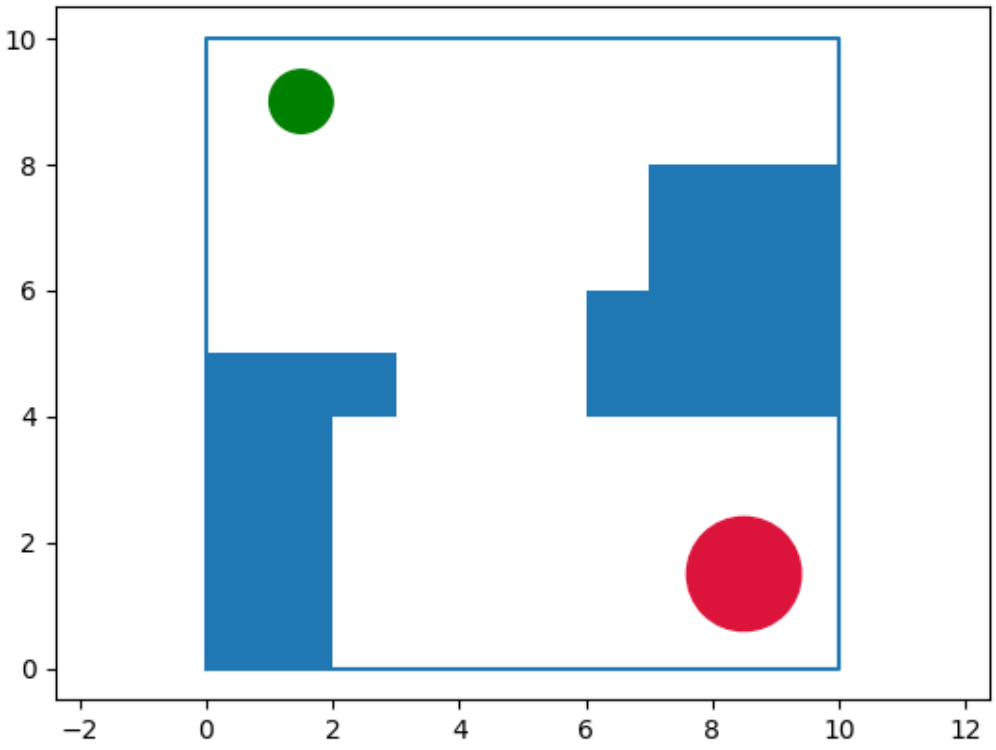


Figura 14-1: Entorno gráfico

Para representarlo se ha construido un grid de dimensión 10x10. Posteriormente, debido a que los pasos eran demasiado cortos porque al discretizar el modelo el incremento de tiempo es muy bajo, ha sido necesario repetir la seleccionada cuatro veces con un pequeño bucle. De esta manera el dron consigue llegar a la zona final en muchos menos pasos, lo cual va a facilitar que al conectarlo a la librería de Reinforcement Learning, el aprendizaje sea más rápido.

Las entradas del entorno son los momentos y el empuje del dron.

Vienen definidas como un input en el que se elige una de las 6 posibles acciones a tomar:

- La acción 0: Incrementa el empuje total del dron.
- La acción 1: Disminuye el empuje del dron.
- La acción 2: Aumenta el momento del roll.
- La acción 3: Reduce el momento del roll.
- La acción 4: Incrementa el momento del pitch.
- La acción 5: Disminuye el momento del pitch.
- La acción 6: Aumenta el momento del yaw.
- La acción 7: Disminuye el momento del yaw.

La acción elegida provoca una variación en la posición, orientación, velocidades y aceleraciones del dron. La modificación de los valores es de ± 0.001. En un principio se hizo esta transformación más grande, pero al realizar cambios bruscos, en las nuevas posiciones y velocidades, el sistema acababa siendo inestable al cabo de una serie de pasos.

Tras varias revisiones para encontrar el fallo que hacía inestable el sistema, se localizó en la aceleración angular del dron respecto del centro de masa. Al estar divida por las constantes de la matriz identidad que son del orden de 10^{-3}, pequeñas fluctuaciones de la velocidad angular respecto del centro de masas, provocan grandes cambios en la aceleración.

En cada paso es necesario elegir una acción, de manera que guarda los resultados obtenidos en listas que contienen valores decimales, floats, y dan información sobre la situación del dron.

Repetir la acción durante cuatro veces da mejor resultado que aumentar la modificación de cada acción porque de ésta manera los cambios en la dinámica del dron son cuatro veces menos bruscos. Esto ocurre porque repitiendo la acción se calculan las ecuaciones de la dinámica en cuatro ocasiones, mientras que si se incrementa la acción más notablemente, los cambios son mucho más bruscos, lo que hace que el sistema de control sea inestable.

La observación es el vector que contiene los datos obtenidos en el modelo y el entorno, que se van a introducir en la librería de reinforcement learning para realizar los experimentos.

El dron es el círculo verde, los obstáculos están formados por distintos bloques azules y la zona final es el círculo rojo. Para el aprendizaje del dron se ha diseñado un sistema de recompensas y castigos basado en lo siguiente:

- Un episodio contiene como máximo 500 pasos, pudiendo terminar antes en el caso de colisionar con el obstáculo o llegar completamente a la zona final.
- Por cada paso realizado sin llegar a la posición final tiene una penalización de -1, mientras que si se choca con el obstáculo, la penalización será de -50.
- Obtiene una recompensa de 50 al situarse por completo en la zona roja.

Ya que el modelo está definido en 3D y la representación sólo consta de dos ha sido necesario imponer una restricción sobre el eje z para controlar también la altura del dron. Se ha impuesto una penalización de -50 y finalización del episodio cuando supere los 20 metros. Para poder hacer un mejor seguimiento del dron se ha necesitado representar la posición del mismo respecto a los ejes x, y, z.

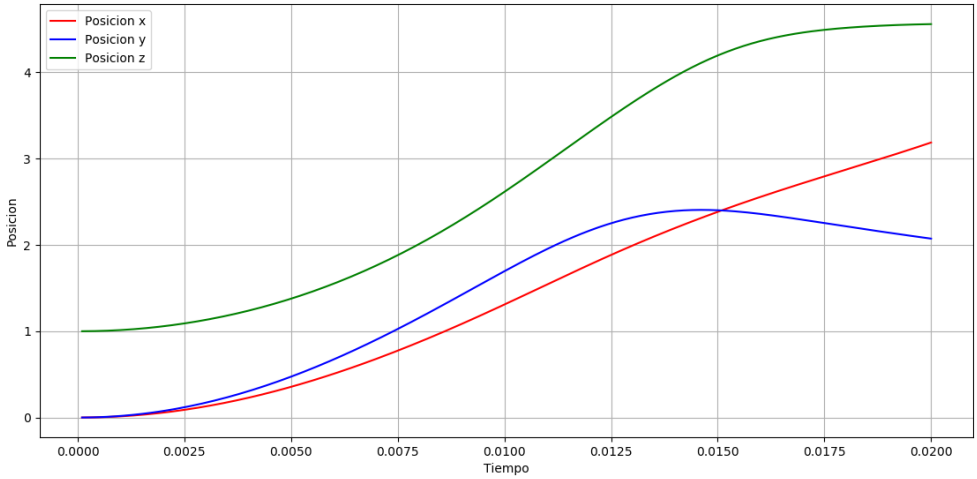


Figura 14-2: Gráfico posición frente al tiempo

De esta manera se conoce la posición exacta del dron en las 3 dimensiones a lo largo del tiempo.

Los datos que se envían a la librería de reinforcement learning son los guardados en la observación. Estos datos son 4 arrays que se encuentran dentro de una lista, de forma que la información de la posición, orientación, velocidad lineal y velocidad angular se envía a la red neuronal de manera separada, para que aplicando redes convolucionales, el algoritmo sea capaz de aprender.

Se ha comprobado que en un principio los pasos que hace el dron son más cortos que al final, esto sucede porque las velocidades iniciales son cero. A continuación se representa la variación de los dos primeros pasos y cuando se encuentra a la mitad del recorrido.

- Posición inicial del dron y primer paso:

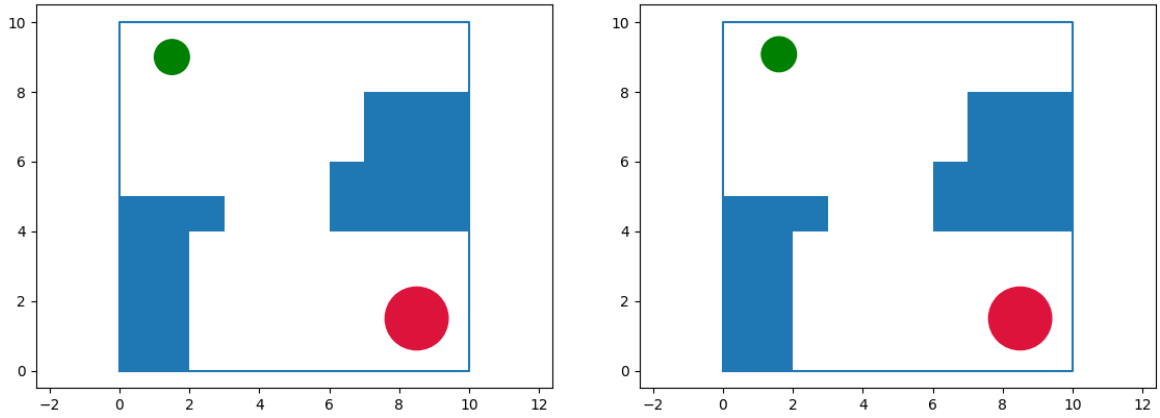


Figura 14-3: Primeras posiciones del dron

- Paso del dron en la mitad del recorrido, donde se comprueba que es más largo que el anterior:

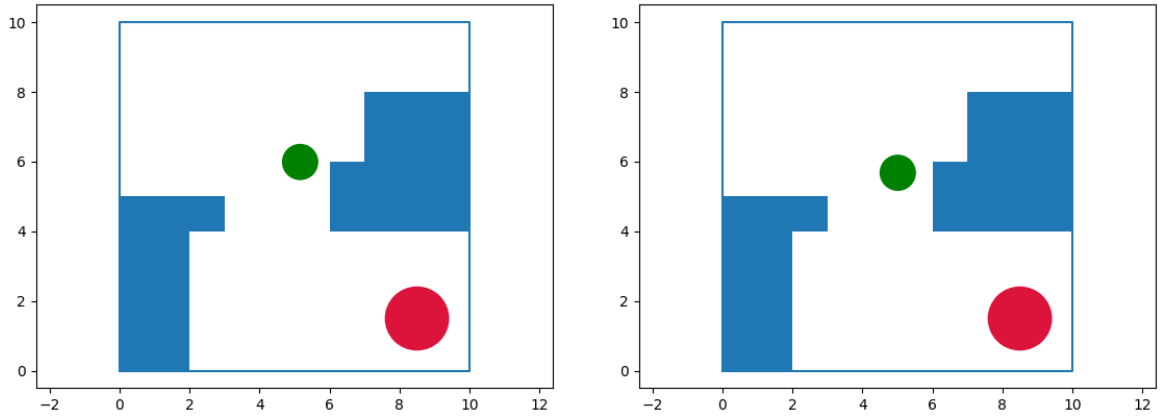


Figura 14-4: Posiciones intermedias del dron

Por otro lado se han impuesto condiciones para que el dron no pueda salir de los límites de la cuadrícula de 10x10, de manera que si colisiona con una de las paredes que delimitan el grid, se imprimirá por pantalla un mensaje mencionando que está fuera de los límites y te permitirá realizar una nueva acción.

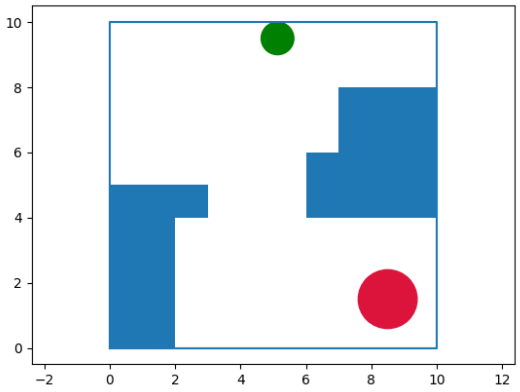


Figura 14-5: Colisión con el límite del entorno

```
Reward:
-1
Elige un acción: 1
Reward:
-1
Out of limits
Elige un acción:
```

Al igual que para la colisión del dron con los límites, al conseguir una altura superior a 20 metros tiene una penalización de -50. La posición del dron viene determinada por el centro del mismo, por lo que se tiene en cuenta el radio.

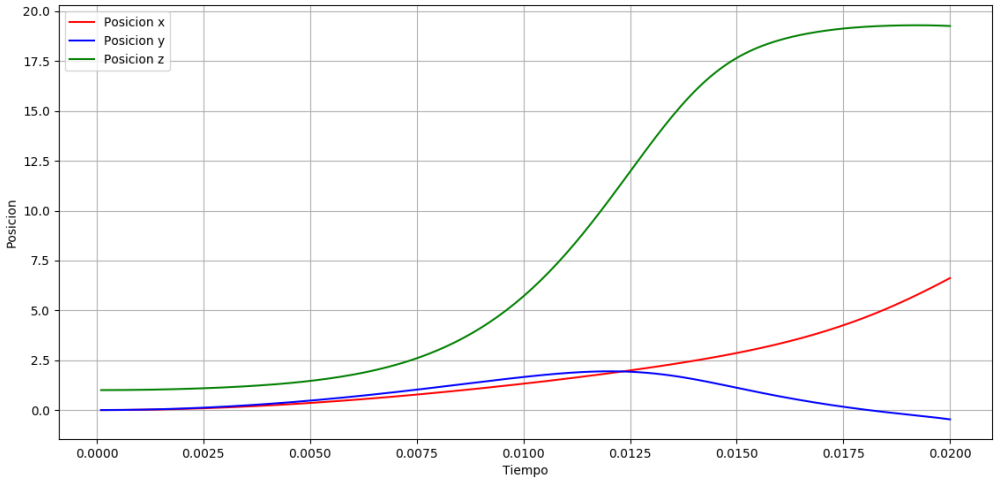


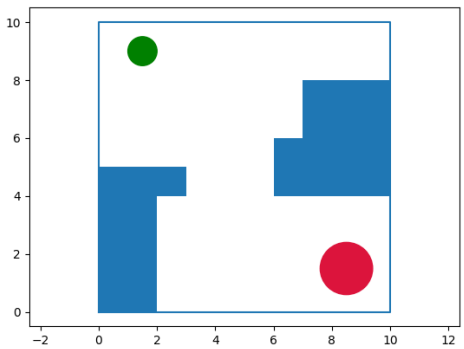
Figura 14-6: Penalización por altura máxima

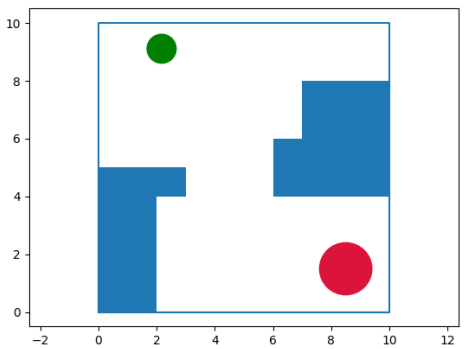
En la gráfica se observa que no llega a los 20 metros, sino que se queda en 19.5, ya que el radio del dron es de 0.5. El resultado que se imprime por pantalla es el siguiente:

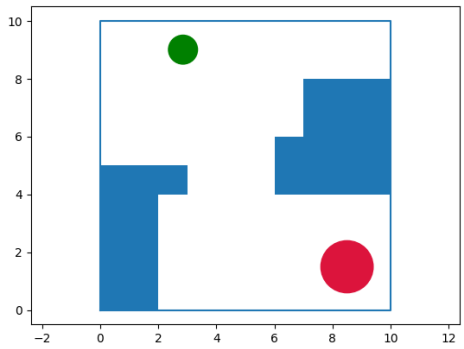
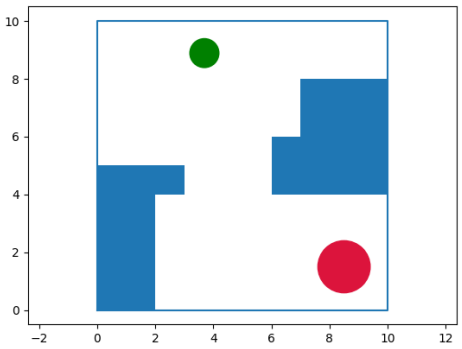
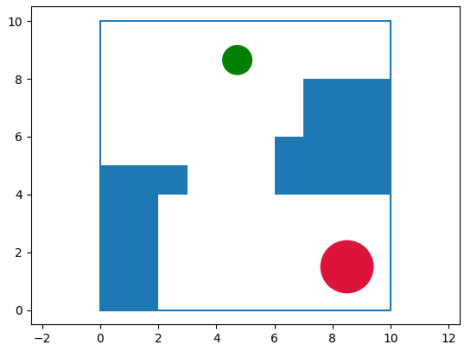
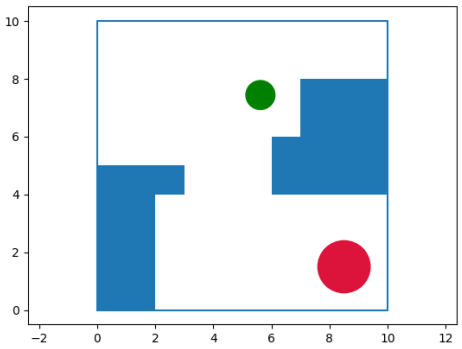
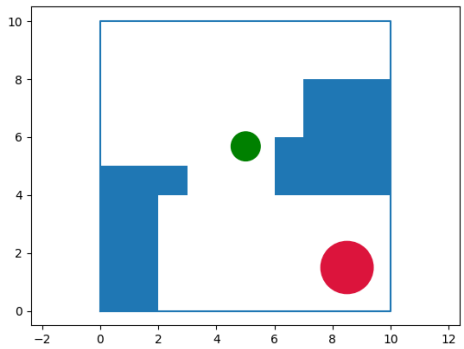
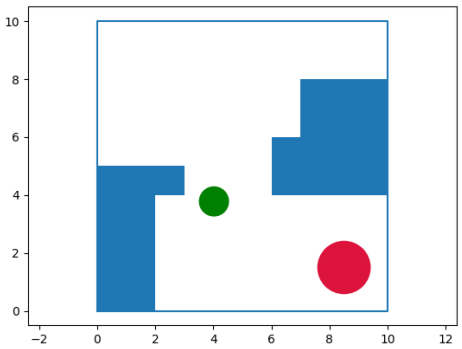
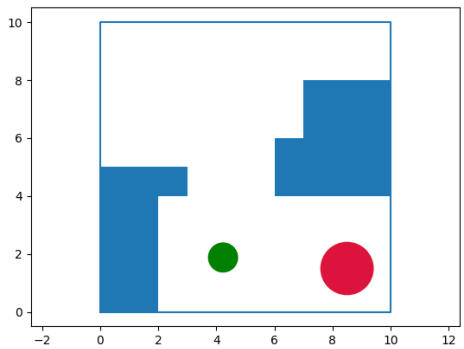
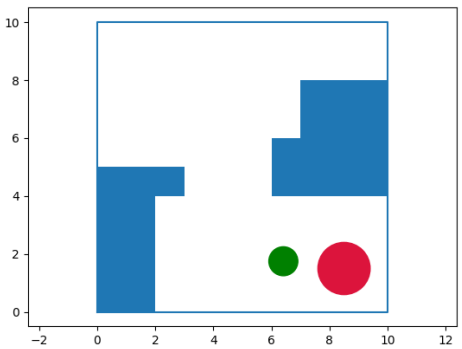
```
Reward:
-1
Elige un acción: 2
Reward:
-50
```

Tras una penalización de -50, es decir, al colisionar con un objeto o sobrepasar los límites de la altura, la función reset devuelve el dron a la posición inicial, cargando en la observación los valores de las posiciones y velocidades. La función reset también se utiliza cuando el dron se ha incorporado completamente en la superficie roja o zona final, obteniendo una recompensa de 50.

El dron se controla a partir de los empujes y momentos que se le aplica en cada paso. Esto dificulta su manejo, siendo complicado realizar una trayectoria sencilla. Por esta razón el recorrido que ha seguido el dron no es tan perfecto como se podría haber conseguido si los controles fuesen lineales. Un resumen del camino trazado hasta la base final, es el siguiente:





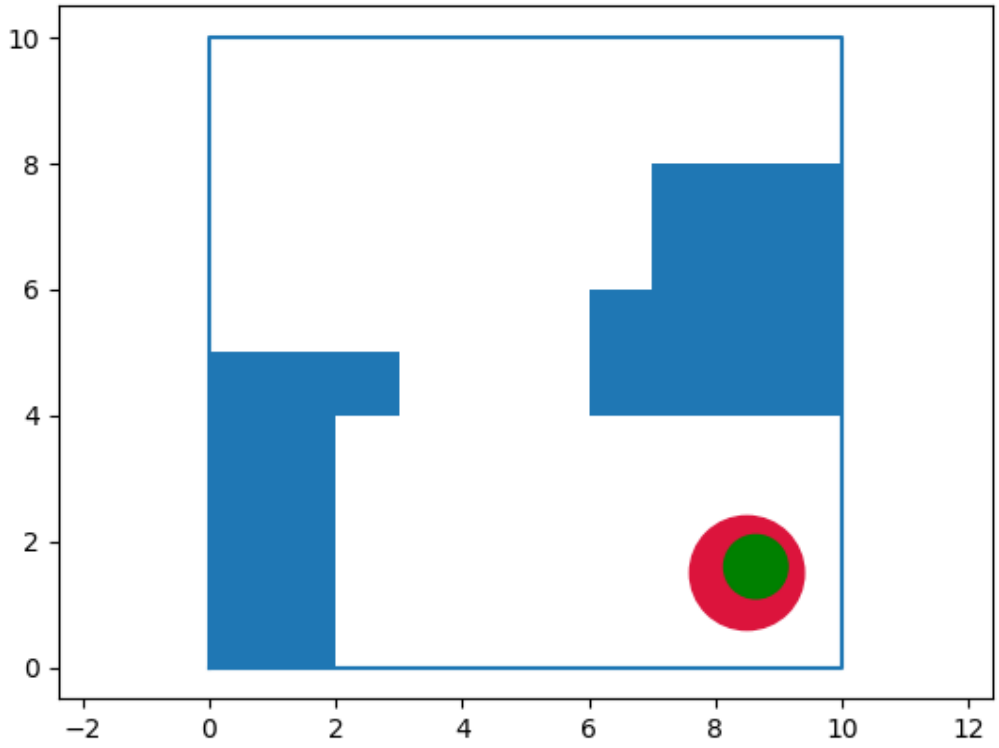
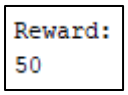


Figura 14-7: Trayectoria total del dron



15. ANÁLISIS DE LOS RESULTADOS

Los resultados finales del proyecto han servido como apoyo para entender mejor los algoritmos estudiados, principalmente en los entornos de CartPole y Pong.

Se han realizado distintas estrategias de selección de hiperparámetros, tanto para conocer la importancia que tiene cada uno de ellos en la obtención de un mejor resultado, como para comprobar cuál ha sido la mejor estrategia seguida.

El modelo dinámico del dron tuvo que ser discretizado ya que era continuo. En las acciones que realiza el dron para desplazarse, incrementaba o disminuía los momentos angulares y el empuje.

Esta alteración no podía ser demasiado alta ya que al generar cambios bruscos en las velocidades y aceleraciones hacía que el sistema se hiciese inestable.

El incremento de tiempo también supuso en un principio un problema ya que era demasiado alto y las ecuaciones que calculaban la posición, aceleración y velocidades nuevas, dependían de él.

Al final se llegó al valor óptimo del parámetro mediante varias pruebas.

Por otro lado se diseñó un modelo relativamente sencillo para conseguir un aprendizaje más rápido. Se realizó un intento de simulación en 3D pero debido a su complejidad y el desconocimiento de la implementación, no funcionó.

Finalmente se optó por la solución de representarlo en dos dimensiones, x e y.

Para poder ver lo que ocurre en el eje z, la altura a la que está el dron, se generó una gráfica que representaba la posición con respecto al tiempo.

Con esta representación se obtienen todos los datos necesarios para poder generar las funciones de recompensas y observaciones.

Al tener el dron una velocidad inicial nula, los primeros pasos son muy cortos y necesitaba muchas acciones para avanzar una distancia pequeña.

El entorno se ajusta perfectamente con la interfaz ya que tiene las entradas y salidas requeridas para implementarlo en una librería de Deep Reinforcement Learning, a falta de ajustar los hiperparámetros y ver si la función de recompensa es suficiente para conseguir un entrenamiento exitoso o es necesario implementar alguna modificación.

16. LINEAS FUTURAS

Para que el agente consiga aprender es necesario un estudio de las redes convolucionales.

El experimento no se ha llegado a probar con redes conv2D, por lo que no se sabe si sirven para el correcto aprendizaje del agente.

Tras crear el modelo y el entorno, no se conectado a la librería de Deep Reinforcement Learning, quedando pendiente para comprobar el aprendizaje del agente.

En un futuro se pretende estudiar las redes convolucionales 3D para poder probar y ajustar hiperparámetros y nuevos algoritmos que permitan un rápido aprendizaje del dron.

Existen dos métodos para la resolución de los entornos, síncronos y asíncronos.

En este estudio se ha enfatizado en métodos síncronos, que hacen uso de una memoria de repetición, explicado su funcionamiento en la figura 9.1, para guardar transiciones y evitar la posible correlación de datos. También se han probado los métodos asíncronos en los que las transiciones aleatorias van entrando directamente a la red con la ayuda de distintos agentes.

En algunos entornos, como el caso del Pong, se han utilizado métodos asíncronos como actor critic, A3C, obteniendo mejores puntuaciones que con los métodos síncronos. Por lo tanto hay que tenerlos en cuenta como línea futura de estudio.

A partir de estos algoritmos se pretende llegar a entrenar sistemas de control de vehículos, robots e incluso satélites, gracias a que son generales y se adaptan a distintos sistemas.

17. REFERENCIAS

[1] “Definición UAV”. [Online]. Available:

http://drones.uv.es/origen-y-desarrollo-de-los-drones/

[2] “The history of drone technology”. [Online]. Available:

http://www.redorbit.com/reference/the-history-of-drone-technology/

[3] ”Fulmar”. [Online]. Available:

http://infodron.es/id/2016/07/13/noticia-thales-espana-lanza-nueva-version-fulmar.html

[4] “UAV aplications”. [Online]. Available:

https://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/gps-och-matning/nkg2016/applications-of-uavs-and-drones.pdf

[5] “Normativa de drones en España”. [Online]. Available:

https://www.boe.es/boe/dias/2016/12/03/pdfs/BOE-A-2016-11481.pdf

[6] “Machine learning”. [Online]. Available:

https://books.google.es/books?id=LfDICgAAQBAJ&pg=PA36&lpg=PA36&dq=machine+learning+oficial+paper&source=bl&ots=QBW5BAcjgt&sig=1BTOIQY4eNblojUVC6revrYvinc&hl=es&sa=X&ved=0ahUKEwjAz5GmvLXWAhWBnBQKHbGPCesQ6AEIXzAJ#v=onepage&q=machine%20learning%20oficial%20paper&f=false

[7] ”GO for machine learning”. [Online]. Available:

https://github.com/sjwhitworth/golearn

[8] Mnih, V. et al. Human-level control through deep reinforcement learning.

Nature 518, 529–533 (2015).

[9] Schulman, John, Levine, Pieter. Trust region policy optimization.

In International Conference on Machine Learning (ICML), 2015a.

[10] Sutton, R. & Barto, A. Reinforcement Learning: An Introduction

(MIT Press, 1998).

[11] Assaf Hallak, Dotan Di-Castro, and Shie Mannor. Model selection in markovian

processes. In Proceedings of the 19th ACM SIGKDD international conference

on Knowledge discovery and data mining. ACM, 2013.

[12] Aijun Bai, Siddharth Srivastava, and Stuart Russell.

Markovian state and action abstractions for mdps via hierarchical mcts.

[13] Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y.

Policy gradient methods for reinforcement learning with function approximation. In NIPS, pp. 1057–1063, 2000.

[14] Degris, Thomas, Pilarski, Patrick M, and Sutton, Richard S.

Model-free reinforcement learning with continuous action in practice.

In American Control Conference (ACC), 2012, pp. 2177–2182. IEEE, 2012.

[15] Bertsekas, Dimitri P. Distributed dynamic programming.

Automatic Control, IEEE Transactions on, 27(3):610–616, 1982.

[16] Chavez, Kevin, Ong, Hao Yi, and Hong, Augustus.

Distributed deep q learning. Technical report, Stanford University, June 2015.

[17] Van Hasselt, H. Double Q-learning. NIPS, 23:2613–2621, 2010.

[18] Brian Sallans and Geoffrey E. Hinton.

Reinforcement learning with factored states and actions.

Journal of Machine Learning Research, 5:1063–1088, 2004.

[19] Baird, L.C. Advantage updating.

Technical Report WLTR-93-1146, Wright-Patterson Air Force Base, 1993.

[20] C. J. Watkins and P. Dayan. Q-learning. Machine Learning, 8(3):279–292, 1992

[21] Kingma, Diederik P. and Ba, Jimmy.

Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

[22] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning.

Nature, 521(7553):436–444, 2015.

[23] Riedmiller, Martin. Neural fitted q iteration first experiences with a data efficient

neural reinforcement learning method.

In Machine Learning: ECML 2005, pp. 317–328.

Springer Berlin Heidelberg, 2005.

[24] Tom Zahavy, Nir Ben Zrihem, and Shie Mannor.

Graying the black box: Understanding dqns.

Proceedings of the 33 rd International Conference on Machine Learning

(ICML-16), JMLR:volume48, 2016.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis

Antonoglou, Daan Wierstra, and Martin Riedmiller.

Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602,

2013.

[26] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling.

The arcade learning environment: An evaluation platform for general agents.

Journal of Artificial Intelligence Research, 47:253–279, 2013.

[27] Van Hasselt, H. Deep Reinforcement

Learning with Double Q-learning. December, 2015.[28] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures

for deep reinforcement learning. arXiv preprint arXiv:1511.06581, 2015.

[29] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay.

In ICLR, 2016.

[30] White, Adam, Modayil, Joseph, and Sutton, Richard S.

Surprise and curiosity for big data robotics. In Workshops at the Twenty-

Eighth AAAI Conference on Artificial Intelligence, 2014.

[31] Andrychowicz, M, Denil, M. Learning to learn by gradient descent by gradient

descent. November, 2016

[32] Mnih, Volodymyr, Kavukcuoglu, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. In NIPS Deep Learning Workshop. 2013.

18. BIBLIOGRAFIA

- “Unmanned Aerial Vehicles and Spatial Thinking: Boarding Education With Geotechnology And Drones”. http://ieeexplore.ieee.org/document/8038992/
- https://upcommons.upc.edu/bitstream/handle/2099.1/3358/36022-9.pdf?sequence=9&isAllowed=y
- https://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/gps-och-matning/nkg2016/applications-of-uavs-and-drones.pdf
- https://github.com/openai/baselines
- https://yilundu.github.io/2016/12/24/Deep-Q-Learning-on-Space-Invaders.html
- https://keras.io/layers/about-keras-layers/
- https://github.com/tensorflow/tensorflow
- https://www.anaconda.com/what-is-anaconda/
- https://github.com/openai/gym
- “Functional Contour-following via Haptic Perception and Reinforcement Learning”. http://ieeexplore.ieee.org/document/8039205/
- https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf

DOCUMENTO 2. ANEXOS

ÍNDICE DOCUMENTO 2. ANEXOS

1 ANEXO 1: SOFWARE

1.1 POO

Los paradigmas de programación iniciales se centraban en los procesos y la programación de funciones. Por contra, surgieron nuevos planteamientos que daban mayor importancia a los datos a tratar, surgiendo de esta manera las bases de datos.

La programación orientada a objetos supone la síntesis de ambos planteamientos. En ella los datos (atributos), y los procesos (métodos) no están separados, sino integrados dentro de objetos, que pertenecen a una clase. Estos objetos reciben los datos de entrada, y operan siguiendo unos métodos, para obtener los datos de salida.

Este planteamiento es ideal para el machine learning, pues tanto los datos de entrenamiento como las operaciones a realizar, tienen gran importancia.

Otras ventajas de la programación orientada a objetos son su gran modularidad, abstracción y encapsulamiento, de forma que cada los objeto no tiene acceso al código interno de otros objetos. Se previenen así modificaciones e interacciones no deseadas.

Gracias a la encapsulación, añadir y modificar objetos es una tarea sencilla (refactoring); pero cambiar la interfaz, con la que interactúa cada objeto, es más complicado, ya que afecta al resto de objetos del programa.

Por ello un buen análisis y diseño es imprescindible.

Mecanismos como la herencia permiten gran flexibilidad y aprovechamiento del código, al permitir relacionar los objetos con distintos grados de jerarquía, permitiendo que compartan funcionalidades sin tener que reescribir código.

Para la programación de los distintos experimentos y scripts, se ha utilizado el lenguaje de programación Python. Su sencillez, y aceptación en el mundo del aprendizaje automático, gracias a librerías como TensorFlow o scikit-learn, facilitan la iniciación en su estudio.

Tanto la programación orientada a objetos como el lenguaje de programación Python, han requerido una formación previa al ser conocimientos no impartidos en el grado.

1.2 API's PARA APRENDIZAJE AUTOMÁTICO

1.2.1 Tensorflow

Tensorflow es una librería de código abierto basada en un sistema de redes neuronales. Esto permite relacionar varios datos en red de manera simultánea, imitando el funcionamiento del cerebro humano.

Esta herramienta de software fue creada por Google en 2015 y gracias a su nivel de desarrollo y su versatilidad se ha extendido de manera exponencial.

Al constar de una arquitectura flexible, le permite con una única API implementar experimentos a una o más CPUs o GPUs.

Es complejo de manejar pero permite un mayor control de las redes.

1.2.2 Keras

En la mayoría de líneas de investigación de inteligencia artificial se ha utilizado Keras por comodidad.

Keras es una interfaz de programación de aplicaciones, API, de redes neuronales a alto nivel.

Se trata de una librería desarrollada en Python que se puede ejecutar en la parte superior de bibliotecas más complejas de manejar como Tensorflow, CNTK o Theano.

Keras se desarrolló con el fin de conseguir hacer una implementación de manera fácil y rápida.

Además se puede ejecutar sin problemas tanto en CPU y como en GPU y soporta todo tipo de redes neuronales.

Una de las principales ventajas de esta API es que las capas neuronales, las funciones de coste, los optimizadores, los esquemas de inicialización, las funciones de activación y la regularización son todos módulos autónomos que se pueden combinar entre ellos para crear nuevos modelos. Estos módulos son simples de agregar dentro de nuevas clases o funciones y se pueden configurar de forma sencilla. Lo que permite tunear fácilmente los optimizadores y capas, añadiendo o quitando los distintos módulos que lo componen.

Keras permite guardar y cargar la arquitectura del modelo, sin necesidad de registrar sus pesos y entrenamientos.

Si se quiere tener más control sobre la red u observar cómo trabaja la misma a lo largo del tiempo, es preferible utilizar Tensorflow. Siempre es posible integrar las dos API's en el mismo script.

Hay dos modelos de Keras:

- Modelo secuencial. Es una sucesión lineal de capas. El modelo debe saber el tipo de entrada, por esa razón la primera capa siempre va a ser un "input" en la que se introduzca la información necesaria.
- Modelo funcional. Crea un modelo a partir de un tensor de entrada y uno de salida. A diferencia del anterior que solo puede tener una salida, el modelo funcional permite obtener varias salidas, siendo útil para implementaciones de algoritmos como el Dueling o A3C.

1.2.3 OpenAI

Es una compañía de investigación en inteligencia artificial.

La misión de OpenAI es crear inteligencia artificial, ofreciendo de forma gratuita y abierta una serie de entornos y algoritmos de reinforcement learning, ya testados para que los desarrolladores puedan realizar y compartir experimentos de forma rápida y contrastada.

Gym es una biblioteca de código abierto que da acceso a una gran cantidad de entornos prediseñados creada por OpenAI. Dentro de esta biblioteca existen modelos de control como el "Frozen Leg", videojuegos de Atari, sistemas en 3D, etc.

Baselines es un conjunto de algoritmos de reinforcement learning desarrollados por OpenAI. Estos algoritmos están revisados y testados por un equipo de expertos. Son por ejemplo DQN, DDQN o A2C,

Durante mi proceso de formación se han utilizado éstas librerías para solucionar distintos juegos.

Además también se ha usado un framework denominado Primate, que pertenece a la empresa Monkey from the Future. Es en una plataforma para la definición y entrenamiento de agentes en Deep Reinforcement Learning.

1.3 CPU vs GPU

Los procesadores de los PCs pueden ser una CPU o una GPU.

Ambas dos están formadas principalmente de transistores que realizan operaciones matemáticas y leen datos en sistema binario.

La CPU está diseñada para el procesamiento en serie de datos y se utiliza para realizar todo tipo de cálculos. Necesita varios núcleos para poder ejecutar distintas tareas a la vez.

Por otro lado la GPU está mucho más preparada para tareas que requieren un alto grado de paralelismo, lo que le hace ser muy útil para realizar trabajos en entornos gráficos, de aquí que también se la conozca como procesador gráfico.

La GPU tiene miles de núcleos en su interior mucho más pequeños que los utilizados en la CPU, por esa razón está optimizada para procesar grandes cantidades de datos, siendo ideal para “machine learning”.

En el proyecto utilizamos GPUs Nvidia porque su estructura CUDA es en la que se basa la mayoría de librerías de inteligencia artificial.

1.4 ANACONDA

Es la plataforma de cálculo de datos de Python más popular.

Se trata de un código abierto para el procesamiento de datos a gran escala, análisis predictivo y computación científica, cuyo objetivo es simplificar la administración de paquetes, que son administradas por el sistema de gestión “conda”.

Durante el proyecto, se ha utilizado ésta plataforma, ya que incorpora paquetes fundamentales para las operaciones y desarrollo del modelo y entorno, como Numpy, Tensorflow o Scipy.

También contiene paquetes para poder hacer gráficas de los resultados, como por ejemplo Matplotlib.

Además, al requerir el proyecto de un gran número de librerías relacionadas entre ellas, y que en ocasiones necesitan de distintas versiones para funcionar, Anaconda permite el uso de cada una de ellas en tareas concretas. Por ejemplo, un mismo script puede necesitar de Python 2.7 y 3.5 para funcionar.

1.5 SERVIDORES

Los primeros experimentos realizados llevaban poca carga computacional ya que eran entornos simples desarrollados para pruebas iniciales, por lo que no era necesario el uso de servidores externos.

A medida que se implementaban entornos más complejos como CartPole o el Pong, el tiempo de entrenamiento se hacía demasiado elevado para poder progresar de forma eficaz, por lo que ha sido necesario el uso nuevos servidores más potentes.

Para desarrollar este trabajo se han utilizado dos ordenadores nuevos que se encargaban de realizar los entrenamientos de las redes neuronales a través de GPU.

1.6 LINUX

Debido a que la mayoría de paquetes, librerías y programas de inteligencia artificial aún no están disponibles para Windows, en el proyecto se ha utilizado el sistema operativo Linux.

Lo que ha supuesto el estudio y familiarización con este sistema operativo.

1.7 TIPOS DE DATOS

Durante la programación de los algoritmos y entornos ha sido necesario el uso de distintos tipos de datos.

Las clases de datos utilizadas han sido:

- Int: Sólo guarda variables de tipo entero.
- Float: Puede almacenar tanto números enteros como decimales. En caso de que no sea preciso utilizar decimales es mejor usar el “int” ya que ocupa menos memoria y se consigue la ejecución del código más rápido.
- Str: Un string guarda una cadena de caracteres que pueden ser letras, puntos, números, espacios, etc.
- Bool: Guarda dos tipos de valores, True o False. Esta variable es muy utilizada en bucles.
- Listas: Son un conjunto ordenado de elementos. Los elementos pueden ser de distinto tipo, como por ejemplo otras listas.
- Tupla: Es un conjunto de valores inmutables. Consume menos espacio que una lista. Se define de la misma manera, salvo que el conjunto se encierra entre paréntesis en lugar de corchetes.

También se ha utilizado un paquete científico denominado Numpy, para simplificar las operaciones matemáticas con un estilo Matlab, y que trae sus propios tipos de datos como el “Numpy array”.

2 ANEXO 2: CÓDIGO DE PROGRAMACIÓN

2.1 CONTENIDOS

A continuación aparece el código utilizado para cada uno de los experimentos.

En el caso del Maze, consta de varios archivos que deben estar en la misma carpeta y que funcionan llamando al archivo dqn.

2.2 RANDOMWALK

```
import random

# Calcula el siguiente estado a partir del actual y la accion
def step(state,action):
    if action == 1 and state<9:
        state += 1
    elif action == 0 and state>0:
        state -= 1
    return state

# Calcula la recompensa segun el estado
def reward(state):
    if state == 0:
        return -1.0
    elif state == 9:
        return 1.0
    else:
        return 0.0

def is_goal(state):
    if state == 9 or state == 0:

        return True
    else:

        return False

#Inicializamos los estados y la funcion de utilidad
def reset():
    state = 5
    return state


def initQ():
    Qlist = []
    for i in range(10):
        Qlist.append([0, 0])
    return(Qlist)

def maxList(Qlist, state):
    return (max(Qlist[state]))


#Calculamos el target
def target(discount, rewardValue, nextRewardValue, state):
```

```
    if state == 9 or state == 0:
        return(rewardValue)
    else:
        return (rewardValue + discount * nextRewardValue)


#Calculamos la utilidad
def valueUpdate(targetValue, lrate, Qlist, state, action):
    return(Qlist[state][action]+lrate*((targetValue)-Qlist[state][action]))

def main():
    #Inicializamos
    Qlist = initQ()
    discount = 0.9
    lrate = 0.1
    episodes = 100
    random.seed()
    end = False
    for i in range(episodes):
        state, end = reset(), False
        while end is False:
            #Calculamos la recompensa actual
            point = reward(state)
            action = random.randint(0, 1)
            nextState = step(state, action)
            #Caculamos la recompensa futura
            Q = maxList(Qlist, nextState)
            #Calculamos la utilidad
            targetValue = target(discount, point, Q, state)
            utility = valueUpdate(targetValue, lrate, Qlist, state, action)
            # Qlist[state][action] = utility
            Qlist[state][action] = round(utility, 2)
            # print(Qlist)
            end = is_goal(state)
            state = nextState
    print('')
    print('Valores con 100 episodios')
    print(Qlist)

if __name__ == '__main__':
    main()
```

2.3 LABERINTO (MAZE)

2.3.1 Environment

```
class EnvironmentDecorator:
    def __init__(self, environment):
        self.decorated = environment

    @property
    def observation_space(self):
        return self.decorated.observation_space

    @property
    def action_space(self):
        return self.decorated.action_space

    def reset(self):
```

```
        return self.decorated.reset()

    def step(self, action):
        return self.decorated.step(action)
```

2.3.2 Maze

```
from gym import spaces
import random


class Maze:
    ROWS = 3
    COLS = 4
    ACTIONS = 4

    def __init__(self):
        self.env_state = None
        self.observation_space = spaces.Discrete(self.ROWS * self.COLS)
        self.action_space = spaces.Discrete(self.ACTIONS)

    @property
    def action_count(self):
        return self.action_space.n

    def get_observation(self):
        if self.env_state is not None:
            row, col = self.env_state
            return self.COLS * row + col
        else:
            return None

    def reset(self):
        self.env_state = (2, 0)
        return self.get_observation()

    def get_reward(self):
        """ It just depends con current state. """
        if self.env_state == (0, 3):
            return 1.0
        elif self.env_state == (1, 3):
            return -1.0
        else:
            return 0.0

    def get_motion(self, action):
        n = random.random()
        if n < 0.1:
            out = action - 1
        elif n < 0.2:
            out = action + 1
        else:
            out = action
        if out == -1:
            out = self.ACTIONS - 1
        elif out == self.ACTIONS:
            out = 0
        return out

    def next_state(self, action):
        if self.is_terminal_state():
```

```
            return None
        else:
            obstacle = (1, 1)
            motion = self.get_motion(action)
            motion_list = [(0, -1), (-1, 0), (0, 1), (1, 0)]
            (row, col) = self.env_state
            next_row = row + motion_list[motion][0]
            next_col = col + motion_list[motion][1]
            if next_col >= self.COLS or next_row >= self.ROWS or next_col <
0 or next_row < 0 or (next_row, next_col) == obstacle:
                return row, col
            else:
                return next_row, next_col

    def is_terminal_state(self):
        if self.env_state == (0, 3) or self.env_state == (1, 3):
            return True
        else:
            return False

    def step(self, action):
        done = self.is_terminal_state()
        reward = self.get_reward()
        self.env_state = self.next_state(action)
        info = {}
        return self.get_observation(), reward, done, info
```

2.3.3 Replay memory

```
import collections as coll
import random

class ReplayMemory(object):
    def __init__(self, max_size=None, max_episode_steps=None):
        self.transitions = coll.deque(maxlen=max_size)
        self.current_episode = (None if max_episode_steps is None
                                else coll.deque(maxlen=max_episode_steps))

    def append(self, transition):
        if self.current_episode is None:
            self.transitions.append(transition)

            return

        self.current_episode.append(transition)
        (state, action, reward, next_state, done) = transition
        if done:
            self.transitions.extend(self.current episode)
            self.current_episode =
coll.deque(maxlen=self.current_episode.maxlen)

    @property
    def size(self):
        return len(self.transitions)

    @property
    def max_size(self):
        return self.transitions.maxlen

    @property
    def full(self):
```

```
        if self.max_size is None:
            return False

        return self.size >= self.max_size

    def get_batch(self, batch_size):
        return random.sample(self.transitions, batch_size)
```

2.3.4 Qnetwork

```
import numpy as np
from keras.models import load_model


class AbstractQNetwork:
    """ Abstract class for Neural Networks used in Q-Learning.
    The only method that must be implemented is create_model(), which
should create the Keras
    model describing the neural network or use the load() method to load it
from a file.
    """

    def __init__(self, environment):
        self.environment = environment
        self.model = self.create_model()

    def create_model(self):
        """ Creates the model.

        It can create the model from scratch using Keras classes, or load
it using the load()
        method."""
        raise NotImplementedError("A subclass must implement this method")

    @property
    def observation_space(self):
        return self.environment.observation_space

    @property
    def action_space(self):
        return self.environment.action_space

    @property
    def weights(self):
        return self.model.get_weights()

    @weights.setter
    def weights(self, weights):
        self.model.set_weights(weights)

    def load(self, filepath):
        """ It loads a Keras model from an h5 file. """
        self.model = load_model(filepath)

    def save(self, filepath):
        """ It saves the current Keras model into as an h5 file. """
        self.model.save(filepath)

    def train(self, training_data, target_data):
        """ It trains the neural network using a batch of training data and
associated
```

```
        target data:
            * training_data: list of state-action pair lists [[s1, a1],
[s2, a2], ...]
            * target_data: list of QValues associated to each state-action
pair [Q1, Q2, ... ]
        """
        self.model.fit(training_data, target_data, batch_size=64, epochs=1,
verbose=0)

    def qvalues(self, state):
        """ It returns the approximate QValues for each action for a state.
"""
        state = np.reshape(state, (1, -1))

        return self.model.predict(state)

    def state_value(self, state):
        """ It returns the approximate value of a state, calculated as the
max
        of the approximate QValues of (s, a) over all available actions.
"""
        return np.amax(self.qvalues(state))

    def greedy_action(self, state):
        """ It returns the greedy action of a state. The greedy action is
the
        action related to the highest QValue of the state. """
        return np.argmax(self.qvalues(state))

    def state_preproc(self, s):
        """ It converts a state (A tuple) into a list. Requirement to be
part of
        the Neural Network input. """
        return list(s)


    def predict(self, states):
        """ It returns the approximate QValues for each action for a list
of states. """
        return self.model.predict(states)
```

2.3.5 Dqn

```
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import *
from keras.optimizers import *
from maze import Maze

from qnetwork import AbstractQNetwork
from replay_memory import ReplayMemory
from enviroment import EnvironmentDecorator


EPSILON = 1.0
REPLAY_MEMORY_SIZE = 1000
REPLAY_MEMORY_BATCH_SIZE = 64
TRAINER_GAMMA = 0.90
LEARNING_STEPS = 1200
MAX_SIZE = 1000
```

```
class SimpleQNetwork(AbstractQNetwork):
    STATE_SIZE = 1

    def __init__(self, environment):
        AbstractQNetwork.__init__(self, environment)

    def create_model(self):
        model = Sequential()
        model.add(Dense(units=128, activation='relu',
input_dim=self.STATE_SIZE))
        model.add(Dense(units=128, activation='relu'))
        model.add(Dense(units=128, activation='relu'))
        model.add(Dense(units=self.environment.action_count,
activation='linear'))
        model.compile(loss='mse', optimizer=RMSprop(lr=0.005))
        return model


class SimpleEnvironment(EnvironmentDecorator):
    def __init__(self):
        env = Maze()
        EnvironmentDecorator.__init__(self, env)
        self._action_space = env.action_space

    @property
    def action_space(self):
        return self._action_space

    @property
    def observation_count(self):
        return 1

    @property
    def action_count(self):
        return self.action_space.n


def act(state, env, qnetwork):
    # Exploration strategy. Random vs Greedy
    if np.random.rand() <= EPSILON:
        # TODO- sample doesn't work
        # return env.action_space.sample
        return np.random.randint(4)
    else:
        return qnetwork.greedy_action(state)


def RN_data(batch, qnetwork):
    q_values = []
    state_values = []
    for i in range(REPLAY_MEMORY_BATCH_SIZE):
        state = batch[i][0]
        reward = batch[i][2]
        done = batch[i][4]
        next_state = batch[i][3]
        state_values.append(state)
        next_reward = qnetwork.state_value(next_state)
        if done:
            target = reward
        else:
```

```
            target = reward + TRAINER_GAMMA * next_reward
        q_array = qnetwork.qvalues(state)
        q_array = [q_array[0][0], q_array[0][1], q_array[0][2],
q_array[0][3]]
        index = np.argmax(q_array)
        q_array[index] = target
        q_values.append(list(q_array))
    return state_values, q_values

def main():
    target_list3 = []
    env = SimpleEnvironment()
    qnetwork = SimpleQNetwork(env)
    replay_memory = ReplayMemory(MAX_SIZE)
    value_state_3 = []
    value_state_2 = []
    value_state_8 = []
    # TODO - state 3 reward
    for i in range(LEARNING_STEPS):
        state = env.reset()
        done = False
        while done is False:
            action = act(state, env, qnetwork)
            next_state, reward, done, info = env.step(action)
            transition = [state, action, reward, next_state, done]
            replay_memory.append(transition)
            # TODO - if replay_memory.full:
            if i > 1000:
                if i % 5 == 0:
                    print(i)
                batch = replay_memory.get_batch(REPLAY_MEMORY_BATCH_SIZE)
                state_values, q_values = RN_data(batch, qnetwork)
                qnetwork.train(state_values, q_values)
                value_state_2.append(qnetwork.state_value(2))
                value_state_3.append(qnetwork.state_value(3))
                value_state_8.append(qnetwork.state_value(8))
            state = next_state
    #print("iteration: {}  target: {}".format(i, target_list[11]))
    plt.plot(value_state_3)
    plt.plot(value_state_2)
    plt.plot(value_state_8)
    plt.show()

if __name__ == '__main__':
    main()
```

2.4 MODELO

```
import numpy as np
from matplotlib import pyplot as plt


class System:
    def __init__(self, x, y, z, roll, pitch, yaw, Vx, Vy, Vz, p, q, r, T,
roll moment, pitch moment, yaw moment):
        # Defino las constantes que van a aparecer
        self.m = 0.468  # masa del dron en Kg
        self.g = 9.81  # gravedad en m/s^2
        self.Ax = 0.25  # kg/s
        self.Ay = 0.25  # kg/s
```

```
        self.Az = 0.25  # Kg/s
        self.Ixx = 4.856e-3  # kg*m^2
        self.Iyy = 4.856e-3  # kg*m^2
        self.Izz = 8.801e-3  # Kg*m^2
        self.J = np.array([[self.Ixx, 0, 0], [0, self.Iyy, 0], [0, 0,
self.Izz]])
        self.l = 0.225  # dist. del centro del rotor al centro de masa del
dron
        self.k = 2.98e-6  # lift
        self.b = 1.14e-7  # drag
        self.increase_time = 0.0001
        # Valores iniciales de posición y orientación.
        self.pos = np.array([[x], [y], [z]], dtype=float)
        self.orientation = np.array([[roll], [pitch], [yaw]], dtype=float)
        self.Vl_cm = np.array([[Vx], [Vy], [Vz]], dtype=float)
        self.v_cm = np.array([[p], [q], [r]], dtype=float)
        self.Vl_i = np.array([[3], [4], [3]], dtype=float)
        self.T = T
        self.roll_moment = roll_moment
        self.pitch_moment = pitch_moment
        self.yaw_moment = yaw_moment
        # Listas
        self.x_list = []
        self.time_list = []
        self.y_list = []
        self.z_list = []
        self.Vx_list = []
        self.Vy_list = []
        self.Vz_list = []
        self.roll_list = []
        self.pitch_list = []
        self.yaw_list = []
        self.p_list = []
        self.q_list = []
        self.r_list = []
        self.save_list = []
        return

    def position(self):
        self.C_Roll = float(np.cos(np.deg2rad(self.orientation[0])))
        self.S_Roll = float(np.sin(np.deg2rad(self.orientation[0])))
        self.C_Pitch = float(np.cos(np.deg2rad(self.orientation[1])))
        self.S_Pitch = float(np.sin(np.deg2rad(self.orientation[1])))
        self.T_Pitch = float(np.tan(np.deg2rad(self.orientation[1])))
        self.C_Yaw = float(np.cos(np.deg2rad(self.orientation[2])))
        self.S_Yaw = float(np.sin(np.deg2rad(self.orientation[2])))
        return

    def rotation_matrix(self):
        r_i = np.array([[self.C_Yaw, -self.S_Yaw, 0],
                        [self.S_Yaw, self.C_Yaw, 0],
                        [0, 0, 1]])  # Rotación 'Yaw' respecto sistema de
referencia inercial
        r_1 = np.array([[self.C_Pitch, 0, self.S_Pitch],
                        [0, 1, 0],
                        [-self.S_Pitch, 0, self.C_Pitch]])  # Rotacion
'Pitch'
        r_cm = np.array([[1, 0, 0],
                         [0, self.C_Roll, self.S_Roll],
                         [0, self.S_Roll, self.C_Roll]])  # Rotacion
'Roll', el nuevo eje es el centro de masa
```

```
        self.r_total = r_cm * r_1 * r_i
        return

    def trans_matrix(self):
        self.transformation = np.array([[1, self.S_Roll * self.T_Pitch,
self.C_Roll * self.T_Pitch],
                                        [0, self.C_Roll, -self.S_Roll],
                                        [0, self.S_Roll / self.C_Pitch,
self.C_Roll / self.C_Pitch]])
        self.inv_transformation = np.array([[1, 0, -self.S_Pitch],
                                            [0, self.C_Roll,
self.C_Pitch*self.S_Roll],
                                            [0, -self.S_Roll,
self.C_Pitch*self.C_Roll]])
        return

    def velocities(self):
        self.n = np.dot(np.array([[1, self.S_Roll * self.T_Pitch,
self.C_Roll * self.T_Pitch],
                                  [0, self.C_Roll, -self.S_Roll],
                                  [0, self.S_Roll / self.C_Pitch,
self.C_Roll / self.C_Pitch]]), self.v_cm)  # velocidad angular en el
sistema inercial
        self.Vl_cm = np.dot(self.r_total, self.Vl_i)
        self.Vl_i = np.dot(self.r_total, self.Vl_cm)  # Velocidad lineal
repecto sitema inercial

        return

    def control(self):

        """Aceleración lineal respecto el centro de masa. Primero calculado
sin incluir
        la fuerza de arrastre y después añadiéndola"""
        self.m11 =
self.C_Yaw*self.S_Pitch*self.C_Roll+self.S_Yaw*self.S_Roll
        self.m21 = self.S_Yaw*self.S_Pitch*self.C_Roll-
self.C_Yaw*self.S_Roll
        self.m31 = self.C_Pitch*self.C_Roll
        self.acel_i1 = -self.g*np.array([[0], [0],
[1]])+(self.T/self.m)*(np.array([[self.m11],

[self.m21],

[self.m31]]))  # Acel. lineal respecto sitema inercial

        self.acel_i = self.acel_i1-(1/self.m)*np.dot(np.array([[self.Ax, 0,
0], [0, self.Ay, 0], [0, 0, self.Az]]), self.Vl_i)  # Añadiendo fuerza de
arrastre

        """Cálculo de la aceleracion angular respecto del centro de masa.
        Es necesaria para poder hallar posteriormente la acel. angular
respecto del sistema inercial"""

        self.a = np.array([[self.v_cm[1][0] / self.Ixx], [-self.v_cm[0][0]
/ self.Iyy], [0]])
        self.product = np.dot(self.J, self.a)  # Parte de la siguiente
ecuación
        self.b11 = float(self.Iyy-
self.Izz)*self.v_cm[1]*self.v_cm[2]/self.Ixx
        self.b21 = float(self.Izz-
```

```
self.Ixx)*self.v_cm[0]*self.v_cm[2]/self.Iyy
        self.b31 = float(self.Ixx-
self.Iyy)*self.v_cm[0]*self.v_cm[1]/self.Izz

        self.d = -self.product + np.array([[self.roll_moment / self.Ixx],
                                           [self.pitch_moment / self.Iyy],
                                           [self.yaw_moment / self.Izz]])
        self.c = np.array([self.b11, self.b21, self.b31], dtype=np.float64)
        self.acel_cm = self.c + self.d
        """Cálculo de la aceleración angular respecto sistema inercial.
Segunda derivada de [pitch, roll, yaw]"""
        self.a11 = 0
        self.a12 = self.n[0] * self.C_Roll
        self.a12 = self.a12 * self.T_Pitch + (self.n[1] * self.S_Roll /
(self.C_Pitch * self.C_Pitch))
        self.a13 = -self.n[0]
        self.a13 = self.a13 * self.S_Roll * self.C_Pitch
        self.a13 = self.a13 + (self.n[1] * self.C_Roll / (self.C_Pitch *
self.C_Pitch))
        self.a21 = 0
        self.a22 = -self.n[0] * self.S_Roll
        self.a23 = -self.n[0] * self.C_Roll
        self.a31 = 0
        self.a32 = self.n[0] * self.C_Roll / self.C_Pitch
        self.a32 = self.a32 + (self.n[0] * self.S_Roll * self.T_Pitch /
self.C_Pitch)
        self.a33 = -self.n[0] * self.S_Roll / self.C_Pitch
        self.a33 = self.a33 + (self.n[1] * self.C_Roll * self.T_Pitch /
self.C_Pitch)

        self.acel_angi = np.dot(np.array([[self.a11, self.a12, self.a13],
                                          [self.a21, self.a22, self.a23],
                                          [self.a31, self.a32, self.a33]]),
self.v_cm) + np.dot(self.inv_transformation, self.acel_cm)

        return

    def motion_equations(self):
        self.new_pos =
self.pos+self.Vl_i*self.increase_time+0.5*self.acel_i*self.increase_time**2
# Nueva pos.lineal
        self.new_vel = self.Vl_i+self.acel_i*self.increase_time  # Nueva
velocidad lineal
        self.new_orientation =
self.orientation+self.v_cm*self.increase_time+0.5*self.acel_angi*self.incre
ase_time**2 # Nueva orientación
        self.new_vel_ang = self.v_cm+self.acel_angi*self.increase_time  #
Nueva velocidad angular

        return

    def actions(self, action):
        # action = float(input("Elige un acción: ")) Descomentar para
probar modelo
        if action == 0:
            self.T = self.T + 0.001
        elif action == 1:
            self.T = self.T - 0.001
        elif action == 2:
            self.roll_moment = self.roll_moment + 0.001
        elif action == 3:
```

```
            self.roll_moment = self.roll_moment - 0.001
        elif action == 4:
            self.pitch_moment = self.pitch_moment + 0.001
        elif action == 5:
            self.pitch_moment = self.pitch_moment - 0.001
        elif action == 6:
            self.yaw_moment = self.yaw_moment + 0.001
        elif action == 7:
            self.yaw_moment = self.yaw_moment - 0.001
        else:
            pass

    def reset(self):
        self.x = 1
        self.y = 1
        self.z = 1
        self.roll = 3.1780
        self.pitch = 1.1745
        self.yaw = 0.1745
        self.Vx = 1
        self.Vy = 1
        self.Vz = 1
        self.p = 0.5
        self.q = 0.5
        self.r = 0.5
        self.T = 8
        self.roll_moment = 0
        self.pitch_moment = 0
        self.yaw_moment = 0
        return

    def render(self):
        plt.figure()
        print('Posición x')
        print(self.x_list)
        print('Posición y')
        print(self.y_list)
        print('Posición z')
        print(self.z_list)
        print('Tiempo')
        print(self.time_list)
        print('Velocidad x')
        print(self.Vx_list)
        print('Velocidad y')
        print(self.Vy_list)
        print('Velocidad z')
        print(self.Vz_list)
        plt.subplot(221)
        plt.xlabel("Tiempo")  # Establece el título del eje x
        plt.ylabel("Posicion")  # Establece el título del eje y
        plt.plot(self.time_list, self.x_list, 'r', label="Posicion x")
        plt.plot(self.time_list, self.y_list, 'b', label="Posicion y")
        plt.plot(self.time_list, self.z_list, 'g', label="Posicion z")
        plt.legend()
        plt.grid(True)
        plt.subplot(222)
        plt.xlabel("Tiempo")
        plt.ylabel("Orientacion")
        plt.plot(self.time_list, self.roll_list, 'r', label="Roll")
        plt.plot(self.time_list, self.pitch_list, 'b', label="Pitch")
        plt.plot(self.time_list, self.yaw_list, 'g', label="Yaw")
```

```
        plt.legend()
        plt.grid(True)
        # plt.tight_layout()
        plt.subplot(223)
        plt.xlabel("Tiempo")
        plt.ylabel("Velocidad lineal")
        plt.plot(self.time_list, self.Vx_list, 'r', label="Velocidad x")
        plt.plot(self.time_list, self.Vy_list, 'b', label="Velocidad y")
        plt.plot(self.time_list, self.Vz_list, 'g', label="Velocidad z")
        plt.legend()
        plt.grid(True)
        plt.subplot(224)
        plt.xlabel("Tiempo")
        plt.ylabel("Velocidad angular")
        plt.plot(self.time_list, self.p_list, 'r', label="Velocidad p")
        plt.plot(self.time_list, self.q_list, 'b', label="Velocidad q")
        plt.plot(self.time_list, self.r_list, 'g', label="Velocidad r")
        plt.legend()
        plt.grid(True)
        plt.show()

    def lists(self):
        self.x_list.append(float(self.pos[0]))
        self.y_list.append(float(self.pos[1]))
        self.z_list.append(float(self.pos[2]))
        self.Vx_list.append(float(self.Vl_i[0]))
        self.Vy_list.append(float(self.Vl_i[1]))
        self.Vz_list.append(float(self.Vl_i[2]))
        self.roll_list.append(float(self.orientation[0]))
        self.pitch_list.append(float(self.orientation[1]))
        self.yaw_list.append(float(self.orientation[2]))
        self.p_list.append(float(self.v_cm[0]))
        self.q_list.append(float(self.v_cm[1]))
        self.r_list.append(float(self.v_cm[2]))

    def first_values(self):
        self.position()
        self.rotation_matrix()
        self.trans_matrix()
        self.velocities()
        self.control()

    def compute(self):
        self.lists()
        self.actions()
        self.motion_equations()
        self.pos = self.new_pos
        self.Vl_i = self.new_vel
        self.orientation = self.new_orientation
        self.v_cm = self.new_vel_ang
        self.position()
        self.rotation_matrix()
        self.trans_matrix()
        self.velocities()
        self.control()
        self.save_list.append([self.pos, self.Vl_i, self.orientation,
self.v_cm])
        self.time_list.append(self.increase_time)
        self.increase_time = self.increase_time + 0.0001
```

```
def main():
    modelo = System(0, 0, 1, 0.1745, 0.1745, 0.1745, 0, 0, 0, 0, 0, 0, 4.2,
0, 0, 0)
    modelo.first_values()
    for i in range(0, 200):
        modelo.compute()
    modelo.render()

if __name__ == "__main__":
    main()
```

2.5 ENTORNO

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.patches as patches
from Model import System


class Drone:
    def __init__(self, pos=(1, 1), radius=0.4):
        self.pos = pos
        self.radius = radius
        return


class Obstacle:
    def __init__(self, pos=(0, 0), width=1, length=1):
        self.pos = pos
        self.width = width
        self.length = length


class End:
    def __init__(self, position, radius=1):
        self.position = position
        self.radius = radius


class Env:

    def __init__(self, lim_x=8, lim_y=8, reward=0, j=4):
        self.lim_x = lim_x
        self.lim_y = lim_y
        self.j = j
        self.reward = reward
        self.model = System(1, 1, 1, 3.1780, 1.1745, 0.1745, 1, 1, 1, 0.5,
0.5, 0.5, 8, 0, 0, 0)
        self.drone = Drone(self.model)
        self.end = End((6.5, 6.5))
        self.obs = Obstacle((3.5, 3.5), 1, 1)
        return

    def step(self, action):
        self.model.compute(action)
        return self.observation(), self.reward(), self.done(), self.info()

    def observation(self):
        return [self.drone.pos, self.model.orientation, self.model.Vl_i,
self.model.v_cm]
```

```
    def reward(self, drone, end, model, obs):
        drone.pos = (model.pos[0], model.pos[1])
        a = (obs.pos[0] - drone.radius <= drone.pos[0])
        b = (drone.pos[0] <= obs.pos[0] + drone.radius + obs.length)
        c = (drone.pos[1] >= obs.pos[1] - drone.radius)
        d = (drone.pos[1] <= obs.pos[1] + drone.radius + obs.width)
        if model.pos[0] - drone.radius <= 0 or model.pos[0] + drone.radius
>= 8:
            self.reward = -50
        elif model.pos[1] - drone.radius <= 0 or model.pos[1] +
drone.radius >= 8:
            self.reward = -50
        elif model.pos[2] - drone.radius <= 0 or model.pos[2] +
drone.radius >= 20:
            self.reward = -50
        elif np.sqrt((end.position[0] - drone.pos[0]) ** 2 +
(end.position[1] - drone.pos[1]) ** 2) <= (
            end.radius - drone.radius):
            self.reward = 50
        elif (a and b) and (c and d):
            self.reward = -50
        else:
            self.reward = -1
        return self.reward

    def reset(self):
        self.model.reset()
        return self.observation()

    def done(self):
        if self.reward == -50 or self.reward == 50:
            plt.pause(0.5)
            return True
        else:
            return False

    def info(self):
        pass

    def drone_positions(self, model):
        plt.figure()
        plt.subplot(311)
        plt.plot(model.time_list, model.x_list, 'r', label="Posicion x")
        plt.legend()
        plt.grid(True)
        plt.subplot(312)
        plt.plot(model.time_list, model.y_list, 'b', label="Posicion y")
        plt.legend()
        plt.grid(True)
        plt.ylabel("Posición")
        plt.subplot(313)
        plt.plot(model.time_list, model.z_list, 'g', label="Posicion z")
        plt.legend()
        plt.grid(True)
        plt.xlabel("Tiempo")
        plt.show(block=True)

    def render(self, drone, end, model, obs):
        if self.j % 4 == 0:
            fig = plt.figure()
```

```
            ax1 = fig.add_subplot(111, aspect='equal')
            ax1.set_xlim([0, self.lim_x])
            ax1.set_ylim([0, self.lim_y])
            ax1.add_patch(patches.Circle(model.pos, drone.radius,
color='green'))
            ax1.add_patch(patches.Rectangle(obs.pos, obs.width, obs.length,
edgecolor='black'))
            ax1.add_patch(patches.Circle(end.position, end.radius,
color='crimson'))
            plt.ion()
            plt.pause(.001)
            plt.show()
            print('J')
            print(self.j)
        self.j = self.j+1


def main():
    environment = Env()
    print('Reward: ')
    print(environment.reward)
    for i in range(0, 500):
        action = float(input("Elige un acción: "))
        environment.render(environment.drone, environment.end,
environment.model, environment.obs)
        environment.reward()
        environment.step(action)
        print('Reward: ')
        print(environment.reward)


if __name__ == "__main__":
    main()
```